

```
# Makefile for all device drivers.
```

```
#
```

```
MAKE = exec make -$(MAKEFLAGS)
```

**usage:**

```
@echo " " >&2
@echo "Makefile for all device drivers." >&2
@echo "Usage:" >&2
@echo " make build   # Compile all device drivers locally" >&2
@echo " make image  # Compile drivers in boot image" >&2
@echo " make clean   # Remove local compiler results" >&2
@echo " make install # Install drivers to /etc/drivers/" >&2
@echo "                (requires root privileges)" >&2
@echo " " >&2
```

**build:** all

all install depend clean:

```
cd ./libdriver && $(MAKE) $@
cd ./tty && $(MAKE) $@
cd ./at_wini && $(MAKE) $@
cd ./floppy && $(MAKE) $@
cd ./printer && $(MAKE) $@
cd ./rtl8139 && $(MAKE) $@
cd ./fxp && $(MAKE) $@
cd ./dpeth && $(MAKE) $@
cd ./log && $(MAKE) $@
cd ./bios_wini && $(MAKE) $@
cd ./cmos && $(MAKE) $@
cd ./random && $(MAKE) $@
cd ./dp8390 && $(MAKE) $@
cd ./sb16 && $(MAKE) $@
cd ./lance && $(MAKE) $@
cd ./pci && $(MAKE) $@
cd ./til225 && $(MAKE) $@
cd ./memory && $(MAKE) $@           # Must be last for ramdisk image
```

**image:**

```
cd ./libdriver && $(MAKE) build
cd ./tty && $(MAKE) build
cd ./at_wini && $(MAKE) build
cd ./floppy && $(MAKE) build
cd ./bios_wini && $(MAKE) build
cd ./log && $(MAKE) build
cd ./pci && $(MAKE) build
cd ./memory && $(MAKE) build       # Must be last for ramdisk image
```

```
/* This is the master header for all device drivers. It includes some other
 * files and defines the principal constants.
 */
#define _POSIX_SOURCE      1      /* tell headers to include POSIX stuff */
#define _MINIX             1      /* tell headers to include MINIX stuff */
#define _SYSTEM           1      /* get negative error number in <errno.h> */

/* The following are so basic, all the *.c files get them automatically. */
#include <minix/config.h>      /* MUST be first */
#include <ansi.h>              /* MUST be second */
#include <minix/type.h>
#include <minix/com.h>
#include <minix/dmap.h>
#include <minix/callnr.h>
#include <sys/types.h>
#include <minix/const.h>
#include <minix/devio.h>
#include <minix/syslib.h>
#include <minix/sysutil.h>
#include <minix/bitmap.h>

#include <ibm/interrupt.h>     /* IRQ vectors and miscellaneous ports */
#include <ibm/bios.h>          /* BIOS index numbers */
#include <ibm/ports.h>        /* Well-known ports */

#include <string.h>
#include <signal.h>
#include <stdlib.h>
#include <limits.h>
#include <stddef.h>
#include <errno.h>
#include <unistd.h>
```

```
# Makefile for the AT disk driver (AT_WINI)
DRIVER = at_wini

# directories
u = /usr
i = $u/include
s = $i/sys
m = $i/minix
b = $i/ibm
d = ..
p = ../libpci

# programs, flags, etc.
MAKE = exec make
CC = exec cc
CFLAGS = -I$i
LDFLAGS = -i
LIBS = -lsysutil -lsys -ltimers

OBJ = at_wini.o
LIBDRIVER = $d/libdriver/driver.o $d/libdriver/drvlib.o

# build local binary
all build: $(DRIVER)
$(DRIVER): $(OBJ) $(LIBDRIVER)
    $(CC) -o $@ $(LDFLAGS) $(OBJ) $(LIBDRIVER) $(LIBS)
    install -S 8k $(DRIVER)

$(LIBDRIVER):
    cd $d/libdriver && $(MAKE)

# install with other drivers
install: /sbin/$(DRIVER)
/sbin/$(DRIVER): $(DRIVER)
    install -o root -cs $? $@

# clean up local files
clean:
    rm -f $(DRIVER) *.o *.bak

depend:
    /usr/bin/mkdep "$(CC) -E $(CPPFLAGS)" *.c ../libdriver/*.c > .depend

# Include generated dependencies.
include .depend
```

```

/* This file contains the device dependent part of a driver for the IBM-AT
 * winchester controller.  Written by Adri Koppes.
 *
 * The file contains one entry point:
 *
 *   at_winchester_task:           main entry when system is brought up
 *
 * Changes:
 *   Aug 19, 2005   ATA PCI support, supports SATA (Ben Gras)
 *   Nov 18, 2004   moved AT disk driver to user-space (Jorrit N. Herder)
 *   Aug 20, 2004   watchdogs replaced by sync alarms (Jorrit N. Herder)
 *   Mar 23, 2000   added ATAPI CDRom support (Michael Temari)
 *   May 14, 2000   d-d/i rewrite (Kees J. Bot)
 *   Apr 13, 1992   device dependent/independent split (Kees J. Bot)
 */

#include "at_wini.h"

#include <minix/sysutil.h>
#include <minix/keymap.h>
#include <sys/ioc_disk.h>
#include <ibm/pci.h>

#define ATAPI_DEBUG          0    /* To debug ATAPI code. */

/* I/O Ports used by winchester disk controllers. */

/* Read and write registers */
#define REG_CMD_BASE0      0x1F0  /* command base register of controller 0 */
#define REG_CMD_BASE1      0x170  /* command base register of controller 1 */
#define REG_CTL_BASE0      0x3F6  /* control base register of controller 0 */
#define REG_CTL_BASE1      0x376  /* control base register of controller 1 */

#define PCI_CTL_OFF         2      /* Offset of control registers from BAR2 */
#define PCI_DMA_2ND_OFF     8      /* Offset of DMA registers from BAR4 for
 * secondary channel
 */

#define REG_DATA            0      /* data register (offset from the base reg.) */
#define REG_PRECOMP         1      /* start of write precompensation */
#define REG_COUNT           2      /* sectors to transfer */
#define REG_SECTOR          3      /* sector number */
#define REG_CYL_LO          4      /* low byte of cylinder number */
#define REG_CYL_HI          5      /* high byte of cylinder number */
#define REG_LDH              6      /* lba, drive and head */
#define LDH_DEFAULT         0xA0    /* ECC enable, 512 bytes per sector */
#define LDH_LBA              0x40    /* Use LBA addressing */
#define LDH_DEV              0x10    /* Drive 1 iff set */
#define ldh_init(drive)      (LDH_DEFAULT | ((drive) << 4))

/* Read only registers */
#define REG_STATUS           7      /* status */
#define STATUS_BSY          0x80    /* controller busy */
#define STATUS_RDY          0x40    /* drive ready */
#define STATUS_WF           0x20    /* write fault */
#define STATUS_SC           0x10    /* seek complete (obsolete) */
#define STATUS_DRQ          0x08    /* data transfer request */
#define STATUS_CRD          0x04    /* corrected data */
#define STATUS_IDX          0x02    /* index pulse */
#define STATUS_ERR          0x01    /* error */
#define STATUS_ADMBSY       0x100    /* administratively busy (software) */
#define REG_ERROR           1      /* error code */
#define ERROR_BB            0x80    /* bad block */
#define ERROR_ECC           0x40    /* bad ecc bytes */
#define ERROR_ID            0x10    /* id not found */
#define ERROR_AC            0x04    /* aborted command */
#define ERROR_TK            0x02    /* track zero error */
#define ERROR_DM            0x01    /* no data address mark */

/* Write only registers */
#define REG_COMMAND          7      /* command */
#define CMD_IDLE            0x00    /* for w_command: drive idle */
#define CMD_RECALIBRATE     0x10    /* recalibrate drive */
#define CMD_READ            0x20    /* read data */

```

```

#define CMD_READ_EXT 0x24 /* read data (LBA48 addressed) */
#define CMD_READ_DMA_EXT 0x25 /* read data using DMA (w/ LBA48) */
#define CMD_WRITE 0x30 /* write data */
#define CMD_WRITE_EXT 0x34 /* write data (LBA48 addressed) */
#define CMD_WRITE_DMA_EXT 0x35 /* write data using DMA (w/ LBA48) */
#define CMD_READVERIFY 0x40 /* read verify */
#define CMD_FORMAT 0x50 /* format track */
#define CMD_SEEK 0x70 /* seek cylinder */
#define CMD_DIAG 0x90 /* execute device diagnostics */
#define CMD_SPECIFY 0x91 /* specify parameters */
#define CMD_READ_DMA 0xC8 /* read data using DMA */
#define CMD_WRITE_DMA 0xCA /* write data using DMA */
#define ATA_IDENTIFY 0xEC /* identify drive */
/* #define REG_CTL 0x206 */ /* control register */
#define REG_CTL 0 /* control register */
#define CTL_NORETRY 0x80 /* disable access retry */
#define CTL_NOECC 0x40 /* disable ecc retry */
#define CTL_EIGHTHEADS 0x08 /* more than eight heads */
#define CTL_RESET 0x04 /* reset controller */
#define CTL_INTDISABLE 0x02 /* disable interrupts */
#define REG_CTL_ALTSTAT 0 /* alternate status register */

/* Identify words */
#define ID_GENERAL 0x00 /* General configuration information */
#define ID_GEN_NOT_ATA 0x8000 /* Not an ATA device */
#define ID_CAPABILITIES 0x31 /* Capabilities (49) */
#define ID_CAP_LBA 0x0200 /* LBA supported */
#define ID_CAP_DMA 0x0100 /* DMA supported */
#define ID_FIELD_VALIDITY 0x35 /* Field Validity (53) */
#define ID_FV_88 0x04 /* Word 88 is valid (UDMA) */
#define ID_MULTIWORD_DMA 0x3f /* Multiword DMA (63) */
#define ID_MWDMA_2_SEL 0x0400 /* Mode 2 is selected */
#define ID_MWDMA_1_SEL 0x0200 /* Mode 1 is selected */
#define ID_MWDMA_0_SEL 0x0100 /* Mode 0 is selected */
#define ID_MWDMA_2_SUP 0x0004 /* Mode 2 is supported */
#define ID_MWDMA_1_SUP 0x0002 /* Mode 1 is supported */
#define ID_MWDMA_0_SUP 0x0001 /* Mode 0 is supported */
#define ID_CSS 0x53 /* Command Sets Supported (83) */
#define ID_CSS_LBA48 0x0400
#define ID_ULTRA_DMA 0x58 /* Ultra DMA (88) */
#define ID_UDMA_5_SEL 0x2000 /* Mode 5 is selected */
#define ID_UDMA_4_SEL 0x1000 /* Mode 4 is selected */
#define ID_UDMA_3_SEL 0x0800 /* Mode 3 is selected */
#define ID_UDMA_2_SEL 0x0400 /* Mode 2 is selected */
#define ID_UDMA_1_SEL 0x0200 /* Mode 1 is selected */
#define ID_UDMA_0_SEL 0x0100 /* Mode 0 is selected */
#define ID_UDMA_5_SUP 0x0020 /* Mode 5 is supported */
#define ID_UDMA_4_SUP 0x0010 /* Mode 4 is supported */
#define ID_UDMA_3_SUP 0x0008 /* Mode 3 is supported */
#define ID_UDMA_2_SUP 0x0004 /* Mode 2 is supported */
#define ID_UDMA_1_SUP 0x0002 /* Mode 1 is supported */
#define ID_UDMA_0_SUP 0x0001 /* Mode 0 is supported */

/* DMA registers */
#define DMA_COMMAND 0 /* Command register */
#define DMA_CMD_WRITE 0x08 /* PCI bus master writes */
#define DMA_CMD_START 0x01 /* Start Bus Master */
#define DMA_STATUS 2 /* Status register */
#define DMA_ST_D1_DMACAP 0x40 /* Drive 1 is DMA capable */
#define DMA_ST_D0_DMACAP 0x20 /* Drive 0 is DMA capable */
#define DMA_ST_INT 0x04 /* Interrupt */
#define DMA_ST_ERROR 0x02 /* Error */
#define DMA_ST_BM_ACTIVE 0x01 /* Bus Master IDE Active */
#define DMA_PRDTP 4 /* PRD Table Pointer */

/* Check for the presence of LBA48 only on drives that are 'big'. */
#define LBA48_CHECK_SIZE 0x0f000000
#define LBA_MAX_SIZE 0x0fffffff /* Highest sector size for
* regular LBA.
*/

#if ENABLE_ATAPI
#define ERROR_SENSE 0xF0 /* sense key mask */
#define SENSE_NONE 0x00 /* no sense key */

```

```

#define SENSE_RECERR 0x10 /* recovered error */
#define SENSE_NOTRDY 0x20 /* not ready */
#define SENSE_MEDERR 0x30 /* medium error */
#define SENSE_HRDERR 0x40 /* hardware error */
#define SENSE_ILRQST 0x50 /* illegal request */
#define SENSE_UATTN 0x60 /* unit attention */
#define SENSE_DPROT 0x70 /* data protect */
#define SENSE_ABRT 0xb0 /* aborted command */
#define SENSE_MISCOM 0xe0 /* miscompare */
#define ERROR_MCR 0x08 /* media change requested */
#define ERROR_ABRT 0x04 /* aborted command */
#define ERROR_EOM 0x02 /* end of media detected */
#define ERROR_ILI 0x01 /* illegal length indication */
#define REG_FEAT 1 /* features */
#define FEAT_OVERLAP 0x02 /* overlap */
#define FEAT_DMA 0x01 /* dma */
#define REG_IRR 2 /* interrupt reason register */
#define IRR_REL 0x04 /* release */
#define IRR_IO 0x02 /* direction for xfer */
#define IRR_COD 0x01 /* command or data */
#define REG_SAMTAG 3
#define REG_CNT_LO 4 /* low byte of cylinder number */
#define REG_CNT_HI 5 /* high byte of cylinder number */
#define REG_DRIVE 6 /* drive select */
#endif

#define REG_STATUS 7 /* status */
#define STATUS_BSY 0x80 /* controller busy */
#define STATUS_DRDY 0x40 /* drive ready */
#define STATUS_DMADF 0x20 /* dma ready/drive fault */
#define STATUS_SRVCDSC 0x10 /* service or dsc */
#define STATUS_DRQ 0x08 /* data transfer request */
#define STATUS_CORR 0x04 /* correctable error occurred */
#define STATUS_CHECK 0x01 /* check error */

#ifdef ENABLE_ATAPI
#define ATAPI_PACKETCMD 0xA0 /* packet command */
#define ATAPI_IDENTIFY 0xA1 /* identify drive */
#define SCSI_READ10 0x28 /* read from disk */
#define SCSI_SENSE 0x03 /* sense request */

#define CD_SECTOR_SIZE 2048 /* sector size of a CD-ROM */
#endif /* ATAPI */

/* Interrupt request lines. */
#define NO_IRQ 0 /* no IRQ set yet */

#define ATAPI_PACKETSIZE 12
#define SENSE_PACKETSIZE 18

/* Common command block */
struct command {
    u8_t precomp; /* REG_PRECOMP, etc. */
    u8_t count;
    u8_t sector;
    u8_t cyl_lo;
    u8_t cyl_hi;
    u8_t ldh;
    u8_t command;

    /* The following at for LBA48 */
    u8_t count_prev;
    u8_t sector_prev;
    u8_t cyl_lo_prev;
    u8_t cyl_hi_prev;
};

/* Error codes */
#define ERR (-1) /* general error */
#define ERR_BAD_SECTOR (-2) /* block marked bad detected */

/* Some controllers don't interrupt, the clock will wake us up. */
#define WAKEUP (32*HZ) /* drive may be out for 31 seconds max */

```

```

/* Miscellaneous. */
#define MAX_DRIVES      8
#define COMPAT_DRIVES   4
#if _WORD_SIZE > 2
#define MAX_SECS        256    /* controller can transfer this many sectors */
#else
#define MAX_SECS        127    /* but not to a 16 bit process */
#endif
#define MAX_ERRORS      4      /* how often to try rd/wt before quitting */
#define NR_MINORS        (MAX_DRIVES * DEV_PER_DRIVE)
#define SUB_PER_DRIVE    (NR_PARTITIONS * NR_PARTITIONS)
#define NR_SUBDEVS       (MAX_DRIVES * SUB_PER_DRIVE)
#define DELAY_USECS      1000   /* controller timeout in microseconds */
#define DELAY_TICKS      1      /* controller timeout in ticks */
#define DEF_TIMEOUT_TICKS 300    /* controller timeout in ticks */
#define RECOVERY_USECS   500000 /* controller recovery time in microseconds */
#define RECOVERY_TICKS   30     /* controller recovery time in ticks */
#define INITIALIZED      0x01   /* drive is initialized */
#define DEAF             0x02   /* controller must be reset */
#define SMART            0x04   /* drive supports ATA commands */
#if ENABLE_ATAPI
#define ATAPI            0x08   /* it is an ATAPI device */
#else
#define ATAPI            0      /* don't bother with ATAPI; optimise out */
#endif
#define IDENTIFIED       0x10   /* w_identify done successfully */
#define IGNORING         0x20   /* w_identify failed once */

/* Timeouts and max retries. */
int timeout_ticks = DEF_TIMEOUT_TICKS, max_errors = MAX_ERRORS;
int wakeup_ticks = WAKEUP;
long w_standard_timeouts = 0, w_pci_debug = 0, w_instance = 0,
    disable_dma = 0, atapi_debug = 0;

int w_testing = 0, w_silent = 0;

int w_next_drive = 0;

/* Variables. */

/* The struct wini is indexed by controller first, then drive (0-3).
 * Controller 0 is always the 'compatability' ide controller, at
 * the fixed locations, whether present or not.
 */
PRIVATE struct wini {
    unsigned state;          /* main drive struct, one entry per drive */
    unsigned short w_status; /* drive state: deaf, initialized, dead */
    unsigned base_cmd;       /* device status register */
    unsigned base_ctl;       /* command base register */
    unsigned base_dma;       /* control base register */
    unsigned irq;            /* dma base register */
    unsigned irq_mask;       /* interrupt request line */
    unsigned irq_need_ack;   /* 1 << irq */
    int irq_hook_id;         /* irq needs to be acknowledged */
    int lba48;               /* id of irq hook at the kernel */
    int dma;                 /* supports lba48 */
    unsigned lcyllinders;    /* supports dma */
    unsigned lheads;         /* logical number of cylinders (BIOS) */
    unsigned lsectors;       /* logical number of heads */
    unsigned pcyllinders;    /* logical number of sectors per track */
    unsigned pheads;        /* physical number of cylinders (translated) */
    unsigned psectors;       /* physical number of heads */
    unsigned ldhpref;        /* physical number of sectors per track */
    unsigned precomp;        /* top four bytes of the LDH (head) register */
    unsigned max_count;      /* write precompensation cylinder / 4 */
    unsigned open_ct;        /* max request for this drive */
    struct device part[DEV_PER_DRIVE]; /* in-use count */
    struct device subpart[SUB_PER_DRIVE]; /* disks and partitions */
    /* subpartitions */
} wini[MAX_DRIVES], *w_wn;

PRIVATE int w_device = -1;
PRIVATE int w_controller = -1;
PRIVATE int w_major = -1;
PRIVATE char w_id_string[40];

```

```

PRIVATE int win_tasknr;          /* my task number */
PRIVATE int w_command;          /* current command in execution */
PRIVATE u8_t w_byteval;         /* used for SYS_IRQCTL */
PRIVATE int w_drive;           /* selected drive */
PRIVATE int w_controller;      /* selected controller */
PRIVATE struct device *w_dv;    /* device's base and size */

/* Unfortunately, DMA_SECTORS and DMA_BUF_SIZE are already defined libdriver
 * for 'tmp_buf'.
 */
#define ATA_DMA_SECTORS 64
#define ATA_DMA_BUF_SIZE (ATA_DMA_SECTORS*SECTOR_SIZE)

PRIVATE char dma_buf[ATA_DMA_BUF_SIZE];
PRIVATE phys_bytes dma_buf_phys;

#define N_PRDTE 1024 /* Should be enough for large requests */

PRIVATE struct prdte
{
    u32_t prdte_base;
    u16_t prdte_count;
    u8_t prdte_reserved;
    u8_t prdte_flags;
} prdt[N_PRDTE];
PRIVATE phys_bytes prdt_phys;

#define PRDTE_FL_EOT 0x80 /* End of table */

/* Some IDE devices announce themselves as RAID controllers */
PRIVATE struct
{
    u16_t vendor;
    u16_t device;
} raid_table[]=
{
    { 0x1106, 0x3149 }, /* VIA VT6420 */
    { 0, 0 } /* end of list */
};

FORWARD _PROTOTYPE( void init_params, (void) );
FORWARD _PROTOTYPE( void init_drive, (struct wini *w, int base_cmd,
    int base_ctl, int base_dma, int irq, int ack, int hook,
    int drive) );
FORWARD _PROTOTYPE( void init_params_pci, (int) );
FORWARD _PROTOTYPE( int w_do_open, (struct driver *dp, message *m_ptr) );
FORWARD _PROTOTYPE( struct device *w_prepare, (int dev) );
FORWARD _PROTOTYPE( int w_identify, (void) );
FORWARD _PROTOTYPE( char *w_name, (void) );
FORWARD _PROTOTYPE( int w_specify, (void) );
FORWARD _PROTOTYPE( int w_io_test, (void) );
FORWARD _PROTOTYPE( int w_transfer, (int proc_nr, int opcode, off_t position,
    iovec_t *iov, unsigned nr_req) );
FORWARD _PROTOTYPE( int com_out, (struct command *cmd) );
FORWARD _PROTOTYPE( int com_out_ext, (struct command *cmd) );
FORWARD _PROTOTYPE( void setup_dma, (unsigned *sizep, int proc_nr,
    iovec_t *iov, int do_write, int *do_copyoutp) );
FORWARD _PROTOTYPE( void w_need_reset, (void) );
FORWARD _PROTOTYPE( void ack_irqs, (unsigned int) );
FORWARD _PROTOTYPE( int w_do_close, (struct driver *dp, message *m_ptr) );
FORWARD _PROTOTYPE( int w_other, (struct driver *dp, message *m_ptr) );
FORWARD _PROTOTYPE( int w_hw_int, (struct driver *dp, message *m_ptr) );
FORWARD _PROTOTYPE( int com_simple, (struct command *cmd) );
FORWARD _PROTOTYPE( void w_timeout, (void) );
FORWARD _PROTOTYPE( int w_reset, (void) );
FORWARD _PROTOTYPE( void w_intr_wait, (void) );
FORWARD _PROTOTYPE( int at_intr_wait, (void) );
FORWARD _PROTOTYPE( int w_waitfor, (int mask, int value) );
FORWARD _PROTOTYPE( int w_waitfor_dma, (int mask, int value) );
FORWARD _PROTOTYPE( void w_geometry, (struct partition *entry) );
#if ENABLE_ATAPI
FORWARD _PROTOTYPE( int atapi_sendpacket, (u8_t *packet, unsigned cnt) );
FORWARD _PROTOTYPE( int atapi_intr_wait, (void) );

```



```

FORWARD _PROTOTYPE( int atapi_open, (void) ) ;
FORWARD _PROTOTYPE( void atapi_close, (void) ) ;
FORWARD _PROTOTYPE( int atapi_transfer, (int proc_nr, int opcode,
                                off_t position, iovec_t *iov, unsigned nr_req) ) ;

#endif

/* Entry points to this driver. */
PRIVATE struct driver w_dtab = {
    w_name,                /* current device's name */
    w_do_open,             /* open or mount request, initialize device */
    w_do_close,           /* release device */
    do_diocntl,           /* get or set a partition's geometry */
    w_prepare,            /* prepare for I/O on a given minor device */
    w_transfer,           /* do the I/O */
    nop_cleanup,          /* nothing to clean up */
    w_geometry,           /* tell the geometry of the disk */
    nop_signal,           /* no cleanup needed on shutdown */
    nop_alarm,            /* ignore leftover alarms */
    nop_cancel,           /* ignore CANCELS */
    nop_select,           /* ignore selects */
    w_other,              /* catch-all for unrecognized commands and ioctls */
    w_hw_int              /* leftover hardware interrupts */
};

/*=====
*                               at_winchester_task                               *
*=====*/
PUBLIC int main()
{
    /* Install signal handlers. Ask PM to transform signal into message. */
    struct sigaction sa;

    sa.sa_handler = SIG_MESS;
    sigemptyset(&sa.sa_mask);
    sa.sa_flags = 0;
    if (sigaction(SIGTERM,&sa,NULL)<0) panic("AT", "sigaction failed", errno);

    /* Set special disk parameters then call the generic main loop. */
    init_params();
    signal(SIGTERM, SIG_IGN);
    driver_task(&w_dtab);
    return(OK);
}

/*=====
*                               init_params                               *
*=====*/
PRIVATE void init_params()
{
    /* This routine is called at startup to initialize the drive parameters. */

    u16_t parv[2];
    unsigned int vector, size;
    int drive, nr_drives;
    struct wini *wn;
    u8_t params[16];
    int s;

    /* Boot variables. */
    env_parse("ata_std_timeout", "d", 0, &w_standard_timeouts, 0, 1);
    env_parse("ata_pci_debug", "d", 0, &w_pci_debug, 0, 1);
    env_parse("ata_instance", "d", 0, &w_instance, 0, 8);
    env_parse("ata_no_dma", "d", 0, &disable_dma, 0, 1);
    env_parse("atapi_debug", "d", 0, &atapi_debug, 0, 1);

    if (disable_dma)
        printf("DMA for ATA devices is disabled.\n");

    s = sys_umap(SELF, D, (vir_bytes)dma_buf, sizeof(dma_buf), &dma_buf_phys);
    if (s != 0)
        panic("at_wini", "can't map dma buffer", s);

    s = sys_umap(SELF, D, (vir_bytes)prdt, sizeof(prdt), &prdt_phys);
    if (s != 0)

```

```

panic("at_wini", "can't map prd table", s);

if (w_instance == 0) {
    /* Get the number of drives from the BIOS data area */
    if ((s=sys_vircopy(SELFS, BIOS_SEG, NR_HD_DRIVES_ADDR,
        SELFS, D, (vir_bytes) params, NR_HD_DRIVES_SIZE)) != OK)
        panic(w_name(), "Couldn't read BIOS", s);
    if ((nr_drives = params[0]) > 2) nr_drives = 2;

    for (drive = 0, wn = wini; drive < COMPAT_DRIVES; drive++, wn++) {
        if (drive < nr_drives) {
            /* Copy the BIOS parameter vector */
            vector = (drive == 0) ? BIOS_HD0_PARAMS_ADDR:BIOS_HD1_PARAMS_ADDR;
            size = (drive == 0) ? BIOS_HD0_PARAMS_SIZE:BIOS_HD1_PARAMS_SIZE;
            if ((s=sys_vircopy(SELFS, BIOS_SEG, vector,
                SELFS, D, (vir_bytes) parv, size)) != OK)
                panic(w_name(), "Couldn't read BIOS", s);

            /* Calculate the address of the parameters and copy them */
            if ((s=sys_vircopy(
                SELFS, BIOS_SEG, hclick_to_physb(parv[1]) + parv[0],
                SELFS, D, (phys_bytes) params, 16L))!=OK)
                panic(w_name(), "Couldn't copy parameters", s);

            /* Copy the parameters to the structures of the drive */
            wn->lcyllinders = bp_cylinders(params);
            wn->lheads = bp_heads(params);
            wn->lsectors = bp_sectors(params);
            wn->precomp = bp_precomp(params)>> 2;
        }

        /* Fill in non-BIOS parameters. */
        init_drive(wn,
            drive < 2 ? REG_CMD_BASE0 : REG_CMD_BASE1,
            drive < 2 ? REG_CTL_BASE0 : REG_CTL_BASE1,
            0 /* no DMA */, NO_IRQ, 0, 0, drive);
        w_next_drive++;
    }
}

/* Look for controllers on the pci bus. Skip none the first instance,
 * skip one and then 2 for every instance, for every next instance.
 */
if (w_instance == 0)
    init_params_pci(0);
else
    init_params_pci(w_instance*2-1);
}

#define ATA_IF_NOTCOMPAT1 (1L << 0)
#define ATA_IF_NOTCOMPAT2 (1L << 2)

/*=====
 *
 * init_drive
 *=====*/
PRIVATE void init_drive(struct wini *w, int base_cmd, int base_ctl,
    int base_dma, int irq, int ack, int hook, int drive)
{
    w->state = 0;
    w->w_status = 0;
    w->base_cmd = base_cmd;
    w->base_ctl = base_ctl;
    w->base_dma = base_dma;
    w->irq = irq;
    w->irq_mask = 1 << irq;
    w->irq_need_ack = ack;
    w->irq_hook_id = hook;
    w->ldhpref = ldh_init(drive);
    w->max_count = MAX_SECS << SECTOR_SHIFT;
    w->lba48 = 0;
    w->dma = 0;
}

```

```

/*=====
*                               init_params_pci                               *
*=====*/
PRIVATE void init_params_pci(int skip)
{
    int i, r, devind, drive;
    int irq, irq_hook, raid;
    u8_t bcr, scr, interface;
    u16_t vid, did;
    u32_t base_dma, t3;

    pci_init();
    for(drive = w_next_drive; drive < MAX_DRIVES; drive++)
        wini[drive].state = IGNORING;
    for(r = pci_first_dev(&devind, &vid, &did); r != 0;
        r = pci_next_dev(&devind, &vid, &did)) {

        raid= 0;

        /* Except class 01h (mass storage), subclass be 01h (ATA).
         * Also check listed RAID controllers.
         */
        bcr= pci_attr_r8(devind, PCI_BCR);
        scr= pci_attr_r8(devind, PCI_SCR);
        interface= pci_attr_r8(devind, PCI_PIFR);
        t3= ((bcr << 16) | (scr << 8) | interface);
        if (bcr == PCI_BCR_MASS_STORAGE && scr == PCI_MS_IDE)
            ; /* Okay */
        else if (t3 == PCI_T3_RAID)
        {
            for (i= 0; raid_table[i].vendor != 0; i++)
            {
                if (raid_table[i].vendor == vid &&
                    raid_table[i].device == did)
                {
                    break;
                }
            }
            if (raid_table[i].vendor == 0)
            {
                printf(
                    "atapci skipping unsupported RAID controller 0x%04x / 0x%04x\n",
                    vid, did);
                continue;
            }
            printf("found supported RAID controller\n");
            raid= 1;
        }
        else
            continue; /* Unsupported device class */

        /* Found a controller.
         * Programming interface register tells us more.
         */
        irq = pci_attr_r8(devind, PCI_ILR);

        /* Any non-compat drives? */
        if (raid || (interface & (ATA_IF_NOTCOMPAT1 | ATA_IF_NOTCOMPAT2))) {
            int s;

            if (w_next_drive >= MAX_DRIVES)
            {
                /* We can't accept more drives, but have to search for
                 * controllers operating in compatibility mode.
                 */
                continue;
            }

            irq_hook = irq;
            if (skip > 0) {
                if (w_pci_debug)
                {
                    printf(
                        "atapci skipping controller (remain %d)\n",

```

```

        skip);
    }
    skip--;
    continue;
}
if ((s=sys_irqsetpolicy(irq, 0, &irq_hook)) != OK) {
    printf("atapci: couldn't set IRQ policy %d\n", irq);
    continue;
}
if ((s=sys_irgenable(&irq_hook)) != OK) {
    printf("atapci: couldn't enable IRQ line %d\n", irq);
    continue;
}
}

base_dma = pci_attr_r32(devind, PCI_BAR_5) & 0xffffffffc;

/* Primary channel not in compatability mode? */
if (raid || (interface & ATA_IF_NOTCOMPAT1)) {
    u32_t base_cmd, base_ctl;

    base_cmd = pci_attr_r32(devind, PCI_BAR) & 0xffffffffc;
    base_ctl = pci_attr_r32(devind, PCI_BAR_2) & 0xffffffffc;
    if (base_cmd != REG_CMD_BASE0 && base_cmd != REG_CMD_BASE1) {
        init_drive(&wini[w_next_drive],
            base_cmd, base_ctl+PCI_CTL_OFF,
            base_dma, irq, 1, irq_hook, 0);
        init_drive(&wini[w_next_drive+1],
            base_cmd, base_ctl+PCI_CTL_OFF,
            base_dma, irq, 1, irq_hook, 1);
        if (w_pci_debug)
            printf("atapci %d: 0x%x 0x%x irq %d\n", devind, base_cmd, base_c
tl, irq);
        w_next_drive += 2;
    } else printf("atapci: ignored drives on primary channel, base %x\n", base_cmd);
}
else
{
    /* Update base_dma for compatability device */
    for (i= 0; i<MAX_DRIVES; i++)
    {
        if (wini[i].base_cmd == REG_CMD_BASE0)
            wini[i].base_dma= base_dma;
    }
}

/* Secondary channel not in compatability mode? */
if (raid || (interface & ATA_IF_NOTCOMPAT2)) {
    u32_t base_cmd, base_ctl;

    base_cmd = pci_attr_r32(devind, PCI_BAR_3) & 0xffffffffc;
    base_ctl = pci_attr_r32(devind, PCI_BAR_4) & 0xffffffffc;
    if (base_dma != 0)
        base_dma += PCI_DMA_2ND_OFF;
    if (base_cmd != REG_CMD_BASE0 && base_cmd != REG_CMD_BASE1) {
        init_drive(&wini[w_next_drive],
            base_cmd, base_ctl+PCI_CTL_OFF, base_dma,
            irq, 1, irq_hook, 2);
        init_drive(&wini[w_next_drive+1],
            base_cmd, base_ctl+PCI_CTL_OFF, base_dma,
            irq, 1, irq_hook, 3);
        if (w_pci_debug)
            printf("atapci %d: 0x%x 0x%x irq %d\n", devind, base_cmd, base_c
tl, irq);
        w_next_drive += 2;
    } else printf("atapci: ignored drives on secondary channel, base %x\n", base_cmd);
}
else
{
    /* Update base_dma for compatability device */
    for (i= 0; i<MAX_DRIVES; i++)
    {
        if (wini[i].base_cmd == REG_CMD_BASE1 && base_dma != 0)
            wini[i].base_dma= base_dma+PCI_DMA_2ND_OFF;
    }
}

```

```

    }
}
}

/*=====
 *                               w_do_open                               *
 *=====*/
PRIVATE int w_do_open(dp, m_ptr)
struct driver *dp;
message *m_ptr;
{
/* Device open: Initialize the controller and read the partition table. */

    struct wini *wn;

    if (w_prepare(m_ptr->DEVICE) == NIL_DEV) return(ENXIO);

    wn = w_wn;

    /* If we've probed it before and it failed, don't probe it again. */
    if (wn->state & IGNORING) return ENXIO;

    /* If we haven't identified it yet, or it's gone deaf,
     * (re-)identify it.
     */
    if (!(wn->state & IDENTIFIED) || (wn->state & DEAF)) {
        /* Try to identify the device. */
        if (w_identify() != OK) {
#if VERBOSE
            printf("%s: probe failed\n", w_name());
#endif
            if (wn->state & DEAF) w_reset();
            wn->state = IGNORING;
            return(ENXIO);
        }
        /* Do a test transaction unless it's a CD drive (then
         * we can believe the controller, and a test may fail
         * due to no CD being in the drive). If it fails, ignore
         * the device forever.
         */
        if (!(wn->state & ATAPI) && w_io_test() != OK) {
            wn->state |= IGNORING;
            return(ENXIO);
        }
    }
#if VERBOSE
    printf("%s: AT driver detected ", w_name());
    if (wn->state & (SMART|ATAPI)) {
        printf("%.40s\n", w_id_string);
    } else {
        printf("%ux%ux%u\n", wn->pcylinders, wn->pheads, wn->psectors);
    }
#endif
}

#if ENABLE_ATAPI
    if ((wn->state & ATAPI) && (m_ptr->COUNT & W_BIT))
        return(EACCES);
#endif

    /* Partition the drive if it's being opened for the first time,
     * or being opened after being closed.
     */
    if (wn->open_ct == 0) {
#if ENABLE_ATAPI
        if (wn->state & ATAPI) {
            int r;
            if ((r = atapi_open()) != OK) return(r);
        }
#endif
        /* Partition the disk. */
        partition(&w_dtab, w_drive * DEV_PER_DRIVE, P_PRIMARY, wn->state & ATAPI);
    }
}

```

```

}
wn->open_ct++;
return(OK);
}

/*=====
 *                               w_prepare                               *
 *=====*/
PRIVATE struct device *w_prepare(int device)
{
/* Prepare for I/O on a device. */
struct wini *prev_wn;
prev_wn = w_wn;
w_device = device;

if (device < NR_MINORS) {
    w_drive = device / DEV_PER_DRIVE;
    w_wn = &wini[w_drive];
    w_dv = &w_wn->part[device % DEV_PER_DRIVE];
} else
if ((unsigned) (device - MINOR_d0p0s0) < NR_SUBDEVS) { /*d[0-7]p[0-3]s[0-3]*/
    w_drive = device / SUB_PER_DRIVE;
    w_wn = &wini[w_drive];
    w_dv = &w_wn->subpart[device % SUB_PER_DRIVE];
} else {
    w_device = -1;
    return(NIL_DEV);
}
return(w_dv);
}

/*=====
 *                               w_identify                               *
 *=====*/
PRIVATE int w_identify()
{
/* Find out if a device exists, if it is an old AT disk, or a newer ATA
 * drive, a removable media device, etc.
 */

    struct wini *wn = w_wn;
    struct command cmd;
    int i, s;
    int id_dma, ultra_dma;
    u32_t dma_base;
    ul6_t w;
    unsigned long dma_status;
    unsigned long size;
#define id_byte(n)      (&tmp_buf[2 * (n)])
#define id_word(n)      (((ul6_t) id_byte(n)[0] << 0) \
                          | ((ul6_t) id_byte(n)[1] << 8))
#define id_longword(n)  (((u32_t) id_byte(n)[0] << 0) \
                          | ((u32_t) id_byte(n)[1] << 8) \
                          | ((u32_t) id_byte(n)[2] << 16) \
                          | ((u32_t) id_byte(n)[3] << 24))

    /* Try to identify the device. */
    cmd.ldh = wn->ldhpref;
    cmd.command = ATA_IDENTIFY;
    if (com_simple(&cmd) == OK && w_waitfor(STATUS_DRQ, STATUS_DRQ) &&
        !(wn->w_status & (STATUS_ERR|STATUS_WF))) {

        /* Device information. */
        if ((s=sys_insw(wn->base_cmd + REG_DATA, SELF, tmp_buf, SECTOR_SIZE)) != OK)
            panic(w_name(), "Call to sys_insw() failed", s);

        if (id_word(0) & ID_GEN_NOT_ATA)
        {
            printf("%s: not an ATA device?\n", w_name());
            return ERR;
        }

        /* This is an ATA device. */
        wn->state |= SMART;

```

```

/* Why are the strings byte swapped??? */
for (i = 0; i < 40; i++) w_id_string[i] = id_byte(27)[i^1];

/* Preferred CHS translation mode. */
wn->pcylinders = id_word(1);
wn->pheads = id_word(3);
wn->psectors = id_word(6);
size = (u32_t) wn->pcylinders * wn->pheads * wn->psectors;

w = id_word(ID_CAPABILITIES);
if ((w & ID_CAP_LBA) && size > 512L*1024*2) {
    /* Drive is LBA capable and is big enough to trust it to
     * not make a mess of it.
     */
    wn->ldhpref |= LDH_LBA;
    size = id_longword(60);

    w = id_word(ID_CSS);
    if (size < LBA48_CHECK_SIZE)
    {
        /* No need to check for LBA48 */
    }
    else if (w & ID_CSS_LBA48) {
        /* Drive is LBA48 capable (and LBA48 is turned on). */
        if (id_longword(102)) {
            /* If no. of sectors doesn't fit in 32 bits,
             * truncate to this. So it's LBA32 for now.
             * This can still address devices up to 2TB
             * though.
             */
            size = ULONG_MAX;
        } else {
            /* Actual number of sectors fits in 32 bits. */
            size = id_longword(100);
        }
        wn->lba48 = 1;
    }
}

/* Check for DMA. Assume that only LBA capable devices can do
 * DMA.
 */
w = id_word(ID_CAPABILITIES);
id_dma = !(w & ID_CAP_DMA);
w = id_byte(ID_FIELD_VALIDITY)[0];
ultra_dma = !(w & ID_FV_88);
dma_base = wn->base_dma;
if (dma_base)
{
    if (sys_inb(dma_base + DMA_STATUS, &dma_status) != OK)
    {
        panic(w_name(),
              "unable to read DMA status register",
              NO_NUM);
    }
}
if (disable_dma)
; /* DMA is disabled */
else if (id_dma && dma_base)
{
    w = id_word(ID_MULTIWORD_DMA);
    if (w & (ID_MWDMA_2_SUP | ID_MWDMA_1_SUP | ID_MWDMA_0_SUP))
    {
        printf(
            "%s: multiword DMA modes supported:%s%s%s\n",
            w_name(),
            (w & ID_MWDMA_0_SUP) ? " 0" : "",
            (w & ID_MWDMA_1_SUP) ? " 1" : "",
            (w & ID_MWDMA_2_SUP) ? " 2" : "");
    }
    if (w & (ID_MWDMA_0_SEL | ID_MWDMA_1_SEL | ID_MWDMA_2_SEL))
    {
        printf(
            "%s: multiword DMA mode selected:%s%s%s\n",

```

```

        w_name(),
        (w & ID_MWDMA_0_SEL) ? " 0" : "",
        (w & ID_MWDMA_1_SEL) ? " 1" : "",
        (w & ID_MWDMA_2_SEL) ? " 2" : "");
    }
    if (ultra_dma)
    {
        w = id_word(ID_ULTRA_DMA);
        if (w & (ID_UDMA_0_SUP | ID_UDMA_1_SUP |
                ID_UDMA_2_SUP | ID_UDMA_3_SUP |
                ID_UDMA_4_SUP | ID_UDMA_5_SUP))
        {
            printf(
                "%s: Ultra DMA modes supported:%s%s%s%s%s%s\n",
                w_name(),
                (w & ID_UDMA_0_SUP) ? " 0" : "",
                (w & ID_UDMA_1_SUP) ? " 1" : "",
                (w & ID_UDMA_2_SUP) ? " 2" : "",
                (w & ID_UDMA_3_SUP) ? " 3" : "",
                (w & ID_UDMA_4_SUP) ? " 4" : "",
                (w & ID_UDMA_5_SUP) ? " 5" : "");
        }
        if (w & (ID_UDMA_0_SEL | ID_UDMA_1_SEL |
                ID_UDMA_2_SEL | ID_UDMA_3_SEL |
                ID_UDMA_4_SEL | ID_UDMA_5_SEL))
        {
            printf(
                "%s: Ultra DMA mode selected:%s%s%s%s%s%s\n",
                w_name(),
                (w & ID_UDMA_0_SEL) ? " 0" : "",
                (w & ID_UDMA_1_SEL) ? " 1" : "",
                (w & ID_UDMA_2_SEL) ? " 2" : "",
                (w & ID_UDMA_3_SEL) ? " 3" : "",
                (w & ID_UDMA_4_SEL) ? " 4" : "",
                (w & ID_UDMA_5_SEL) ? " 5" : "");
        }
    }
    wn->dma = 1;
}
else if (id_dma || dma_base)
{
    printf("id_dma %d, dma_base 0x%x\n", id_dma, dma_base);
}
else
    printf("no DMA support\n");

#if 0
    if (wn->dma && wn == &wini[0])
    {
        printf("disabling DMA for drive 0\n");
        wn->dma = 0;
    }
#endif

}

if (wn->lcyllinders == 0) {
    /* No BIOS parameters? Then make some up. */
    wn->lcyllinders = wn->pcylinders;
    wn->lheads = wn->pheads;
    wn->lsectors = wn->psectors;
    while (wn->lcyllinders > 1024) {
        wn->lheads *= 2;
        wn->lcyllinders /= 2;
    }
}

#if ENABLE_ATAPI
} else
if (cmd.command == ATAPI_IDENTIFY,
    com_simple(&cmd) == OK && w_waitfor(STATUS_DRQ, STATUS_DRQ) &&
    !(wn->w_status & (STATUS_ERR | STATUS_WF))) {
    /* An ATAPI device. */
    wn->state |= ATAPI;

    /* Device information. */

```



```

    if ((s=sys_insw(wn->base_cmd + REG_DATA, SELF, tmp_buf, 512)) != OK)
        panic(w_name(), "Call to sys_insw() failed", s);

    /* Why are the strings byte swapped??? */
    for (i = 0; i < 40; i++) w_id_string[i] = id_byte(27)[i^1];

    size = 0;          /* Size set later. */
#endif
} else {
    /* Not an ATA device; no translations, no special features. Don't
     * touch it unless the BIOS knows about it.
     */
    if (wn->lcyllinders == 0) { return(ERR); }          /* no BIOS parameters */
    wn->pcylinders = wn->lcyllinders;
    wn->pheads = wn->lheads;
    wn->psectors = wn->lsectors;
    size = (u32_t) wn->pcylinders * wn->pheads * wn->psectors;
}

/* Size of the whole drive */
wn->part[0].dv_size = mul64u(size, SECTOR_SIZE);

/* Reset/calibrate (where necessary) */
if (w_specify() != OK && w_specify() != OK) {
    return(ERR);
}

if (wn->irq == NO_IRQ) {
    /* Everything looks OK; register IRQ so we can stop polling. */
    wn->irq = w_drive < 2 ? AT_WINI_0_IRQ : AT_WINI_1_IRQ;
    wn->irq_hook_id = wn->irq;          /* id to be returned if interrupt occurs */
    if ((s=sys_irqsetpolicy(wn->irq, IRQ_REENABLE, &wn->irq_hook_id)) != OK)
        panic(w_name(), "couldn't set IRQ policy", s);
    if ((s=sys_irgenable(&wn->irq_hook_id)) != OK)
        panic(w_name(), "couldn't enable IRQ line", s);
}
wn->state |= IDENTIFIED;
return(OK);
}

/*=====
 *                               w_name                               *
 *=====*/
PRIVATE char *w_name()
{
    /* Return a name for the current device. */
    static char name[] = "AT-D0";

    name[4] = '0' + w_drive;
    return name;
}

/*=====
 *                               w_io_test                           *
 *=====*/
PRIVATE int w_io_test(void)
{
    int r, save_dev;
    int save_timeout, save_errors, save_wakeup;
    iovec_t iov;
#ifdef CD_SECTOR_SIZE
    static char buf[CD_SECTOR_SIZE];
#else
    static char buf[SECTOR_SIZE];
#endif

    iov.iov_addr = (vir_bytes) buf;
    iov.iov_size = sizeof(buf);
    save_dev = w_device;

    /* Reduce timeout values for this test transaction. */
    save_timeout = timeout_ticks;
    save_errors = max_errors;
    save_wakeup = wakeup_ticks;

```

```

    if (!w_standard_timeouts) {
        timeout_ticks = HZ * 4;
        wakeup_ticks = HZ * 6;
        max_errors = 3;
    }

    w_testing = 1;

    /* Try I/O on the actual drive (not any (sub)partition). */
    if (w_prepare(w_drive * DEV_PER_DRIVE) == NIL_DEV)
        panic(w_name(), "Couldn't switch devices", NO_NUM);

    r = w_transfer(SELF, DEV_GATHER, 0, &iiov, 1);

    /* Switch back. */
    if (w_prepare(save_dev) == NIL_DEV)
        panic(w_name(), "Couldn't switch back devices", NO_NUM);

    /* Restore parameters. */
    timeout_ticks = save_timeout;
    max_errors = save_errors;
    wakeup_ticks = save_wakeup;
    w_testing = 0;

    /* Test if everything worked. */
    if (r != OK || iiov.iov_size != 0) {
        return ERR;
    }

    /* Everything worked. */

    return OK;
}

/*=====
 *                               w_specify                               *
 *=====*/
PRIVATE int w_specify()
{
    /* Routine to initialize the drive after boot or when a reset is needed. */

    struct wini *wn = w_wn;
    struct command cmd;

    if ((wn->state & DEAF) && w_reset() != OK) {
        return(ERR);
    }

    if (!(wn->state & ATAPI)) {
        /* Specify parameters: precompensation, number of heads and sectors. */
        cmd.precomp = wn->precomp;
        cmd.count = wn->psectors;
        cmd.ldh = w_wn->ldhpref | (wn->pheads - 1);
        cmd.command = CMD_SPECIFY; /* Specify some parameters */

        /* Output command block and see if controller accepts the parameters. */
        if (com_simple(&cmd) != OK) return(ERR);

        if (!(wn->state & SMART)) {
            /* Calibrate an old disk. */
            cmd.sector = 0;
            cmd.cyl_lo = 0;
            cmd.cyl_hi = 0;
            cmd.ldh = w_wn->ldhpref;
            cmd.command = CMD_RECALIBRATE;

            if (com_simple(&cmd) != OK) return(ERR);
        }
    }
    wn->state |= INITIALIZED;
    return(OK);
}

```

```

/*=====
*
*                               do_transfer
*=====*/
PRIVATE int do_transfer(struct wini *wn, unsigned int precomp,
    unsigned int count, unsigned int sector,
    unsigned int opcode, int do_dma)
{
    struct command cmd;
    unsigned int sector_high;
    unsigned secspcyl = wn->pheads * wn->psectors;
    int do_lba48;

    sector_high= 0; /* For future extensions */

    do_lba48= 0;
    if (sector >= LBA48_CHECK_SIZE || sector_high != 0)
    {
        if (wn->lba48)
            do_lba48= 1;
        else if (sector > LBA_MAX_SIZE || sector_high != 0)
        {
            /* Strange sector count for LBA device */
            return EIO;
        }
    }

    cmd.precomp = precomp;
    cmd.count   = count;
    if (do_dma)
    {
        cmd.command = opcode == DEV_SCATTER ? CMD_WRITE_DMA :
            CMD_READ_DMA;
    }
    else
        cmd.command = opcode == DEV_SCATTER ? CMD_WRITE : CMD_READ;

    if (do_lba48) {
        if (do_dma)
        {
            cmd.command = ((opcode == DEV_SCATTER) ?
                CMD_WRITE_DMA_EXT : CMD_READ_DMA_EXT);
        }
        else
        {
            cmd.command = ((opcode == DEV_SCATTER) ?
                CMD_WRITE_EXT : CMD_READ_EXT);
        }
        cmd.count_prev= (count >> 8);
        cmd.sector   = (sector >> 0) & 0xFF;
        cmd.cyl_lo   = (sector >> 8) & 0xFF;
        cmd.cyl_hi   = (sector >> 16) & 0xFF;
        cmd.sector_prev= (sector >> 24) & 0xFF;
        cmd.cyl_lo_prev= (sector_high) & 0xFF;
        cmd.cyl_hi_prev= (sector_high >> 8) & 0xFF;
        cmd.ldh      = wn->ldhpref;

        return com_out_ext(&cmd);
    } else if (wn->ldhpref & LDH_LBA) {
        cmd.sector   = (sector >> 0) & 0xFF;
        cmd.cyl_lo   = (sector >> 8) & 0xFF;
        cmd.cyl_hi   = (sector >> 16) & 0xFF;
        cmd.ldh      = wn->ldhpref | ((sector >> 24) & 0xF);
    } else {
        int cylinder, head, sec;
        cylinder = sector / secspcyl;
        head = (sector % secspcyl) / wn->psectors;
        sec = sector % wn->psectors;
        cmd.sector   = sec + 1;
        cmd.cyl_lo   = cylinder & BYTE;
        cmd.cyl_hi   = (cylinder >> 8) & BYTE;
        cmd.ldh      = wn->ldhpref | head;
    }

    return com_out(&cmd);
}

```

```

}

/*=====
 *
 * w_transfer
 *=====*/
PRIVATE int w_transfer(proc_nr, opcode, position, iov, nr_req)
int proc_nr;          /* process doing the request */
int opcode;           /* DEV_GATHER or DEV_SCATTER */
off_t position;       /* offset on device to read or write */
iovec_t *iov;         /* pointer to read or write request vector */
unsigned nr_req;      /* length of request vector */
{
    struct wini *wn = w_wn;
    iovec_t *iop, *iov_end = iov + nr_req;
    int n, r, s, errors, do_dma, do_write, do_copyout;
    unsigned long v, block, w_status;
    unsigned long dv_size = cv64ul(w_dv->dv_size);
    unsigned cylinder, head, sector, nbytes;
    unsigned dma_buf_offset;

#ifdef ENABLE_ATAPI
    if (w_wn->state & ATAPI) {
        return atapi_transfer(proc_nr, opcode, position, iov, nr_req);
    }
#endif

    /* Check disk address. */
    if ((position & SECTOR_MASK) != 0) return(EINVAL);

    errors = 0;

    while (nr_req > 0) {
        /* How many bytes to transfer? */
        nbytes = 0;
        for (iop = iov; iop < iov_end; iop++) nbytes += iop->iov_size;
        if ((nbytes & SECTOR_MASK) != 0) return(EINVAL);

        /* Which block on disk and how close to EOF? */
        if (position >= dv_size) return(OK);          /* At EOF */
        if (position + nbytes > dv_size) nbytes = dv_size - position;
        block = div64u(add64ul(w_dv->dv_base, position), SECTOR_SIZE);

        do_dma= wn->dma;
        do_write= (opcode == DEV_SCATTER);

        if (nbytes >= wn->max_count) {
            /* The drive can't do more than max_count at once. */
            nbytes = wn->max_count;
        }

        /* First check to see if a reinitialization is needed. */
        if (!(wn->state & INITIALIZED) && w_specify() != OK) return(EIO);

        if (do_dma)
        {
            setup_dma(&nbytes, proc_nr, iov, do_write, &do_copyout);
#ifdef 0
            printf("nbytes = %d\n", nbytes);
#endif
        }

        /* Tell the controller to transfer nbytes bytes. */
        r = do_transfer(wn, wn->precomp, (nbytes >> SECTOR_SHIFT),
            block, opcode, do_dma);

        if (opcode == DEV_SCATTER) {
            /* The specs call for a 400 ns wait after issuing the command.
             * Reading the alternate status register is the suggested
             * way to implement this wait.
             */
            if (sys_inb((wn->base_ctl+REG_CTL_ALTSTAT), &w_status) != OK)
                panic(w_name(), "couldn't get status", NO_NUM);
        }
    }
}

```

```

    }

    if (do_dma)
    {
        /* Wait for the interrupt, check DMA status and optionally
         * copy out.
         */

        if ((r = at_intr_wait()) != OK)
        {
            /* Don't retry if sector marked bad or too many
             * errors.
             */
            if (r == ERR_BAD_SECTOR || ++errors == max_errors) {
                w_command = CMD_IDLE;
                return(EIO);
            }
            continue;
        }

        /* Wait for DMA_ST_INT to get set */
        w_waitfor_dma(DMA_ST_INT, DMA_ST_INT);

        r= sys_inb(wn->base_dma + DMA_STATUS, &v);
        if (r != 0) panic("at_wini", "w_transfer: sys_inb failed", r);

#if 0
        printf("dma_status: 0x%x\n", v);
#endif

        if (!(v & DMA_ST_INT))
        {
            /* DMA did not complete successfully */
            if (v & DMA_ST_BM_ACTIVE)
                panic(w_name(), "DMA did not complete", NO_NUM);
            else if (v & DMA_ST_ERROR)
            {
                printf("at_wini: DMA error\n");
                r= EIO;
                break;
            }
            else
            {
                printf("DMA buffer too small\n");

                panic(w_name(), "DMA buffer too small", NO_NUM);
            }
        }
        else if (v & DMA_ST_BM_ACTIVE)
            panic(w_name(), "DMA buffer too large", NO_NUM);

        dma_buf_offset= 0;
        while (r == OK && nbytes > 0)
        {
            n= iov->iov_size;
            if (n > nbytes)
                n= nbytes;

            if (do_copyout)
            {
                s= sys_vircopy(SELF, D,
                               (vir_bytes)dma_buf+dma_buf_offset,
                               proc_nr, D, iov->iov_addr, n);
                if (s != OK)
                {
                    panic(w_name(),
                          "w_transfer: sys_vircopy failed",
                          s);
                }
            }

            /* Book the bytes successfully transferred. */
            nbytes -= n;
            position += n;
        }
    }
}

```

```

        iov->iov_addr += n;
        if ((iov->iov_size -= n) == 0)
            { iov++; nr_req--; }
        dma_buf_offset += n;
    }
}

while (r == OK && nbytes > 0) {
    /* For each sector, wait for an interrupt and fetch the data
     * (read), or supply data to the controller and wait for an
     * interrupt (write).
     */

    if (opcode == DEV_GATHER) {
        /* First an interrupt, then data. */
        if ((r = at_intr_wait()) != OK) {
            /* An error, send data to the bit bucket. */
            if (w_wn->w_status & STATUS_DRQ) {
                if ((s=sys_insw(wn->base_cmd+REG_DATA,
                                SELF, tmp_buf,
                                SECTOR_SIZE)) != OK)
                {
                    panic(w_name(),
                        "Call to sys_insw() failed",
                        s);
                }
            }
            break;
        }
    }

    /* Wait for busy to clear. */
    if (!w_waitfor(STATUS_BSY, 0)) { r = ERR; break; }

    /* Wait for data transfer requested. */
    if (!w_waitfor(STATUS_DRQ, STATUS_DRQ)) { r = ERR; break; }

    /* Copy bytes to or from the device's buffer. */
    if (opcode == DEV_GATHER) {
        if ((s=sys_insw(wn->base_cmd + REG_DATA, proc_nr,
                        (void *) iov->iov_addr, SECTOR_SIZE)) != OK)
        {
            panic(w_name(), "Call to sys_insw() failed", s);
        }
    }
    else {
        if ((s=sys_outsw(wn->base_cmd + REG_DATA, proc_nr,
                        (void *) iov->iov_addr, SECTOR_SIZE)) != OK)
        {
            panic(w_name(), "Call to sys_outsw() failed",
                s);
        }

        /* Data sent, wait for an interrupt. */
        if ((r = at_intr_wait()) != OK) break;
    }

    /* Book the bytes successfully transferred. */
    nbytes -= SECTOR_SIZE;
    position += SECTOR_SIZE;
    iov->iov_addr += SECTOR_SIZE;
    if ((iov->iov_size -= SECTOR_SIZE) == 0) { iov++; nr_req--; }
}

/* Any errors? */
if (r != OK) {
    /* Don't retry if sector marked bad or too many errors. */
    if (r == ERR_BAD_SECTOR || ++errors == max_errors) {
        w_command = CMD_IDLE;
        return(EIO);
    }
}
}

w_command = CMD_IDLE;

```

```

    return(OK);
}

/*=====
 *                               com_out                               *
 *=====*/
PRIVATE int com_out(cmd)
struct command *cmd;          /* Command block */
{
    /* Output the command block to the winchester controller and return status */

    struct wini *wn = w_wn;
    unsigned base_cmd = wn->base_cmd;
    unsigned base_ctl = wn->base_ctl;
    pvb_pair_t outbyte[7];    /* vector for sys_voutb() */
    int s;                    /* status for sys_(v)outb() */

    if (w_wn->state & IGNORING) return ERR;

    if (!w_waitfor(STATUS_BSY, 0)) {
        printf("%s: controller not ready\n", w_name());
        return(ERR);
    }

    /* Select drive. */
    if ((s=sys_outb(base_cmd + REG_LDH, cmd->ldh)) != OK)
        panic(w_name(), "Couldn't write register to select drive", s);

    if (!w_waitfor(STATUS_BSY, 0)) {
        printf("%s: com_out: drive not ready\n", w_name());
        return(ERR);
    }

    /* Schedule a wakeup call, some controllers are flaky. This is done with
     * a synchronous alarm. If a timeout occurs a SYN_ALARM message is sent
     * from HARDWARE, so that w_intr_wait() can call w_timeout() in case the
     * controller was not able to execute the command. Leftover timeouts are
     * simply ignored by the main loop.
     */
    sys_setalarm(wakeup_ticks, 0);

    wn->w_status = STATUS_ADMBSY;
    w_command = cmd->command;
    pv_set(outbyte[0], base_ctl + REG_CTL, wn->pheads >= 8 ? CTL_EIGHTHEADS : 0);
    pv_set(outbyte[1], base_cmd + REG_PRECOMP, cmd->precomp);
    pv_set(outbyte[2], base_cmd + REG_COUNT, cmd->count);
    pv_set(outbyte[3], base_cmd + REG_SECTOR, cmd->sector);
    pv_set(outbyte[4], base_cmd + REG_CYL_LO, cmd->cyl_lo);
    pv_set(outbyte[5], base_cmd + REG_CYL_HI, cmd->cyl_hi);
    pv_set(outbyte[6], base_cmd + REG_COMMAND, cmd->command);
    if ((s=sys_voutb(outbyte, 7)) != OK)
        panic(w_name(), "Couldn't write registers with sys_voutb()", s);
    return(OK);
}

/*=====
 *                               com_out_ext                               *
 *=====*/
PRIVATE int com_out_ext(cmd)
struct command *cmd;          /* Command block */
{
    /* Output the command block to the winchester controller and return status */

    struct wini *wn = w_wn;
    unsigned base_cmd = wn->base_cmd;
    unsigned base_ctl = wn->base_ctl;
    pvb_pair_t outbyte[11];   /* vector for sys_voutb() */
    int s;                    /* status for sys_(v)outb() */
    unsigned long w_status;

    if (w_wn->state & IGNORING) return ERR;

    if (!w_waitfor(STATUS_BSY, 0)) {
        printf("%s: controller not ready\n", w_name());

```

```

    return(ERR);
}

/* Select drive. */
if ((s=sys_outb(base_cmd + REG_LDH, cmd->ldh)) != OK)
    panic(w_name(), "Couldn't write register to select drive", s);

if (!w_waitfor(STATUS_BSY, 0)) {
    printf("%s: com_out: drive not ready\n", w_name());
    return(ERR);
}

/* Schedule a wakeup call, some controllers are flaky. This is done with
 * a synchronous alarm. If a timeout occurs a SYN_ALARM message is sent
 * from HARDWARE, so that w_intr_wait() can call w_timeout() in case the
 * controller was not able to execute the command. Leftover timeouts are
 * simply ignored by the main loop.
 */
sys_setalarm(wakeup_ticks, 0);

wn->w_status = STATUS_ADMSY;
w_command = cmd->command;
pv_set(outbyte[0], base_ctl + REG_CTL, 0);
pv_set(outbyte[1], base_cmd + REG_COUNT, cmd->count_prev);
pv_set(outbyte[2], base_cmd + REG_SECTOR, cmd->sector_prev);
pv_set(outbyte[3], base_cmd + REG_CYL_LO, cmd->cyl_lo_prev);
pv_set(outbyte[4], base_cmd + REG_CYL_HI, cmd->cyl_hi_prev);
pv_set(outbyte[5], base_cmd + REG_COUNT, cmd->count);
pv_set(outbyte[6], base_cmd + REG_SECTOR, cmd->sector);
pv_set(outbyte[7], base_cmd + REG_CYL_LO, cmd->cyl_lo);
pv_set(outbyte[8], base_cmd + REG_CYL_HI, cmd->cyl_hi);

pv_set(outbyte[10], base_cmd + REG_COMMAND, cmd->command);
if ((s=sys_voutb(outbyte, 11)) != OK)
    panic(w_name(), "Couldn't write registers with sys_voutb()", s);

return(OK);
}

/*=====
 *                               setup_dma                               *
 *=====*/
PRIVATE void setup_dma(sizep, proc_nr, iov, do_write, do_copyoutp)
unsigned *sizep;
int proc_nr;
iovec_t *iov;
int do_write;
int *do_copyoutp;
{
    phys_bytes phys, user_phys;
    unsigned n, offset, size;
    int i, j, r, bad;
    unsigned long v;
    struct wini *wn = w_wn;

    /* First try direct scatter/gather to the supplied buffers */
    size= *sizep;
    i= 0;    /* iov index */
    j= 0;    /* prdt index */
    bad= 0;
    offset= 0;    /* Offset in current iov */

#ifdef 0
    printf("setup_dma: proc_nr %d\n", proc_nr);
#endif

    while (size > 0)
    {
#ifdef 0
        printf(
            "setup_dma: iov[%d]: addr 0x%x, size %d offset %d, size %d\n",
            i, iov[i].iov_addr, iov[i].iov_size, offset, size);
#endif
    }
}

```



```

n= iov[i].iov_size-offset;
if (n > size)
    n= size;
if (n == 0 || (n & 1))
    panic("at_wini", "bad size in iov", iov[i].iov_size);
r= sys_umap(proc_nr, D, iov[i].iov_addr+offset, n, &user_phys);
if (r != 0)
    panic("at_wini", "can't map user buffer", r);
if (user_phys & 1)
{
    /* Buffer is not aligned */
    printf("setup_dma: user buffer is not aligned\n");
    bad= 1;
    break;
}

/* vector is not allowed to cross a 64K boundary */
if (user_phys/0x10000 != (user_phys+n-1)/0x10000)
    n= ((user_phys/0x10000)+1)*0x10000 - user_phys;

/* vector is not allowed to be bigger than 64K, but we get that
 * for free.
 */

if (j >= N_PRDTE)
{
    /* Too many entries */
    bad= 1;
    break;
}

prdt[j].prdte_base= user_phys;
prdt[j].prdte_count= n;
prdt[j].prdte_reserved= 0;
prdt[j].prdte_flags= 0;
j++;

offset += n;
if (offset >= iov[i].iov_size)
{
    i++;
    offset= 0;
}

size -= n;
}

if (!bad)
{
    if (j <= 0 || j > N_PRDTE)
        panic("at_wini", "bad prdt index", j);
    prdt[j-1].prdte_flags |= PRDTE_FL_EOT;
}

#if 0

    for (i= 0; i<j; i++)
    {
        printf("prdt[%d]: base 0x%x, size %d, flags 0x%x\n",
            i, prdt[i].prdte_base, prdt[i].prdte_count,
            prdt[i].prdte_flags);
    }

#endif
}

/* The caller needs to perform a copy-out from the dma buffer if
 * this is a read request and we can't DMA directly to the user's
 * buffers.
 */
do_copyoutp= (!do_write && bad);

if (bad)
{
    /* Adjust request size */
    size= *sizep;
    if (size > ATA_DMA_BUF_SIZE)

```

```

        *sizep= size= ATA_DMA_BUF_SIZE;

    if (do_write)
    {
        /* Copy-in */
        for (offset= 0; offset < size; offset += n)
        {
            n= size-offset;
            if (n > iov->iov_size)
                n= iov->iov_size;

            r= sys_vircopy(proc_nr, D, iov->iov_addr,
                          SELF, D, (vir_bytes)dma_buf+offset,
                          n);
            if (r != OK)
            {
                panic(w_name(),
                      "setup_dma: sys_vircopy failed",
                      r);
            }
            iov++;
        }
    }

    /* Fill-in the physical region descriptor table */
    phys= dma_buf_phys;
    if (phys & 1)
    {
        /* Two byte alignment is required */
        panic("at_wini", "bad buffer alignment in setup_dma",
              phys);
    }
    for (j= 0; j<N_PRDTE; i++)
    {
        if (size == 0)
        {
            panic("at_wini", "bad size in setup_dma",
                  size);
        }
        if (size & 1)
        {
            /* Two byte alignment is required for size */
            panic("at_wini",
                  "bad size alignment in setup_dma",
                  size);
        }
        n= size;

        /* Buffer is not allowed to cross a 64K boundary */
        if (phys / 0x10000 != (phys+n-1) / 0x10000)
        {
            n= ((phys/0x10000)+1)*0x10000 - phys;
        }
        prdt[j].prdte_base= phys;
        prdt[j].prdte_count= n;
        prdt[j].prdte_reserved= 0;
        prdt[j].prdte_flags= 0;

        size -= n;
        if (size == 0)
        {
            prdt[j].prdte_flags |= PRDTE_FL_EOT;
            break;
        }
    }
    if (size != 0)
        panic("at_wini", "size to large for prdt", NO_NUM);

    #if 0

    for (i= 0; i<=j; i++)
    {
        printf("prdt[%d]: base 0x%x, size %d, flags 0x%x\n",
              i, prdt[i].prdte_base, prdt[i].prdte_count,
              prdt[i].prdte_flags);
    }

```

```

    }
#endif
}

/* Stop bus master operation */
r= sys_outb(wn->base_dma + DMA_COMMAND, 0);
if (r != 0) panic("at_wini", "setup_dma: sys_outb failed", r);

/* Verify that the bus master is not active */
r= sys_inb(wn->base_dma + DMA_STATUS, &v);
if (r != 0) panic("at_wini", "setup_dma: sys_inb failed", r);
if (v & DMA_ST_BM_ACTIVE)
    panic("at_wini", "Bus master IDE active", NO_NUM);

if (prdt_phys & 3)
    panic("at_wini", "prdt not aligned", prdt_phys);
r= sys_outl(wn->base_dma + DMA_PRDTP, prdt_phys);
if (r != 0) panic("at_wini", "setup_dma: sys_outl failed", r);

/* Clear interrupt and error flags */
r= sys_outb(wn->base_dma + DMA_STATUS, DMA_ST_INT | DMA_ST_ERROR);
if (r != 0) panic("at_wini", "setup_dma: sys_outb failed", r);

/* Assume disk reads. Start DMA */
v= DMA_CMD_START;
if (!do_write)
{
    /* Disk reads generate PCI write cycles. */
    v |= DMA_CMD_WRITE;
}
r= sys_outb(wn->base_dma + DMA_COMMAND, v);
if (r != 0) panic("at_wini", "setup_dma: sys_outb failed", r);

#if 0
r= sys_inb(wn->base_dma + DMA_STATUS, &v);
if (r != 0) panic("at_wini", "setup_dma: sys_inb failed", r);
printf("dma status: 0x%x\n", v);
#endif
}

/*=====
*                                     w_need_reset                                     *
*=====*/
PRIVATE void w_need_reset()
{
    /* The controller needs to be reset. */
    struct wini *wn;
    int dr = 0;

    for (wn = wini; wn < &wini[MAX_DRIVES]; wn++, dr++) {
        if (wn->base_cmd == w_wn->base_cmd) {
            wn->state |= DEAF;
            wn->state &= ~INITIALIZED;
        }
    }
}

/*=====
*                                     w_do_close                                     *
*=====*/
PRIVATE int w_do_close(dp, m_ptr)
struct driver *dp;
message *m_ptr;
{
    /* Device close: Release a device. */
    if (w_prepare(m_ptr->DEVICE) == NIL_DEV)
        return(ENXIO);
    w_wn->open_ct--;
#if ENABLE_ATAPI
    if (w_wn->open_ct == 0 && (w_wn->state & ATAPI)) atapi_close();
#endif
    return(OK);
}

```

```

/*=====
*                                     com_simple                                     *
*=====*/
PRIVATE int com_simple(cmd)
struct command *cmd;          /* Command block */
{
/* A simple controller command, only one interrupt and no data-out phase. */
int r;

if (w_wn->state & IGNORING) return ERR;

if ((r = com_out(cmd)) == OK) r = at_intr_wait();
w_command = CMD_IDLE;
return(r);
}

/*=====
*                                     w_timeout                                     *
*=====*/
PRIVATE void w_timeout(void)
{
struct wini *wn = w_wn;

switch (w_command) {
case CMD_IDLE:
break;          /* fine */
case CMD_READ:
case CMD_READ_EXT:
case CMD_WRITE:
case CMD_WRITE_EXT:
/* Impossible, but not on PC's: The controller does not respond. */

/* Limiting multisector I/O seems to help. */
if (wn->max_count > 8 * SECTOR_SIZE) {
wn->max_count = 8 * SECTOR_SIZE;
} else {
wn->max_count = SECTOR_SIZE;
}
/*FALL THROUGH*/
default:
/* Some other command. */
if (w_testing) wn->state |= IGNORING; /* Kick out this drive. */
else if (!w_silent) printf("%s: timeout on command 0x%02x\n",
w_name(), w_command);
w_need_reset();
wn->w_status = 0;
}
}

/*=====
*                                     w_reset                                     *
*=====*/
PRIVATE int w_reset()
{
/* Issue a reset to the controller. This is done after any catastrophe,
* like the controller refusing to respond.
*/
int s;
struct wini *wn = w_wn;

/* Don't bother if this drive is forgotten. */
if (w_wn->state & IGNORING) return ERR;

/* Wait for any internal drive recovery. */
tickdelay(RECOVERY_TICKS);

/* Strobe reset bit */
if ((s=sys_outb(wn->base_ctl + REG_CTL, CTL_RESET)) != OK)
panic(w_name(), "Couldn't strobe reset bit", s);
tickdelay(DELAY_TICKS);
if ((s=sys_outb(wn->base_ctl + REG_CTL, 0)) != OK)
panic(w_name(), "Couldn't strobe reset bit", s);
tickdelay(DELAY_TICKS);
}

```

```

/* Wait for controller ready */
if (!w_waitfor(STATUS_BSY, 0)) {
    printf("%s: reset failed, drive busy\n", w_name());
    return(ERR);
}

/* The error register should be checked now, but some drives mess it up. */

for (wn = wini; wn < &wini[MAX_DRIVES]; wn++) {
    if (wn->base_cmd == w_wn->base_cmd) {
        wn->state &= ~DEAF;
        if (w_wn->irq_need_ack) {
            /* Make sure irq is actually enabled.. */
            sys_irgenable(&w_wn->irq_hook_id);
        }
    }
}

return(OK);
}

/*=====
*                                     w_intr_wait                                     *
*=====*/
PRIVATE void w_intr_wait()
{
    /* Wait for a task completion interrupt. */

    int r;
    unsigned long w_status;
    message m;

    if (w_wn->irq != NO_IRQ) {
        /* Wait for an interrupt that sets w_status to "not busy". */
        while (w_wn->w_status & (STATUS_ADMBSY|STATUS_BSY)) {
            int rr;
            if((rr=receive(ANY, &m)) != OK) { /* expect HARD_INT message */
                printf("w_intr_wait: receive from ANY failed (%d)\n",
                    r);
                continue; /* try again */
            }
            if (m.m_type == SYN_ALARM) { /* but check for timeout */
                w_timeout(); /* a.o. set w_status */
            } else if (m.m_type == HARD_INT) {
                r = sys_inb(w_wn->base_cmd + REG_STATUS, &w_status);
                if (r != 0)
                    panic("at_wini", "sys_inb failed", r);
                w_wn->w_status = w_status;
                ack_irqs(m.NOTIFY_ARG);
            } else if (m.m_type == DEV_PING) {
                notify(m.m_source);
            } else {
                printf("AT_WINI got unexpected message %d from %d\n",
                    m.m_type, m.m_source);
            }
        }
    } else {
        /* Interrupt not yet allocated; use polling. */
        (void) w_waitfor(STATUS_BSY, 0);
    }
}

/*=====
*                                     at_intr_wait                                     *
*=====*/
PRIVATE int at_intr_wait()
{
    /* Wait for an interrupt, study the status bits and return error/success. */
    int r, s;
    unsigned long inbval;

    w_intr_wait();

```

```

if ((w_wn->w_status & (STATUS_BSY | STATUS_WF | STATUS_ERR)) == 0) {
    r = OK;
} else {
    if ((s=sys_inb(w_wn->base_cmd + REG_ERROR, &inbval)) != OK)
        panic(w_name(), "Couldn't read register", s);
    if ((w_wn->w_status & STATUS_ERR) && (inbval & ERROR_BB)) {
        r = ERR_BAD_SECTOR;        /* sector marked bad, retries won't help */
    } else {
        r = ERR;                    /* any other error */
    }
}
w_wn->w_status |= STATUS_ADMBSY;    /* assume still busy with I/O */
return(r);
}

/*=====
*                               w_waitfor                               *
*=====*/
PRIVATE int w_waitfor(mask, value)
int mask;                /* status mask */
int value;                /* required status */
{
    /* Wait until controller is in the required state. Return zero on timeout.
    * An alarm that set a timeout flag is used. TIMEOUT is in micros, we need
    * ticks. Disabling the alarm is not needed, because a static flag is used
    * and a leftover timeout cannot do any harm.
    */
    unsigned long w_status;
    clock_t t0, t1;
    int s;

    getuptime(&t0);
    do {
        if ((s=sys_inb(w_wn->base_cmd + REG_STATUS, &w_status)) != OK)
            panic(w_name(), "Couldn't read register", s);
        w_wn->w_status = w_status;
        if ((w_wn->w_status & mask) == value) {
            return 1;
        }
    } while ((s=getuptime(&t1)) == OK && (t1-t0) < timeout_ticks);
    if (OK != s) printf("AT_WINI: warning, get_uptime failed: %d\n", s);

    w_need_reset();        /* controller gone deaf */
    return(0);
}

/*=====
*                               w_waitfor_dma                               *
*=====*/
PRIVATE int w_waitfor_dma(mask, value)
int mask;                /* status mask */
int value;                /* required status */
{
    /* Wait until controller is in the required state. Return zero on timeout.
    * An alarm that set a timeout flag is used. TIMEOUT is in micros, we need
    * ticks. Disabling the alarm is not needed, because a static flag is used
    * and a leftover timeout cannot do any harm.
    */
    unsigned long w_status;
    clock_t t0, t1;
    int s;

    getuptime(&t0);
    do {
        if ((s=sys_inb(w_wn->base_dma + DMA_STATUS, &w_status)) != OK)
            panic(w_name(), "Couldn't read register", s);
        if ((w_status & mask) == value) {
            return 1;
        }
    } while ((s=getuptime(&t1)) == OK && (t1-t0) < timeout_ticks);
    if (OK != s) printf("AT_WINI: warning, get_uptime failed: %d\n", s);

    return(0);
}

```

```

/*=====
 *
 *                               w_geometry
 *=====*/
PRIVATE void w_geometry(entry)
struct partition *entry;
{
    struct wini *wn = w_wn;

    if (wn->state & ATAPI) {                /* Make up some numbers. */
        entry->cylinders = div64u(wn->part[0].dv_size, SECTOR_SIZE) / (64*32);
        entry->heads = 64;
        entry->sectors = 32;
    } else {                               /* Return logical geometry. */
        entry->cylinders = wn->l cylinders;
        entry->heads = wn->l heads;
        entry->sectors = wn->l sectors;
    }
}

#if ENABLE_ATAPI
/*=====
 *
 *                               atapi_open
 *=====*/
PRIVATE int atapi_open()
{
    /* Should load and lock the device and obtain its size. For now just set the
     * size of the device to something big. What is really needed is a generic
     * SCSI layer that does all this stuff for ATAPI and SCSI devices (kjb). (XXX)
     */
    w_wn->part[0].dv_size = mul64u(800L*1024, 1024);
    return(OK);
}

/*=====
 *
 *                               atapi_close
 *=====*/
PRIVATE void atapi_close()
{
    /* Should unlock the device. For now do nothing. (XXX) */
}

void sense_request(void)
{
    int r, i;
    static u8_t sense[100], packet[ATAPI_PACKETSIZE];

    packet[0] = SCSI_SENSE;
    packet[1] = 0;
    packet[2] = 0;
    packet[3] = 0;
    packet[4] = SENSE_PACKETSIZE;
    packet[5] = 0;
    packet[7] = 0;
    packet[8] = 0;
    packet[9] = 0;
    packet[10] = 0;
    packet[11] = 0;

    for(i = 0; i < SENSE_PACKETSIZE; i++) sense[i] = 0xff;
    r = atapi_sendpacket(packet, SENSE_PACKETSIZE);
    if (r != OK) { printf("request sense command failed\n"); return; }
    if (atapi_intr_wait() <= 0) { printf("WARNING: request response failed\n"); }

    if (sys_insw(w_wn->base_cmd + REG_DATA, SELF, (void *) sense, SENSE_PACKETSIZE) !
= OK)
        printf("WARNING: sense reading failed\n");

    printf("sense data:");
    for(i = 0; i < SENSE_PACKETSIZE; i++) printf(" %02x", sense[i]);
    printf("\n");
}

/*=====

```

```

*                               atapi_transfer                               *
*=====*/
PRIVATE int atapi_transfer(proc_nr, opcode, position, iov, nr_req)
int proc_nr;                /* process doing the request */
int opcode;                 /* DEV_GATHER or DEV_SCATTER */
off_t position;             /* offset on device to read or write */
iovec_t *iov;               /* pointer to read or write request vector */
unsigned nr_req;            /* length of request vector */
{
    struct wini *wn = w_wn;
    iovec_t *iop, *iop_end = iov + nr_req;
    int r, s, errors, fresh;
    u64_t pos;
    unsigned long block;
    unsigned long dv_size = cv64ul(w_dv->dv_size);
    unsigned nbytes, nblocks, count, before, chunk;
    static u8_t packet[ATAPI_PACKETSIZE];

    errors = fresh = 0;

    while (nr_req > 0 && !fresh) {
        /* The Minix block size is smaller than the CD block size, so we
         * may have to read extra before or after the good data.
         */
        pos = add64ul(w_dv->dv_base, position);
        block = div64u(pos, CD_SECTOR_SIZE);
        before = rem64u(pos, CD_SECTOR_SIZE);

        /* How many bytes to transfer? */
        nbytes = count = 0;
        for (iop = iov; iop < iop_end; iop++) {
            nbytes += iop->iov_size;
            if ((before + nbytes) % CD_SECTOR_SIZE == 0) count = nbytes;
        }

        /* Does one of the memory chunks end nicely on a CD sector multiple? */
        if (count != 0) nbytes = count;

        /* Data comes in as words, so we have to enforce even byte counts. */
        if ((before | nbytes) & 1) return(EINVAL);

        /* Which block on disk and how close to EOF? */
        if (position >= dv_size) return(OK);          /* At EOF */
        if (position + nbytes > dv_size) nbytes = dv_size - position;

        nblocks = (before + nbytes + CD_SECTOR_SIZE - 1) / CD_SECTOR_SIZE;
        if (ATAPI_DEBUG) {
            printf("block=%lu, before=%u, nbytes=%u, nblocks=%u\n",
                block, before, nbytes, nblocks);
        }

        /* First check to see if a reinitialization is needed. */
        if (!(wn->state & INITIALIZED) && w_specify() != OK) return(EIO);

        /* Build an ATAPI command packet. */
        packet[0] = SCSI_READ10;
        packet[1] = 0;
        packet[2] = (block >> 24) & 0xFF;
        packet[3] = (block >> 16) & 0xFF;
        packet[4] = (block >> 8) & 0xFF;
        packet[5] = (block >> 0) & 0xFF;
        packet[7] = (nblocks >> 8) & 0xFF;
        packet[8] = (nblocks >> 0) & 0xFF;
        packet[9] = 0;
        packet[10] = 0;
        packet[11] = 0;

        /* Tell the controller to execute the packet command. */
        r = atapi_sendpacket(packet, nblocks * CD_SECTOR_SIZE);
        if (r != OK) goto err;

        /* Read chunks of data. */
        while ((r = atapi_intr_wait()) > 0) {
            count = r;

```



```

        if (ATAPI_DEBUG) {
            printf("before=%u, nbytes=%u, count=%u\n",
                before, nbytes, count);
        }

        while (before > 0 && count > 0) {           /* Discard before. */
            chunk = before;
            if (chunk > count) chunk = count;
            if (chunk > DMA_BUF_SIZE) chunk = DMA_BUF_SIZE;
            if ((s=sys_insw(wn->base_cmd + REG_DATA, SELF, tmp_buf, chunk)) != OK)
                panic(w_name(), "Call to sys_insw() failed", s);
            before -= chunk;
            count -= chunk;
        }

        while (nbytes > 0 && count > 0) {           /* Requested data. */
            chunk = nbytes;
            if (chunk > count) chunk = count;
            if (chunk > iov->iov_size) chunk = iov->iov_size;
            if ((s=sys_insw(wn->base_cmd + REG_DATA, proc_nr, (void *) iov->iov_addr, chunk))
                != OK)
                panic(w_name(), "Call to sys_insw() failed", s);
            position += chunk;
            nbytes -= chunk;
            count -= chunk;
            iov->iov_addr += chunk;
            fresh = 0;
            if ((iov->iov_size -= chunk) == 0) {
                iov++;
                nr_req--;
                fresh = 1;           /* new element is optional */
            }
        }

        while (count > 0) {                         /* Excess data. */
            chunk = count;
            if (chunk > DMA_BUF_SIZE) chunk = DMA_BUF_SIZE;
            if ((s=sys_insw(wn->base_cmd + REG_DATA, SELF, tmp_buf, chunk)) != OK)
                panic(w_name(), "Call to sys_insw() failed", s);
            count -= chunk;
        }

        if (r < 0) {
err:        /* Don't retry if too many errors. */
            if (atapi_debug) sense_request();
            if (++errors == max_errors) {
                w_command = CMD_IDLE;
                if (atapi_debug) printf("giving up (%d)\n", errors);
                return(EIO);
            }
            if (atapi_debug) printf("retry (%d)\n", errors);
        }
    }

    w_command = CMD_IDLE;
    return(OK);
}

/*=====
 *                               atapi_sendpacket                               *
 *=====*/
PRIVATE int atapi_sendpacket(packet, cnt)
u8_t *packet;
unsigned cnt;
{
    /* Send an Atapi Packet Command */
    struct wini *wn = w_wn;
    pvb_pair_t outbyte[6];           /* vector for sys_voutb() */
    int s;

    if (wn->state & IGNORING) return ERR;

```

```

/* Select Master/Slave drive */
if ((s=sys_outb(wn->base_cmd + REG_DRIVE, wn->ldhpref)) != OK)
    panic(w_name(), "Couldn't select master/ slave drive", s);

if (!w_waitfor(STATUS_BSY | STATUS_DRQ, 0)) {
    printf("%s: atapi_sendpacket: drive not ready\n", w_name());
    return(ERR);
}

/* Schedule a wakeup call, some controllers are flaky. This is done with
 * a synchronous alarm. If a timeout occurs a SYN_ALARM message is sent
 * from HARDWARE, so that w_intr_wait() can call w_timeout() in case the
 * controller was not able to execute the command. Leftover timeouts are
 * simply ignored by the main loop.
 */
sys_setalarm(wakeup_ticks, 0);

#ifdef _WORD_SIZE > 2
    if (cnt > 0xFFFF) cnt = 0xFFFF; /* Max data per interrupt. */
#endif

w_command = ATAPI_PACKETCMD;
pv_set(outbyte[0], wn->base_cmd + REG_FEAT, 0);
pv_set(outbyte[1], wn->base_cmd + REG_IRR, 0);
pv_set(outbyte[2], wn->base_cmd + REG_SAMTAG, 0);
pv_set(outbyte[3], wn->base_cmd + REG_CNT_LO, (cnt >> 0) & 0xFF);
pv_set(outbyte[4], wn->base_cmd + REG_CNT_HI, (cnt >> 8) & 0xFF);
pv_set(outbyte[5], wn->base_cmd + REG_COMMAND, w_command);
if (atapi_debug) printf("cmd:%x ", w_command);
if ((s=sys_voutb(outbyte,6)) != OK)
    panic(w_name(), "Couldn't write registers with sys_voutb()", s);

if (!w_waitfor(STATUS_BSY | STATUS_DRQ, STATUS_DRQ)) {
    printf("%s: timeout (BSY|DRQ -> DRQ)\n", w_name());
    return(ERR);
}
wn->w_status |= STATUS_ADMSY; /* Command not at all done yet. */

/* Send the command packet to the device. */
if ((s=sys_outsw(wn->base_cmd + REG_DATA, SELF, packet, ATAPI_PACKETSIZE)) != OK)
    panic(w_name(), "sys_outsw() failed", s);

{
    int p;
    if (atapi_debug) {
        printf("sent command:");
        for(p = 0; p < ATAPI_PACKETSIZE; p++) { printf(" %02x", packet[p]); }
        printf("\n");
    }
}
return(OK);
}

#endif /* ENABLE_ATAPI */

/*=====
 *                                w_other                                *
 *=====*/
PRIVATE int w_other(dr, m)
struct driver *dr;
message *m;
{
    int r, timeout, prev;

    if (m->m_type != DEV_IOCTL ) {
        return EINVAL;
    }

    if (m->REQUEST == DIOCTIMEOUT) {
        if ((r=sys_datacopy(m->IO_ENDPT, (vir_bytes)m->ADDRESS,
            SELF, (vir_bytes)&timeout, sizeof(timeout))) != OK)
            return r;
    }
}

```

```

        if (timeout == 0) {
            /* Restore defaults. */
            timeout_ticks = DEF_TIMEOUT_TICKS;
            max_errors = MAX_ERRORS;
            wakeup_ticks = WAKEUP;
            w_silent = 0;
        } else if (timeout < 0) {
            return EINVAL;
        } else {
            prev = wakeup_ticks;

            if (!w_standard_timeouts) {
                /* Set (lower) timeout, lower error
                 * tolerance and set silent mode.
                 */
                wakeup_ticks = timeout;
                max_errors = 3;
                w_silent = 1;

                if (timeout_ticks > timeout)
                    timeout_ticks = timeout;
            }

            if ((r=sys_datacopy(SELF, (vir_bytes)&prev,
                               m->IO_ENDPT, (vir_bytes)m->ADDRESS, sizeof(prev))) != OK)
                return r;
        }

        return OK;
    } else if (m->REQUEST == DIOCOPENCT) {
        int count;
        if (w_prepare(m->DEVICE) == NIL_DEV) return ENXIO;
        count = w_wn->open_ct;
        if ((r=sys_datacopy(SELF, (vir_bytes)&count,
                           m->IO_ENDPT, (vir_bytes)m->ADDRESS, sizeof(count))) != OK)
            return r;
        return OK;
    }
    return EINVAL;
}

/*=====
 *                               w_hw_int                               *
 *=====*/
PRIVATE int w_hw_int(dr, m)
struct driver *dr;
message *m;
{
    /* Leftover interrupt(s) received; ack it/them. */
    ack_irqs(m->NOTIFY_ARG);

    return OK;
}

/*=====
 *                               ack_irqs                               *
 *=====*/
PRIVATE void ack_irqs(unsigned int irqs)
{
    unsigned int drive;
    unsigned long w_status;

    for (drive = 0; drive < MAX_DRIVES && irqs; drive++) {
        if (!(wini[drive].state & IGNORING) && wini[drive].irq_need_ack &&
            (wini[drive].irq_mask & irqs)) {
            if (sys_inb((wini[drive].base_cmd + REG_STATUS),
                       &w_status) != OK)
            {
                panic(w_name(), "couldn't ack irq on drive %d\n",
                     drive);
            }
            wini[drive].w_status = w_status;
            if (sys_irgenable(&wini[drive].irq_hook_id) != OK)

```

```

        printf("couldn't re-enable drive %d\n", drive);
        irqs &= ~wini[drive].irq_mask;
    }
}

#define STSTR(a) if (status & STATUS_ ## a) { strcat(str, #a); strcat(str, " "); }
#define ERRSTR(a) if (e & ERROR_ ## a) { strcat(str, #a); strcat(str, " "); }
char *strstatus(int status)
{
    static char str[200];
    str[0] = '\0';

    STSTR(BSY);
    STSTR(DRDY);
    STSTR(DMADF);
    STSTR(SRVCDSC);
    STSTR(DRQ);
    STSTR(CORR);
    STSTR(CHECK);
    return str;
}

char *strerr(int e)
{
    static char str[200];
    str[0] = '\0';

    ERRSTR(BB);
    ERRSTR(ECC);
    ERRSTR(ID);
    ERRSTR(AC);
    ERRSTR(TK);
    ERRSTR(DM);

    return str;
}

#if ENABLE_ATAPI

/*=====
 *                               atapi_intr_wait                               *
 *=====*/
PRIVATE int atapi_intr_wait()
{
    /* Wait for an interrupt and study the results. Returns a number of bytes
     * that need to be transferred, or an error code.
     */
    struct wini *wn = w_wn;
    pvb_pair_t inbyte[4];          /* vector for sys_vinb() */
    int s;                        /* status for sys_vinb() */
    int e;
    int len;
    int irr;
    int r;
    int phase;

    w_intr_wait();

    /* Request series of device I/O. */
    inbyte[0].port = wn->base_cmd + REG_ERROR;
    inbyte[1].port = wn->base_cmd + REG_CNT_LO;
    inbyte[2].port = wn->base_cmd + REG_CNT_HI;
    inbyte[3].port = wn->base_cmd + REG_IRR;
    if ((s=sys_vinb(inbyte, 4)) != OK)
        panic(w_name(), "ATAPI failed sys_vinb()", s);
    e = inbyte[0].value;
    len = inbyte[1].value;
    len |= inbyte[2].value << 8;
    irr = inbyte[3].value;

#if ATAPI_DEBUG
    printf("wn %p S=%x=%s E=%02x=%s L=%04x I=%02x\n", wn, wn->w_status, strstatus(wn->w_

```

```
status), e, strerror(e), len, irr);
#endif
    if (wn->w_status & (STATUS_BSY | STATUS_CHECK)) {
        if (ataapi_debug) {
            printf("ataapi fail: S=%x=%s E=%02x=%s L=%04x I=%02x\n", wn->w_status, strstatus(w
n->w_status), e, strerror(e), len, irr);
        }
        return ERR;
    }

    phase = (wn->w_status & STATUS_DRQ) | (irr & (IRR_COD | IRR_IO));

    switch (phase) {
    case IRR_COD | IRR_IO:
        if (ATAPI_DEBUG) printf("ACD: Phase Command Complete\n");
        r = OK;
        break;
    case 0:
        if (ATAPI_DEBUG) printf("ACD: Phase Command Aborted\n");
        r = ERR;
        break;
    case STATUS_DRQ | IRR_COD:
        if (ATAPI_DEBUG) printf("ACD: Phase Command Out\n");
        r = ERR;
        break;
    case STATUS_DRQ:
        if (ATAPI_DEBUG) printf("ACD: Phase Data Out %d\n", len);
        r = len;
        break;
    case STATUS_DRQ | IRR_IO:
        if (ATAPI_DEBUG) printf("ACD: Phase Data In %d\n", len);
        r = len;
        break;
    default:
        if (ATAPI_DEBUG) printf("ACD: Phase Unknown\n");
        r = ERR;
        break;
    }

    #if 0
    /* retry if the media changed */
    XXX while (phase == (IRR_IO | IRR_COD) && (wn->w_status & STATUS_CHECK)
        && (e & ERROR_SENSE) == SENSE_UATTN && --try > 0);
    #endif

    wn->w_status |= STATUS_ADMSY;          /* Assume not done yet. */
    return(r);
}
#endif /* ENABLE_ATAPI */
```

```
#include "../drivers.h"
#include "../libdriver/driver.h"
#include "../libdriver/drvlib.h"

_PROTOTYPE(int main, (void));

#define VERBOSE          0    /* display identify messages during boot */
#define ENABLE_ATAPI     1    /* add ATAPI cd-rom support to driver */
```

```
# Makefile for the BIOS disk driver (BIOS_WINI)
DRIVER = bios_wini

# directories
u = /usr
i = $u/include
s = $i/sys
m = $i/minix
b = $i/ibm
d = ..

# programs, flags, etc.
MAKE = exec make
CC = exec cc
CFLAGS = -I$i
LDFLAGS = -i
LIBS = -lsysutil -lsys -ltimers

OBJ = bios_wini.o
LIBDRIVER = $d/libdriver/driver.o $d/libdriver/drvlib.o

# build local binary
all build: $(DRIVER)
$(DRIVER): $(OBJ) $(LIBDRIVER)
    $(CC) -o $@ $(LDFLAGS) $(OBJ) $(LIBDRIVER) $(LIBS)
    install -S 4k $(DRIVER)

$(LIBDRIVER):
    cd $d/libdriver && $(MAKE)

# install with other drivers
install: /sbin/$(DRIVER)
/sbin/$(DRIVER): $(DRIVER)
    install -o root -cs $? $@

# clean up local files
clean:
    rm -f $(DRIVER) *.o *.bak

depend:
    /usr/bin/mkdep "$(CC) -E $(CPPFLAGS)" *.c ../libdriver/*.c > .depend

# Include generated dependencies.
include .depend
```

```

/* This file contains the "device dependent" part of a hard disk driver that
 * uses the ROM BIOS. It makes a call and just waits for the transfer to
 * happen. It is not interrupt driven and thus will (*) have poor performance.
 * The advantage is that it should work on virtually any PC, XT, 386, PS/2
 * or clone. The demo disk uses this driver. It is suggested that all
 * MINIX users try the other drivers, and use this one only as a last resort,
 * if all else fails.
 *
 * (*) The performance is within 10% of the AT driver for reads on any disk
 * and writes on a 2:1 interleaved disk, it will be DMA_BUF_SIZE bytes
 * per revolution for a minimum of 60 kb/s for writes to 1:1 disks.
 *
 * The file contains one entry point:
 *
 *      bios_winchester_task:  main entry when system is brought up
 *
 * Changes:
 *      30 Apr 1992 by Kees J. Bot: device dependent/independent split.
 *      14 May 2000 by Kees J. Bot: d-d/i rewrite.
 */

```

```

#include "../drivers.h"
#include "../libdriver/driver.h"
#include "../libdriver/drvlib.h"
#include <minix/sysutil.h>
#include <minix/keymap.h>
#include <sys/ioc_disk.h>
#include <ibm/int86.h>
#include <assert.h>

#define ME "BIOS_WINI"

/* Error codes */
#define ERR          (-1)  /* general error */

/* Parameters for the disk drive. */
#define MAX_DRIVES      8    /* this driver supports 8 drives (d0 - d7) */
#define MAX_SECS        255  /* bios can transfer this many sectors */
#define NR_MINORS        (MAX_DRIVES * DEV_PER_DRIVE)
#define SUB_PER_DRIVE    (NR_PARTITIONS * NR_PARTITIONS)
#define NR_SUBDEVS       (MAX_DRIVES * SUB_PER_DRIVE)

PRIVATE int pc_at = 1; /* What about PC XTs? */

/* Variables. */
PRIVATE struct wini {
    unsigned cylinders; /* number of cylinders */
    unsigned heads;     /* number of heads */
    unsigned sectors;    /* number of sectors per track */
    unsigned open_ct;    /* in-use count */
    int drive_id;        /* Drive ID at BIOS level */
    int present;         /* Valid drive */
    int int13ext;        /* IBM/MS INT 13 extensions supported? */
    struct device part[DEV_PER_DRIVE]; /* disks and partitions */
    struct device subpart[SUB_PER_DRIVE]; /* subpartitions */
} wini[MAX_DRIVES], *w_wn;

PRIVATE int w_drive; /* selected drive */
PRIVATE struct device *w_dv; /* device's base and size */
PRIVATE vir_bytes bios_buf_vir, bios_buf_size;
PRIVATE phys_bytes bios_buf_phys;
PRIVATE int remap_first = 0; /* Remap drives for CD HD emulation */

_PROTOTYPE(int main, (void) );
FORWARD _PROTOTYPE( struct device *w_prepare, (int device) );
FORWARD _PROTOTYPE( char *w_name, (void) );
FORWARD _PROTOTYPE( int w_transfer, (int proc_nr, int opcode, off_t position,
                                     iovec_t *iov, unsigned nr_req) );
FORWARD _PROTOTYPE( int w_do_open, (struct driver *dp, message *m_ptr) );
FORWARD _PROTOTYPE( int w_do_close, (struct driver *dp, message *m_ptr) );
FORWARD _PROTOTYPE( void w_init, (void) );
FORWARD _PROTOTYPE( void w_geometry, (struct partition *entry) );
FORWARD _PROTOTYPE( int w_other, (struct driver *dp, message *m_ptr) );

```



```

/* Entry points to this driver. */
PRIVATE struct driver w_dtab = {
    w_name,          /* current device's name */
    w_do_open,       /* open or mount request, initialize device */
    w_do_close,      /* release device */
    do_diocntl,      /* get or set a partition's geometry */
    w_prepare,       /* prepare for I/O on a given minor device */
    w_transfer,      /* do the I/O */
    nop_cleanup,     /* no cleanup needed */
    w_geometry,      /* tell the geometry of the disk */
    nop_signal,      /* no cleanup needed on shutdown */
    nop_alarm,       /* ignore leftover alarms */
    nop_cancel,      /* ignore CANCELs */
    nop_select,      /* ignore selects */
    w_other,         /* catch-all for unrecognized commands and ioctls */
    NULL,            /* leftover hardware interrupts */
};

/*=====
*
*                               bios_winchester_task
*=====*/
PUBLIC int main()
{
    long v;
    struct sigaction sa;

    sa.sa_handler = SIG_MESS;
    sigemptyset(&sa.sa_mask);
    sa.sa_flags = 0;
    if (sigaction(SIGTERM,&sa,NULL)<0) panic("BIOS","sigaction failed", errno);

    v= 0;
    env_parse("bios_remap_first", "d", 0, &v, 0, 1);
    remap_first= v;

    /* Set special disk parameters then call the generic main loop. */
    driver_task(&w_dtab);
    return(OK);
}

/*=====
*
*                               w_prepare
*=====*/
PRIVATE struct device *w_prepare(device)
int device;
{
    /* Prepare for I/O on a device. */

    if (device < NR_MINORS) {
        w_drive = device / DEV_PER_DRIVE; /* d0, d0p[0-3], d1, ... */
        w_wn = &wini[w_drive];           /* save drive number */
        w_dv = &w_wn->part[device % DEV_PER_DRIVE];
    } else
    if ((unsigned) (device - MINOR_d0p0s0) < NR_SUBDEVS) { /* d[0-7]p[0-3]s[0-3] */
        w_drive = device / SUB_PER_DRIVE;
        w_wn = &wini[w_drive];
        w_dv = &w_wn->subpart[device % SUB_PER_DRIVE];
    } else {
        return(NIL_DEV);
    }
    if (w_drive >= MAX_DRIVES || !w_wn->present)
        return NIL_DEV;
    return(w_dv);
}

/*=====
*
*                               w_name
*=====*/
PRIVATE char *w_name()
{
    /* Return a name for the current device. */
    static char name[] = "bios-d0";

```

```

    name[6] = '0' + w_drive;
    return name;
}

/*=====
 *                               w_transfer                               *
 *=====*/
PRIVATE int w_transfer(proc_nr, opcode, position, iov, nr_req)
int proc_nr;           /* process doing the request */
int opcode;            /* DEV_GATHER or DEV_SCATTER */
off_t position;        /* offset on device to read or write */
iovec_t *iov;          /* pointer to read or write request vector */
unsigned nr_req;       /* length of request vector */
{
    struct wini *wn = w_wn;
    iovec_t *iop, *iop_end = iov + nr_req;
    int r, errors;
    unsigned nbytes, count, chunk;
    unsigned long block;
    vir_bytes il3e_rw_off, rem_buf_size;
    unsigned long dv_size = cv64ul(w_dv->dv_size);
    unsigned secspcyl = wn->heads * wn->sectors;
    struct int13ext_rw {
        u8_t    len;
        u8_t    res1;
        u16_t    count;
        u16_t    addr[2];
        u32_t    block[2];
    } il3e_rw;
    struct reg86u reg86;

    /* Check disk address. */
    if ((position & SECTOR_MASK) != 0) return(EINVAL);

    errors = 0;

    il3e_rw_off= bios_buf_size-sizeof(il3e_rw);
    rem_buf_size= (il3e_rw_off & ~SECTOR_MASK);
    assert(rem_buf_size != 0);

    while (nr_req > 0) {
        /* How many bytes to transfer? */
        nbytes = 0;
        for (iop = iov; iop < iop_end; iop++) {
            if (nbytes + iop->iov_size > rem_buf_size) {
                /* Don't do half a segment if you can avoid it. */
                if (nbytes == 0) nbytes = rem_buf_size;
                break;
            }
            nbytes += iop->iov_size;
        }
        if ((nbytes & SECTOR_MASK) != 0) return(EINVAL);

        /* Which block on disk and how close to EOF? */
        if (position >= dv_size) return(OK);           /* At EOF */
        if (position + nbytes > dv_size) nbytes = dv_size - position;
        block = div64u(add64ul(w_dv->dv_base, position), SECTOR_SIZE);

        /* Degrade to per-sector mode if there were errors. */
        if (errors > 0) nbytes = SECTOR_SIZE;

        if (opcode == DEV_SCATTER) {
            /* Copy from user space to the DMA buffer. */
            count = 0;
            for (iop = iov; count < nbytes; iop++) {
                chunk = iop->iov_size;
                if (count + chunk > nbytes) chunk = nbytes - count;
                assert(chunk <= rem_buf_size);
                r = sys_vircopy(proc_nr, D, iop->iov_addr,
                               SYSTEM, D, bios_buf_vir+count,
                               chunk);
                if (r != OK)
                    panic(ME, "sys_vircopy failed", r);
                count += chunk;
            }
        }
    }
}

```

```

    }
}

/* Do the transfer */
if (wn->int13ext) {
    il3e_rw.len = 0x10;
    il3e_rw.res1 = 0;
    il3e_rw.count = nbytes >> SECTOR_SHIFT;
    il3e_rw.addr[0] = bios_buf_phys % HCLICK_SIZE;
    il3e_rw.addr[1] = bios_buf_phys / HCLICK_SIZE;
    il3e_rw.block[0] = block;
    il3e_rw.block[1] = 0;
    r = sys_vircopy(SELFS, D, (vir_bytes)&il3e_rw,
        SYSTEM, D, (bios_buf_vir+il3e_rw_off),
        sizeof(il3e_rw));
    if (r != OK)
        panic(ME, "sys_vircopy failed", r);

    /* Set up an extended read or write BIOS call. */
    reg86.u.b.intno = 0x13;
    reg86.u.w.ax = opcode == DEV_SCATTER ? 0x4300 : 0x4200;
    reg86.u.b.dl = wn->drive_id;
    reg86.u.w.si = (bios_buf_phys + il3e_rw_off) % HCLICK_SIZE;
    reg86.u.w.ds = (bios_buf_phys + il3e_rw_off) / HCLICK_SIZE;
} else {
    /* Set up an ordinary read or write BIOS call. */
    unsigned cylinder = block / secspcyl;
    unsigned sector = (block % wn->sectors) + 1;
    unsigned head = (block % secspcyl) / wn->sectors;

    reg86.u.b.intno = 0x13;
    reg86.u.b.ah = opcode == DEV_SCATTER ? 0x03 : 0x02;
    reg86.u.b.al = nbytes >> SECTOR_SHIFT;
    reg86.u.w.bx = bios_buf_phys % HCLICK_SIZE;
    reg86.u.w.es = bios_buf_phys / HCLICK_SIZE;
    reg86.u.b.ch = cylinder & 0xFF;
    reg86.u.b.cl = sector | ((cylinder & 0x300) >> 2);
    reg86.u.b.dh = head;
    reg86.u.b.dl = wn->drive_id;
}

r = sys_int86(&reg86);
if (r != OK)
    panic(ME, "BIOS call failed", r);

if (reg86.u.w.f & 0x0001) {
    /* An error occurred, try again sector by sector unless */
    if (++errors == 2) return(EIO);
    continue;
}

if (opcode == DEV_GATHER) {
    /* Copy from the DMA buffer to user space. */
    count = 0;
    for (iop = iov; count < nbytes; iop++) {
        chunk = iov->iov_size;
        if (count + chunk > nbytes) chunk = nbytes - count;
        assert(chunk <= rem_buf_size);
        r = sys_vircopy(SYSTEM, D, bios_buf_vir+count,
            proc_nr, D, iop->iov_addr,
            chunk);
        if (r != OK)
            panic(ME, "sys_vircopy failed", r);
        count += chunk;
    }
}

/* Book the bytes successfully transferred. */
position += nbytes;
for (;;) {
    if (nbytes < iov->iov_size) {
        /* Not done with this one yet. */
        iov->iov_addr += nbytes;
        iov->iov_size -= nbytes;
    }
}

```

```

        break;
    }
    nbytes -= iov->iov_size;
    iov->iov_addr += iov->iov_size;
    iov->iov_size = 0;
    if (nbytes == 0) {
        /* The rest is optional, so we return to give FS a
         * chance to think it over.
         */
        return(OK);
    }
    iov++;
    nr_req--;
}
}
return(OK);
}

/*=====
 *                               w_do_open                               *
 *=====*/
PRIVATE int w_do_open(dp, m_ptr)
struct driver *dp;
message *m_ptr;
{
    /* Device open: Initialize the controller and read the partition table. */

    static int init_done = FALSE;

    if (!init_done) { w_init(); init_done = TRUE; }

    if (w_prepare(m_ptr->DEVICE) == NIL_DEV) return(ENXIO);

    if (w_wn->open_ct++ == 0) {
        /* Partition the disk. */
        partition(&w_dtab, w_drive * DEV_PER_DRIVE, P_PRIMARY, 0);
    }
    return(OK);
}

/*=====
 *                               w_do_close                             *
 *=====*/
PRIVATE int w_do_close(dp, m_ptr)
struct driver *dp;
message *m_ptr;
{
    /* Device close: Release a device. */

    if (w_prepare(m_ptr->DEVICE) == NIL_DEV) return(ENXIO);
    w_wn->open_ct--;
    return(OK);
}

/*=====
 *                               w_init                                *
 *=====*/
PRIVATE void w_init()
{
    /* This routine is called at startup to initialize the drive parameters. */

    int r, drive, drive_id, nr_drives;
    struct wini *wn;
    unsigned long capacity;
    struct int13ext_params {
        u16_t    len;
        u16_t    flags;
        u32_t    cylinders;
        u32_t    heads;
        u32_t    sectors;
        u32_t    capacity[2];
        u16_t    bts_per_sec;
        u16_t    config[2];
    } il3e_par;

```

```
struct reg86u reg86;

/* Ask the system task for a suitable buffer */
r= sys_getbiosbuffer(&bios_buf_vir, &bios_buf_size);
if (r != OK)
    panic(ME, "sys_getbiosbuffer failed", r);
r= sys_umap(SYSTEM, D, (vir_bytes)bios_buf_vir, (phys_bytes)bios_buf_size,
    &bios_buf_phys);
if (r != OK)
    panic(ME, "sys_umap failed", r);
if (bios_buf_phys+bios_buf_size > 0x100000)
    panic(ME, "bad BIOS buffer, phys", bios_buf_phys);
#if 0
    printf("bios_wini: got buffer size %d, virtual 0x%x, phys 0x%x\n",
        bios_buf_size, bios_buf_vir, bios_buf_phys);
#endif

/* Get the geometry of the drives */
for (drive = 0; drive < MAX_DRIVES; drive++) {
    if (remap_first)
    {
        if (drive == 7)
            drive_id= 0x80;
        else
            drive_id= 0x80 + drive + 1;
    }
    else
        drive_id= 0x80 + drive;

    (void) w_prepare(drive * DEV_PER_DRIVE);
    wn = w_wn;
    wn->drive_id= drive_id;

    reg86.u.b.intno = 0x13;
    reg86.u.b.ah = 0x08; /* Get drive parameters. */
    reg86.u.b.dl = drive_id;
    r= sys_int86(&reg86);
    if (r != OK)
        panic(ME, "BIOS call failed", r);

    nr_drives = !(reg86.u.w.f & 0x0001) ? reg86.u.b.dl : drive;
    if (drive_id >= 0x80 + nr_drives) continue;
    wn->present= 1;

    wn->heads = reg86.u.b.dh + 1;
    wn->sectors = reg86.u.b.cl & 0x3F;
    wn->cylinders = (reg86.u.b.ch | ((reg86.u.b.cl & 0xC0) << 2)) + 1;

    capacity = (unsigned long) wn->cylinders * wn->heads * wn->sectors;

    reg86.u.b.intno = 0x13;
    reg86.u.b.ah = 0x41; /* INT 13 Extensions - Installation check */
    reg86.u.w.bx = 0x55AA;
    reg86.u.b.dl = drive_id;

    if (pc_at) {
        r= sys_int86(&reg86);
        if (r != OK)
            panic(ME, "BIOS call failed", r);
    }

    if (!(reg86.u.w.f & 0x0001) && reg86.u.w.bx == 0xAA55
        && (reg86.u.w.cx & 0x0001)) {
        /* INT 13 Extensions available. */
        il3e_par.len = 0x001E; /* Input size of parameter packet */
        r= sys_vircopy(SELF, D, (vir_bytes)&il3e_par,
            SYSTEM, D, bios_buf_vir,
            sizeof(il3e_par));
        if (r != OK)
            panic(ME, "sys_vircopy failed\n", r);
        reg86.u.b.intno = 0x13;
        reg86.u.b.ah = 0x48; /* Ext. Get drive parameters. */
        reg86.u.b.dl = drive_id;
        reg86.u.w.si = bios_buf_phys % HCLICK_SIZE;
    }
}
```

```

    reg86.u.w.ds = bios_buf_phys / HCLICK_SIZE;

    r= sys_int86(&reg86);
    if (r != OK)
        panic(ME, "BIOS call failed", r);

    r= sys_vircopy(SYSTEM, D, bios_buf_vir,
        SELF, D, (vir_bytes)&i13e_par,
        sizeof(i13e_par));
    if (r != OK)
        panic(ME, "sys_vircopy failed\n", r);

    if (!(reg86.u.w.f & 0x0001)) {
        wn->int13ext = 1;          /* Extensions can be used. */
        capacity = i13e_par.capacity[0];
        if (i13e_par.capacity[1] != 0) capacity = 0xFFFFFFFF;
    }
}

if (wn->int13ext) {
    printf("%s: %lu sectors\n", w_name(), capacity);
} else {
    printf("%s: %d cylinders, %d heads, %d sectors per track\n",
        w_name(), wn->cylinders, wn->heads, wn->sectors);
}
wn->part[0].dv_size = mul64u(capacity, SECTOR_SIZE);
}
}

/*=====
 *                                w_geometry                                *
 *=====*/
PRIVATE void w_geometry(entry)
struct partition *entry;
{
    entry->cylinders = w_wn->cylinders;
    entry->heads = w_wn->heads;
    entry->sectors = w_wn->sectors;
}

/*=====
 *                                w_other                                *
 *=====*/
PRIVATE int w_other(dr, m)
struct driver *dr;
message *m;
{
    int r, timeout, prev;

    if (m->m_type != DEV_IOCTL ) {
        return EINVAL;
    }

    if (m->REQUEST == DIOCOPENCT) {
        int count;
        if (w_prepare(m->DEVICE) == NIL_DEV) return ENXIO;
        count = w_wn->open_ct;
        if ((r=sys_datacopy(SELF, (vir_bytes)&count,
            m->IO_ENDPT, (vir_bytes)m->ADDRESS, sizeof(count))) != OK)
            return r;
        return OK;
    }

    return EINVAL;
}
}

```

```
# Makefile for the CMOS driver
DRIVER = cmos

# directories
u = /usr
i = $u/include
s = $i/sys
m = $i/minix
b = $i/ibm
d = ..

# programs, flags, etc.
MAKE = exec make
CC = exec cc
CFLAGS = -I$i
LDFLAGS = -i
LIBS = -lsys -lsysutil

OBJ = cmos.o
LIBDRIVER = $d/libdriver/driver.o

# build local binary
all build: $(DRIVER)
$(DRIVER): $(OBJ) $(LIBDRIVER)
    $(CC) -o $@ $(LDFLAGS) $(OBJ) $(LIBDRIVER) $(LIBS)
    install -S 1024w $(DRIVER)

$(LIBDRIVER):
    cd $d/libdriver && $(MAKE)

# install with other drivers
install: /sbin/$(DRIVER)
/sbin/$(DRIVER): $(DRIVER)
    install -o root -cs $? $@

# clean up local files
clean:
    rm -f $(DRIVER) *.o *.bak

depend:
    /usr/bin/mkdep "$(CC) -E $(CPPFLAGS)" *.c ../libdriver/*.c > .depend

# Include generated dependencies.
include .depend
```

```

/* This file contains a device driver that can access the CMOS chip to
 * get or set the system time. It drives the special file:
 *
 *      /dev/cmos          - CMOS chip
 *
 * Changes:
 *      Aug 04, 2005      Created. Read CMOS time. (Jorrit N. Herder)
 *
 * Manufacturers usually use the ID value of the IBM model they emulate.
 * However some manufacturers, notably HP and COMPAQ, have had different
 * ideas in the past.
 *
 * Machine ID byte information source:
 *      _The Programmer's PC Sourcebook_ by Thom Hogan,
 *      published by Microsoft Press
 */

#include "../drivers.h"
#include <sys/ioc_cmos.h>
#include <time.h>
#include <ibm/cmos.h>
#include <ibm/bios.h>

extern int errno;                                /* error number for PM calls */

FORWARD _PROTOTYPE( int gettime, (int who, int y2kflag, vir_bytes dst_time));
FORWARD _PROTOTYPE( void reply, (int reply, int replyee, int proc, int s));

FORWARD _PROTOTYPE( int read_register, (int register_address));
FORWARD _PROTOTYPE( int get_cmostime, (struct tm *tmp, int y2kflag));
FORWARD _PROTOTYPE( int dec_to_bcd, (int dec));
FORWARD _PROTOTYPE( int bcd_to_dec, (int bcd));

/*=====
 *
 *                                main
 *=====*/
PUBLIC void main(void)
{
    message m;
    int y2kflag;
    int result;
    int suspended = NONE;
    int s;

    while(TRUE) {

        /* Get work. */
        if (OK != (s=receive(ANY, &m)))
            panic("CMOS", "attempt to receive work failed", s);

        /* Handle request. */
        switch(m.m_type) {

            case DEV_OPEN:
            case DEV_CLOSE:
            case CANCEL:
                reply(TASK_REPLY, m.m_source, m.IO_ENDPT, OK);
                break;

            case DEV_PING:
                notify(m.m_source);
                break;

            case DEV_IOCTL:

                /* Probably best to SUSPEND the caller, CMOS I/O has nasty timeouts.
                 * This way we don't block the rest of the system. First check if
                 * another process is already suspended. We cannot handle multiple
                 * requests at a time.
                 */
                if (suspended != NONE) {
                    reply(TASK_REPLY, m.m_source, m.IO_ENDPT, EBUSY);
                    break;
                }
                suspended = m.IO_ENDPT;

```



```

    reply(TASK_REPLY, m.m_source, m.IO_ENDPT, SUSPEND);

    switch(m.REQUEST) {
    case CIOCGETTIME: /* get CMOS time */
    case CIOCGETTIMEY2K:
        y2kflag = (m.REQUEST == CIOCGETTIME) ? 0 : 1;
        result = gettime(m.IO_ENDPT, y2kflag, (vir_bytes) m.ADDRESS);
        break;
    case CIOCSETTIME:
    case CIOCSETTIMEY2K:
    default: /* unsupported ioctl */
        result = ENOSYS;
    }

    /* Request completed. Tell the caller to check our status. */
    notify(m.m_source);
    break;

case DEV_STATUS:

    /* The FS calls back to get our status. Revive the suspended
     * processes and return the status of reading the CMOS.
     */
    if (suspended == NONE)
        reply(DEV_NO_STATUS, m.m_source, NONE, OK);
    else
        reply(DEV_REVIVE, m.m_source, suspended, result);
    suspended = NONE;
    break;

case SYN_ALARM: /* shouldn't happen */
case SYS_SIG: /* ignore system events */
    continue;

default:
    reply(TASK_REPLY, m.m_source, m.IO_ENDPT, EINVAL);
}
}
}

/*=====
 *
 *                      reply
 *=====*/
PRIVATE void reply(int code, int replyee, int process, int status)
{
    message m;
    int s;

    m.m_type = code; /* TASK_REPLY or REVIVE */
    m.REP_STATUS = status; /* result of device operation */
    m.REP_ENDPT = process; /* which user made the request */
    if (OK != (s=send(replyee, &m)))
        panic("CMOS", "sending reply failed", s);
}

/*=====
 *
 *                      gettime
 *=====*/
PRIVATE int gettime(int who, int y2kflag, vir_bytes dst_time)
{
    unsigned char mach_id, cmos_state;
    struct tm timel;
    int i, s;

    /* First obtain the machine ID to see if we can read the CMOS clock. Only
     * for PS_386 and PC_AT this is possible. Otherwise, return an error.
     */
    sys_vircopy(SELF, BIOS_SEG, (vir_bytes) MACHINE_ID_ADDR,
        SELF, D, (vir_bytes) &mach_id, MACHINE_ID_SIZE);
    if (mach_id != PS_386_MACHINE && mach_id != PC_AT_MACHINE) {
        printf("IS: Machine ID unknown. ID byte = %02x.\n", mach_id);
        return(EFAULT);
    }
}

```

```

/* Now check the CMOS' state to see if we can read a proper time from it.
 * If the state is crappy, return an error.
 */
cmos_state = read_register(CMOS_STATUS);
if (cmos_state & (CS_LOST_POWER | CS_BAD_CHKSUM | CS_BAD_TIME)) {
    printf( "IS: CMOS RAM error(s) found. State= 0x%02x\n", cmos_state );
    if (cmos_state & CS_LOST_POWER)
        printf("IS: RTC lost power. Reset CMOS RAM with SETUP." );
    if (cmos_state & CS_BAD_CHKSUM)
        printf("IS: CMOS RAM checksum is bad. Run SETUP." );
    if (cmos_state & CS_BAD_TIME)
        printf("IS: Time invalid in CMOS RAM. Reset clock." );
    return(EFAULT);
}

/* Everything seems to be OK. Read the CMOS real time clock and copy the
 * result back to the caller.
 */
if (get_cmostime(&timel, y2kflag) != 0)
    return(EFAULT);
sys_datacopy(SELFS, (vir_bytes) &timel,
    who, dst_time, sizeof(struct tm));

return(OK);
}

PRIVATE int get_cmostime(struct tm *t, int y2kflag)
{
/* Update the structure pointed to by time with the current time as read
 * from CMOS RAM of the RTC. If necessary, the time is converted into a
 * binary format before being stored in the structure.
 */
int osec, n;
unsigned long i;
clock_t t0,t1;

/* Start a timer to keep us from getting stuck on a dead clock. */
getuptime(&t0);
do {
    osec = -1;
    n = 0;
    do {
        getuptime(&t1);
        if (t1-t0 > 5*HZ) {
            printf("readclock: CMOS clock appears dead\n");
            return(1);
        }

        /* Clock update in progress? */
        if (read_register(RTC_REG_A) & RTC_A_UIP) continue;

        t->tm_sec = read_register(RTC_SEC);
        if (t->tm_sec != osec) {
            /* Seconds changed. First from -1, then because the
             * clock ticked, which is what we're waiting for to
             * get a precise reading.
             */
            osec = t->tm_sec;
            n++;
        }
    } while (n < 2);

    /* Read the other registers. */
    t->tm_min = read_register(RTC_MIN);
    t->tm_hour = read_register(RTC_HOUR);
    t->tm_mday = read_register(RTC_MDAY);
    t->tm_mon = read_register(RTC_MONTH);
    t->tm_year = read_register(RTC_YEAR);

    /* Time stable? */
} while (read_register(RTC_SEC) != t->tm_sec
    || read_register(RTC_MIN) != t->tm_min
    || read_register(RTC_HOUR) != t->tm_hour
    || read_register(RTC_MDAY) != t->tm_mday

```

```
    read_register(RTC_MONTH) != t->tm_mon
    read_register(RTC_YEAR) != t->tm_year);

if ((read_register(RTC_REG_B) & RTC_B_DM_BCD) == 0) {
    /* Convert BCD to binary (default RTC mode). */
    t->tm_year = bcd_to_dec(t->tm_year);
    t->tm_mon = bcd_to_dec(t->tm_mon);
    t->tm_mday = bcd_to_dec(t->tm_mday);
    t->tm_hour = bcd_to_dec(t->tm_hour);
    t->tm_min = bcd_to_dec(t->tm_min);
    t->tm_sec = bcd_to_dec(t->tm_sec);
}
t->tm_mon--; /* Counts from 0. */

/* Correct the year, good until 2080. */
if (t->tm_year < 80) t->tm_year += 100;

if (y2kflag) {
    /* Clock with Y2K bug, interpret 1980 as 2000, good until 2020. */
    if (t->tm_year < 100) t->tm_year += 20;
}
return 0;
}

PRIVATE int read_register(int reg_addr)
{
    /* Read a single CMOS register value. */
    unsigned long r;
    sys_outb(RTC_INDEX, reg_addr);
    sys_inb(RTC_IO, &r);
    return r;
}

PRIVATE int bcd_to_dec(int n)
{
    return ((n >> 4) & 0x0F) * 10 + (n & 0x0F);
}

PRIVATE int dec_to_bcd(int n)
{
    return ((n / 10) << 4) | (n % 10);
}
```

```

/*
 *      3c503.c          A shared memory driver for Etherlink II board.
 *
 *      Created:         Dec. 20, 1996 by G. Falzoni <falzoni@marina.scn.de>
 *
 *      Inspired by the TNET package by M. Ostrowski, the driver for Linux
 *      by D. Becker, the Crynwr 3c503 packet driver, and the Amoeba driver.
 *
 *      It works in shared memory mode and should be used with the
 *      device driver for NS 8390 based cards of Minix. Programmed
 *      I/O could be used as well but would result in poor performance.
 */

#include "../drivers.h"

#include <net/gen/ether.h>
#include <net/gen/eth_io.h>

#include "local.h"
#include "dp8390.h"
#include "3c503.h"

#if ENABLE_3C503

#define MILLIS_TO_TICKS(m)  (((m)*HZ/1000)+1)

_PROTOTYPE(static void el2_init, (dpeth_t *dep));
_PROTOTYPE(static void el2_stop, (dpeth_t *dep));
_PROTOTYPE( static void milli_delay, (unsigned long millis)           );

/*=====
 *                               el2_init                               *
 *=====*/
static void el2_init(dep)
dpeth_t * dep;
{
    /* Initialize hardware and data structures. */
    int ix, irq;
    int sendq_nr;
    int cntr;

    /* Map the address PROM to lower I/O address range */
    cntr = inb_el2(dep, EL2_CNTR);
    outb_el2(dep, EL2_CNTR, cntr | ECNTR_SAPROM);

    /* Read station address from PROM */
    for (ix = EL2_EA0; ix <= EL2_EA5; ix += 1)
        dep->de_address.ea_addr[ix] = inb_el2(dep, ix);

    /* Map the 8390 back to lower I/O address range */
    outb_el2(dep, EL2_CNTR, cntr);

    /* Enable memory, but turn off interrupts until we are ready */
    outb_el2(dep, EL2_CFGR, ECFGR_IRQOFF);

    dep->de_data_port = dep->de_dp8390_port = dep->de_base_port;
    dep->de_prog_IO = 0;          /* Programmed I/O not yet available */

    /* Check width of data bus:
     * 1. Write 0 to WTS bit. The board will drive it to 1 if it is a
     *    16-bit card.
     * 2. Select page 2
     * 3. See if it is a 16-bit card
     * 4. Select page 0
     */
    outb_el2(dep, DP_CR, CR_PS_P0|CR_DM_ABORT|CR_STP);
    outb_el2(dep, DP_DCR, 0);
    outb_el2(dep, DP_CR, CR_PS_P2|CR_DM_ABORT|CR_STP);
    dep->de_16bit = (inb_el2(dep, DP_DCR) & DCR_WTS) != 0;
    outb_el2(dep, DP_CR, CR_PS_P0|CR_DM_ABORT|CR_STP);

    /* Allocate one send buffer (1.5KB) per 8KB of on board memory. */
    sendq_nr = (dep->de_ramsize - dep->de_offset_page) / 0x2000;
    if (sendq_nr < 1)

```

```

    sendq_nr = 1;
else if (sendq_nr > SENDQ_NR)
    sendq_nr = SENDQ_NR;

dep->de_sendq_nr = sendq_nr;
for (ix = 0; ix < sendq_nr; ix++)
    dep->de_sendq[ix].sq_sendpage = (ix * SENDQ_PAGES) + EL2_SM_START_PG;

dep->de_startpage = (ix * SENDQ_PAGES) + EL2_SM_START_PG;
dep->de_stoppage = EL2_SM_STOP_PG;

outb_el2(dep, EL2_STARTPG, dep->de_startpage);
outb_el2(dep, EL2_STOPPG, dep->de_stoppage);

/* Point the vector pointer registers somewhere ?harmless?. */
outb_el2(dep, EL2_VP2, 0xFF); /* Point at the ROM restart location */
outb_el2(dep, EL2_VP1, 0xFF); /* 0xFFFF:0000 (from original sources) */
outb_el2(dep, EL2_VP0, 0x00); /* - What for protected mode? */

/* Set interrupt level for 3c503 */
irq = (dep->de_irq &= ~DEI_DEFAULT); /* Strip the default flag. */
if (irq == 9) irq = 2;
if (irq < 2 || irq > 5) panic("", "bad 3c503 irq configuration", irq);
outb_el2(dep, EL2_IDCFG, (0x04 << irq));

outb_el2(dep, EL2_DRQCNT, 0x08); /* Set burst size to 8 */
outb_el2(dep, EL2_DMAAH, EL2_SM_START_PG); /* Put start of TX */
outb_el2(dep, EL2_DMAAL, 0x00); /* buffer in the GA DMA reg */

outb_el2(dep, EL2_CFGR, ECFGR_NORM); /* Enable shared memory */

if (!debug) {
    printf("%s: 3c503 at %X:%d:%lX\n",
        dep->de_name, dep->de_base_port, dep->de_irq,
        dep->de_linmem + dep->de_offset_page);
} else {
    printf("%s: 3Com Etherlink II %sat I/O address 0x%X, "
        "memory address 0x%lX, irq %d\n",
        dep->de_name, dep->de_16bit ? "(16-bit) " : "",
        dep->de_base_port,
        dep->de_linmem + dep->de_offset_page,
        dep->de_irq);
}
}

/*=====
*                               el2_stop                               *
*=====*/
static void el2_stop(dep)
dpeth_t * dep;
{
    /* Stops board by disabling interrupts. */

#ifdef DEBUG
    printf("%s: stopping Etherlink\n", dep->de_name);
#endif
    outb_el2(dep, EL2_CFGR, ECFGR_IRQOFF);
    return;
}

/*=====
*                               el2_probe                               *
*=====*/
int el2_probe(dep)
dpeth_t * dep;
{
    /* Probe for the presence of an EtherLink II card. Initialize memory
     * addressing if card detected.
     */
    int iobase, membase;
    int thin;

    /* Thin ethernet or AUI? */
    thin = (dep->de_linmem & 1) ? ECNTR_AUI : ECNTR_THIN;

```

```
/* Location registers should have 1 bit set */
if (!(iobase = inb_el2(dep, EL2_IOBASE))) return 0;
if (!(membase = inb_el2(dep, EL2_MEMBASE) & 0xF0)) return 0;
if ((iobase & (iobase - 1)) || (membase & (membase - 1))) return 0;

/* Resets board */
outb_el2(dep, EL2_CNTR, ECNTR_RESET | thin);
milli_delay(1);
outb_el2(dep, EL2_CNTR, thin);
milli_delay(5);

/* Map the address PROM to lower I/O address range */
outb_el2(dep, EL2_CNTR, ECNTR_SAPROM | thin);
if (inb_el2(dep, EL2_EA0) != 0x02 || /* Etherlink II Station address */
    inb_el2(dep, EL2_EA1) != 0x60 || /* MUST be 02:60:8c:xx:xx:xx */
    inb_el2(dep, EL2_EA2) != 0x8C)
    return 0; /* No Etherlink board at this address */

/* Map the 8390 back to lower I/O address range */
outb_el2(dep, EL2_CNTR, thin);

/* Setup shared memory addressing for 3c503 */
dep->de_linmem = ((membase & 0xC0) ? EL2_BASE_0D8000 : EL2_BASE_0C8000) +
    ((membase & 0xA0) ? (EL2_BASE_0CC000 - EL2_BASE_0C8000) : 0x0000);
dep->de_offset_page = (EL2_SM_START_PG * DP_PAGESIZE);
dep->de_ramsize = (EL2_SM_STOP_PG - EL2_SM_START_PG) * DP_PAGESIZE;

/* (Bad kludge, something Philip needs to look into. -- kjb) */
dep->de_linmem -= dep->de_offset_page;
dep->de_ramsize += dep->de_offset_page;

/* Board initialization and stop functions */
dep->de_initf = el2_init;
dep->de_stopf = el2_stop;
return 1;
}

static void milli_delay(unsigned long millis)
{
    tickdelay(MILLIS_TO_TICKS(millis));
}

#endif /* ENABLE_3C503 */

/** 3c503.c */

/*
 * $PchId: 3c503.c,v 1.3 2003/09/10 15:33:04 philip Exp $
 */
```

```
/*
 *      3c503.h          A shared memory driver for Etherlink II board.
 *
 *      Created:         Dec. 20, 1996 by G. Falzoni <falzoni@marina.scn.de>
 */

#define EL2_MEMTEST      0          /* Set to 1 for on board memory test */

#define EL2_GA            0x0400    /* Offset of registers in Gate Array */

/* EtherLink II card */

#define EL2_STARTPG      (EL2_GA+0x00) /* Start page matching DP_PSTARTPG */
#define EL2_STOPPG       (EL2_GA+0x01) /* Stop page matching DP_PSTOPPG */
#define EL2_DRQCNT       (EL2_GA+0x02) /* DMA burst count */
#define EL2_IOBASE       (EL2_GA+0x03) /* I/O base jumpers (bit coded) */
#define EL2_MEMBASE      (EL2_GA+0x04) /* Memory base jumpers (bit coded) */
#define EL2_CFGR         (EL2_GA+0x05) /* Configuration Register for GA */
#define EL2_CNTR         (EL2_GA+0x06) /* Control(write) and status(read) */
#define EL2_STATUS       (EL2_GA+0x07)
#define EL2_IDCFG        (EL2_GA+0x08) /* Interrupt/DMA configuration reg */
#define EL2_DMAAH        (EL2_GA+0x09) /* DMA address register (High byte) */
#define EL2_DMAAL        (EL2_GA+0x0A) /* DMA address register (Low byte) */
#define EL2_VP2          (EL2_GA+0x0B) /* Vector pointer - set to */
#define EL2_VP1          (EL2_GA+0x0C) /* reset address (0xFFFF:0) */
#define EL2_VP0          (EL2_GA+0x0D) /* */
#define EL2_FIFOH        (EL2_GA+0x0E) /* FIFO for progr. I/O (High byte) */
#define EL2_FIFOL        (EL2_GA+0x0F) /* FIFO for progr. I/O (Low byte) */

#define EL2_EA0          0x00        /* Most significant byte of ethernet address */
#define EL2_EA1          0x01
#define EL2_EA2          0x02
#define EL2_EA3          0x03
#define EL2_EA4          0x04
#define EL2_EA5          0x05        /* Least significant byte of ethernet address */

/* Bits in EL2_CNTR register */
#define ECNTR_RESET      0x01        /* Software Reset */
#define ECNTR_THIN       0x02        /* Onboard transceiver enable */
#define ECNTR_AUI        0x00        /* Onboard transceiver disable */
#define ECNTR_SAPROM     0x04        /* Map the station address prom */

/* Bits in EL2_CFGR register */
#define ECFGR_NORM       0x49        /* Enable 8k shared memory, no DMA, TC int */
#define ECFGR_IRQOFF     0xC9        /* As above, disable 8390 IRQ */

/* Shared memory management parameters */
#define EL2_SM_START_PG  0x20        /* First page of TX buffer */
#define EL2_SM_STOP_PG   0x40        /* Last page +1 of RX ring */

/* Physical addresses where an Etherlink board can be configured */
#define EL2_BASE_0C8000  0x0C8000
#define EL2_BASE_0CC000  0x0CC000
#define EL2_BASE_0D8000  0x0D8000
#define EL2_BASE_0DC000  0x0DC000

#define inb_el2(dep,reg)  (inb((dep)->de_base_port+(reg)))
#define outb_el2(dep,reg,data) (outb((dep)->de_base_port+(reg),(data)))

/** 3c503.h */

/*
 * $PchId: 3c503.h,v 1.3 2003/09/10 15:34:29 philip Exp $
 */
```

```
# Makefile for dp8390 driver
DRIVER = dp8390

# directories
u = /usr
i = $u/include
s = $i/sys
m = $i/minix
b = $i/ibm
d = ..

# programs, flags, etc.
CC =      exec cc
CFLAGS = -I$i
LDFLAGS = -i
LIBS = -lsys -lsysutil -ltimers

OBJ = 3c503.o dp8390.o ne2000.o rtl8029.o wdeth.o

# build local binary
all build:      $(DRIVER)
$(DRIVER):      $(OBJ)
                $(CC) -o $@ $(LDFLAGS) $(OBJ) $(LIBS)
                install -S 4096 $(DRIVER)

# install with other drivers
install:      /usr/sbin/$(DRIVER)
/usr/sbin/$(DRIVER):  $(DRIVER)
                install -o root -cs $? $@

# clean up local files
clean:
                rm -f *.o *.bak $(DRIVER)

depend:
                /usr/bin/mkdep "$(CC) -E $(CPPFLAGS)" *.c > .depend

# Include generated dependencies.
include .depend
```



```

/*
 * dp8390.c
 *
 * This file contains a ethernet device driver for NS dp8390 based ethernet
 * cards.
 *
 * The valid messages and their parameters are:
 *
 *      m_type      DL_PORT      DL_PROC      DL_COUNT      DL_MODE      DL_ADDR
 *      /-----+-----+-----+-----+-----+-----/
 *      / HARDINT   /             /             /             /             /
 *      /-----+-----+-----+-----+-----+-----/
 *      / DL_WRITE  / port nr    / proc nr    / count      / mode       / address
 *      /-----+-----+-----+-----+-----+-----/
 *      / DL_WRITEV / port nr    / proc nr    / count      / mode       / address
 *      /-----+-----+-----+-----+-----+-----/
 *      / DL_READ   / port nr    / proc nr    / count      /             / address
 *      /-----+-----+-----+-----+-----+-----/
 *      / DL_READV  / port nr    / proc nr    / count      /             / address
 *      /-----+-----+-----+-----+-----+-----/
 *      / DL_INIT   / port nr    / proc nr    / mode       /             / address
 *      /-----+-----+-----+-----+-----+-----/
 *      / DL_GETSTAT / port nr    / proc nr    /             /             / address
 *      /-----+-----+-----+-----+-----+-----/
 *      / DL_STOP   / port_nr    /             /             /             /
 *      /-----+-----+-----+-----+-----+-----/
 *
 * The messages sent are:
 *
 *      m-type      DL_PORT      DL_PROC      DL_COUNT      DL_STAT      DL_CLK
 *      /-----+-----+-----+-----+-----+-----/
 *      / DL_TASK_REPLY / port nr  / proc nr  / rd-count / err/stat / clock
 *      /-----+-----+-----+-----+-----+-----/
 *
 *      m_type      m3_i1      m3_i2      m3_ca1
 *      /-----+-----+-----+-----+-----/
 *      / DL_INIT_REPLY / port nr  / last port / ethernet addr /
 *      /-----+-----+-----+-----+-----/
 *
 * Created:      before Dec 28, 1992 by Philip Homburg <philip@f-mnx.phicoh.com>
 *
 * Modified Mar 10 1994 by Philip Homburg
 *      Become a generic dp8390 driver.
 *
 * Modified Dec 20 1996 by G. Falzoni <falzoni@marina.scn.de>
 *      Added support for 3c503 boards.
 */

#include "../drivers.h"

#include <stdlib.h>
#include <minix/com.h>
#include <net/hton.h>
#include <net/gen/ether.h>
#include <net/gen/eth_io.h>
#include "assert.h"

#include "local.h"
#include "dp8390.h"

#define DE_PORT_NR      3

static dpeth_t de_table[DE_PORT_NR];
static ul6_t eth_ign_proto;
static char *progname;

/* Configuration */
typedef struct dp_conf
{
    port_t dpc_port;
    int dpc_irq;
    phys_bytes dpc_mem;
    char *dpc_envvar;
} dp_conf_t;

```

```

dp_conf_t dp_conf[] =      /* Card addresses */
{
    /* I/O port, IRQ, Buffer address, Env. var. */
    { 0x280, 3, 0xD0000, "DPETH0" },
    { 0x300, 5, 0xC8000, "DPETH1" },
    { 0x380, 10, 0xD8000, "DPETH2" },
};

/* Test if dp_conf has exactly DE_PORT_NR entries. If not then you will see
 * the error: "array size is negative".
 */
extern int __dummy[DE_PORT_NR == sizeof(dp_conf)/sizeof(dp_conf[0]) ? 1 : -1];

/* Card inits configured out? */
#ifdef !ENABLE_WDETH
#define wdeth_probe(dep) (0)
#endif
#ifdef !ENABLE_NE2000
#define ne_probe(dep) (0)
#endif
#ifdef !ENABLE_3C503
#define el2_probe(dep) (0)
#endif

/* Some clones of the dp8390 and the PC emulator 'Bochs' require the CR_STA
 * on writes to the CR register. Additional CR_STAs do not appear to hurt
 * genuine dp8390s
 */
#define CR_EXTRA CR_STA

#ifdef ENABLE_PCI
_PROTOTYPE( static void pci_conf, (void) );
#endif
_PROTOTYPE( static void do_vwrite, (message *mp, int from_int,
                                     int vectored) );
_PROTOTYPE( static void do_vread, (message *mp, int vectored) );
_PROTOTYPE( static void do_init, (message *mp) );
_PROTOTYPE( static void do_int, (dpeth_t *dep) );
_PROTOTYPE( static void do_getstat, (message *mp) );
_PROTOTYPE( static void do_getname, (message *mp) );
_PROTOTYPE( static void do_stop, (message *mp) );
_PROTOTYPE( static void dp_init, (dpeth_t *dep) );
_PROTOTYPE( static void dp_confaddr, (dpeth_t *dep) );
_PROTOTYPE( static void dp_reinit, (dpeth_t *dep) );
_PROTOTYPE( static void dp_reset, (dpeth_t *dep) );
_PROTOTYPE( static void dp_check_ints, (dpeth_t *dep) );
_PROTOTYPE( static void dp_recv, (dpeth_t *dep) );
_PROTOTYPE( static void dp_send, (dpeth_t *dep) );
_PROTOTYPE( static void dp8390_stop, (void) );
_PROTOTYPE( static void dp_getblock, (dpeth_t *dep, int page,
                                     size_t offset, size_t size, void *dst) );
_PROTOTYPE( static void dp_pio8_getblock, (dpeth_t *dep, int page,
                                     size_t offset, size_t size, void *dst) );
_PROTOTYPE( static void dp_pio16_getblock, (dpeth_t *dep, int page,
                                     size_t offset, size_t size, void *dst) );
_PROTOTYPE( static int dp_pkt2user, (dpeth_t *dep, int page,
                                     int length) );
_PROTOTYPE( static void dp_user2nic, (dpeth_t *dep, iovec_dat_t *iovp,
                                     vir_bytes offset, int nic_addr, vir_bytes count) );
_PROTOTYPE( static void dp_pio8_user2nic, (dpeth_t *dep,
                                     iovec_dat_t *iovp, vir_bytes offset,
                                     int nic_addr, vir_bytes count) );
_PROTOTYPE( static void dp_pio16_user2nic, (dpeth_t *dep,
                                     iovec_dat_t *iovp, vir_bytes offset,
                                     int nic_addr, vir_bytes count) );
_PROTOTYPE( static void dp_nic2user, (dpeth_t *dep, int nic_addr,
                                     iovec_dat_t *iovp, vir_bytes offset, vir_bytes count) );
_PROTOTYPE( static void dp_pio8_nic2user, (dpeth_t *dep, int nic_addr,
                                     iovec_dat_t *iovp, vir_bytes offset, vir_bytes count) );
_PROTOTYPE( static void dp_pio16_nic2user, (dpeth_t *dep, int nic_addr,
                                     iovec_dat_t *iovp, vir_bytes offset, vir_bytes count) );
_PROTOTYPE( static void dp_next_iovec, (iovec_dat_t *iovp) );
_PROTOTYPE( static void conf_hw, (dpeth_t *dep) );

```

```

_PROTOTYPE( static void update_conf, (dpeth_t *dep, dp_conf_t *dcp) ) ;
_PROTOTYPE( static int calc_iovec_size, (iovec_dat_t *iovp) ) ;
_PROTOTYPE( static void reply, (dpeth_t *dep, int err, int may_block) ) ;
_PROTOTYPE( static void mess_reply, (message *req, message *reply) ) ;
_PROTOTYPE( static void get_userdata, (int user_proc,
    vir_bytes user_addr, vir_bytes count, void *loc_addr) ) ;
_PROTOTYPE( static void put_userdata, (int user_proc,
    vir_bytes user_addr, vir_bytes count, void *loc_addr) ) ;
_PROTOTYPE( static void insb, (port_t port, void *buf, size_t size)
);
_PROTOTYPE( static void insw, (port_t port, void *buf, size_t size)
);
_PROTOTYPE( static void do_vir_insb, (port_t port, int proc,
    vir_bytes buf, size_t size) ) ;
_PROTOTYPE( static void do_vir_insw, (port_t port, int proc,
    vir_bytes buf, size_t size) ) ;
_PROTOTYPE( static void do_vir_outsb, (port_t port, int proc,
    vir_bytes buf, size_t size) ) ;
_PROTOTYPE( static void do_vir_outsw, (port_t port, int proc,
    vir_bytes buf, size_t size) ) ;

/*=====
*                                     dpeth_task                                     *
*=====*/
int main(int argc, char *argv[])
{
    message m;
    int i, irq, r, tasknr;
    dpeth_t *dep;
    long v;

    if (argc < 1)
    {
        panic("DP8390",
            "A head which at this time has no name", NO_NUM);
    }
    (progname=strrchr(argv[0], '/')) ? progname++ : (progname=argv[0]);

    env_setargs(argc, argv);

    for (i= 0, dep= de_table; i<DE_PORT_NR; i++, dep++)
    {
        strcpy(dep->de_name, "dp8390#0");
        dep->de_name[7] += i;
    }

    v= 0;
    (void) env_parse("ETH_IGN_PROTO", "x", 0, &v, 0x0000L, 0xFFFFL);
    eth_ign_proto= htons((u16_t) v);

    /* Try to notify inet that we are present (again) */
    r = _pm_findproc("inet", &tasknr);
    if (r == OK)
        notify(tasknr);

    while (TRUE)
    {
        if ((r= receive(ANY, &m)) != OK)
            panic("", "dp8390: receive failed", r);

        switch (m.m_type)
        {
            case DEV_PING:  notify(m.m_source);                continue;
            case DL_WRITE:  do_vwrite(&m, FALSE, FALSE);      break;
            case DL_WRITEV: do_vwrite(&m, FALSE, TRUE);       break;
            case DL_READ:   do_vread(&m, FALSE);              break;
            case DL_READV:  do_vread(&m, TRUE);               break;
            case DL_INIT:   do_init(&m);                     break;
            case DL_GETSTAT: do_getstat(&m);                  break;
            case DL_GETNAME: do_getname(&m);                  break;
            case DL_STOP:   do_stop(&m);                      break;
            case HARD_INT:
                for (i= 0, dep= &de_table[0]; i<DE_PORT_NR; i++, dep++)
                {

```

```

        if (dep->de_mode != DEM_ENABLED)
            continue;
        assert(dep->de_flags & DEF_ENABLED);
        irq= dep->de_irq;
        assert(irq >= 0 && irq < NR_IRQ_VECTORS);
        if (dep->de_int_pending || 1)
        {
            dep->de_int_pending= 0;
            dp_check_ints(dep);
            do_int(dep);
            r= sys_irgenable(&dep->de_hook);
            if (r != OK)
            {
                panic("DP8390",
                    "unable enable interrupts", r);
            }
        }
    }
    break;
case SYS_SIG: {
    sigset_t sigset = m.NOTIFY_ARG;
    if (sigismember(&sigset, SIGKSTOP)) dp8390_stop();
    break;
}
case SYN_ALARM:
    printf("dp8390: strange, got SYN_ALARM\n");
    break;
case PROC_EVENT:
    break;
default:
    panic("", "dp8390: illegal message", m.m_type);
}
}

}

#if 0
/*=====
 *
 *                               dp8390_dump
 *=====*/
void dp8390_dump()
{
    dpeth_t *dep;
    int i, isr;

    printf("\n");
    for (i= 0, dep = &de_table[0]; i<DE_PORT_NR; i++, dep++)
    {
        #if XXX
            if (dep->de_mode == DEM_DISABLED)
                printf("dp8390 port %d is disabled\n", i);
            else if (dep->de_mode == DEM_SINK)
                printf("dp8390 port %d is in sink mode\n", i);
        #endif

        if (dep->de_mode != DEM_ENABLED)
            continue;

        printf("dp8390 statistics of port %d:\n", i);

        printf("recvErr  :%8ld\t", dep->de_stat.ets_recvErr);
        printf("sendErr   :%8ld\t", dep->de_stat.ets_sendErr);
        printf("OVW      :%8ld\n", dep->de_stat.ets_OVW);

        printf("CRCerr    :%8ld\t", dep->de_stat.ets_CRCerr);
        printf("frameAll  :%8ld\t", dep->de_stat.ets_frameAll);
        printf("missedP   :%8ld\n", dep->de_stat.ets_missedP);

        printf("packetR   :%8ld\t", dep->de_stat.ets_packetR);
        printf("packetT   :%8ld\t", dep->de_stat.ets_packetT);
        printf("transDef  :%8ld\n", dep->de_stat.ets_transDef);

        printf("collision :%8ld\t", dep->de_stat.ets_collision);
        printf("transAb   :%8ld\t", dep->de_stat.ets_transAb);
        printf("carrSense :%8ld\n", dep->de_stat.ets_carrSense);

```

```

        printf("fifoUnder :%8ld\t", dep->de_stat.ets_fifoUnder);
        printf("fifoOver  :%8ld\t", dep->de_stat.ets_fifoOver);
        printf("CDheartbeat:%8ld\n", dep->de_stat.ets_CDheartbeat);

        printf("OWC      :%8ld\t", dep->de_stat.ets_OWC);

        isr= inb_reg0(dep, DP_ISR);
        printf("dp_isr= 0x%x + 0x%x, de_flags= 0x%x\n", isr,
                inb_reg0(dep, DP_ISR), dep->de_flags);
    }
}
#endif

/*=====
 *                                dp8390_stop                                *
 *=====*/
static void dp8390_stop()
{
    message mess;
    int i;

    for (i= 0; i<DE_PORT_NR; i++)
    {
        if (de_table[i].de_mode != DEM_ENABLED)
            continue;
        mess.m_type= DL_STOP;
        mess.DL_PORT= i;
        do_stop(&mess);
    }
}

#if ENABLE_PCI
/*=====
 *                                pci_conf                                *
 *=====*/
static void pci_conf()
{
    int i, h;
    char *envvar;
    struct dpeth *dep;
    static char envfmt[] = ":%d.d.d";
    long v;
    static int first_time= 1;

    if (!first_time)
        return;
    first_time= 0;

    for (i= 0, dep= de_table; i<DE_PORT_NR; i++, dep++)
    {
        envvar= dp_conf[i].dpc_envvar;
        if (!(dep->de_pci= env_prefix(envvar, "pci")))
            continue; /* no PCI config */
        v= 0;
        (void) env_parse(envvar, envfmt, 1, &v, 0, 255);
        dep->de_pcibus= v;
        v= 0;
        (void) env_parse(envvar, envfmt, 2, &v, 0, 255);
        dep->de_pcidev= v;
        v= 0;
        (void) env_parse(envvar, envfmt, 3, &v, 0, 255);
        dep->de_pcifunc= v;
    }

    for (h= 1; h >= 0; h--) {
        for (i= 0, dep= de_table; i<DE_PORT_NR; i++, dep++)
        {
            if (!dep->de_pci)
                continue;
            if (((dep->de_pcibus | dep->de_pcidev |
                dep->de_pcifunc) != 0) != h)
            {
                continue;
            }

```

```

        }
        if (!rtl_probe(dep))
            dep->de_pci = -1;
    }
}
#endif /* ENABLE_PCI */

/*=====
 *                               do_vwrite
 *=====*/
static void do_vwrite(mp, from_int, vectored)
message *mp;
int from_int;
int vectored;
{
    int port, count, size;
    int sendq_head;
    dpeth_t *dep;

    port = mp->DL_PORT;
    count = mp->DL_COUNT;
    if (port < 0 || port >= DE_PORT_NR)
        panic("", "dp8390: illegal port", port);
    dep = &de_table[port];
    dep->de_client = mp->DL_PROC;

    if (dep->de_mode == DEM_SINK)
    {
        assert(!from_int);
        dep->de_flags |= DEF_PACK_SEND;
        reply(dep, OK, FALSE);
        return;
    }
    assert(dep->de_mode == DEM_ENABLED);
    assert(dep->de_flags & DEF_ENABLED);
    if (dep->de_flags & DEF_SEND_AVAIL)
        panic("", "dp8390: send already in progress", NO_NUM);

    sendq_head = dep->de_sendq_head;
    if (dep->de_sendq[sendq_head].sq_filled)
    {
        if (from_int)
            panic("", "dp8390: should not be sending\n", NO_NUM);
        dep->de_sendmsg = *mp;
        dep->de_flags |= DEF_SEND_AVAIL;
        reply(dep, OK, FALSE);
        return;
    }
    assert(!(dep->de_flags & DEF_PACK_SEND));

    if (vectored)
    {
        get_userdata(mp->DL_PROC, (vir_bytes) mp->DL_ADDR,
            (count > IOVEC_NR ? IOVEC_NR : count) *
            sizeof(iovec_t), dep->de_write_iovec.iod_iovec);
        dep->de_write_iovec.iod_iovec_s = count;
        dep->de_write_iovec.iod_proc_nr = mp->DL_PROC;
        dep->de_write_iovec.iod_iovec_addr = (vir_bytes) mp->DL_ADDR;

        dep->de_tmp_iovec = dep->de_write_iovec;
        size = calc_iovec_size(&dep->de_tmp_iovec);
    }
    else
    {
        dep->de_write_iovec.iod_iovec[0].iov_addr =
            (vir_bytes) mp->DL_ADDR;
        dep->de_write_iovec.iod_iovec[0].iov_size =
            mp->DL_COUNT;
        dep->de_write_iovec.iod_iovec_s = 1;
        dep->de_write_iovec.iod_proc_nr = mp->DL_PROC;
        dep->de_write_iovec.iod_iovec_addr = 0;
        size = mp->DL_COUNT;
    }
}

```

```

    if (size < ETH_MIN_PACK_SIZE || size > ETH_MAX_PACK_SIZE_TAGGED)
    {
        panic("", "dp8390: invalid packet size", size);
    }
    (dep->de_user2nicf)(dep, &dep->de_write_iovec, 0,
        dep->de_sendq[sendq_head].sq_sendpage * DP_PAGESIZE,
        size);
    dep->de_sendq[sendq_head].sq_filled= TRUE;
    if (dep->de_sendq_tail == sendq_head)
    {
        outb_reg0(dep, DP_TPSR, dep->de_sendq[sendq_head].sq_sendpage);
        outb_reg0(dep, DP_TBCR1, size >> 8);
        outb_reg0(dep, DP_TBCR0, size & 0xff);
        outb_reg0(dep, DP_CR, CR_TXP | CR_EXTRA);/* there it goes.. */
    }
    else
        dep->de_sendq[sendq_head].sq_size= size;

    if (++sendq_head == dep->de_sendq_nr)
        sendq_head= 0;
    assert(sendq_head < SENDQ_NR);
    dep->de_sendq_head= sendq_head;

    dep->de_flags |= DEF_PACK_SEND;

    /* If the interrupt handler called, don't send a reply. The reply
       * will be sent after all interrupts are handled.
       */
    if (from_int)
        return;
    reply(dep, OK, FALSE);

    assert(dep->de_mode == DEM_ENABLED);
    assert(dep->de_flags & DEF_ENABLED);
}

/*=====
 *                               do_vread                               *
 *=====*/
static void do_vread(mp, vectored)
message *mp;
int vectored;
{
    int port, count;
    int size;
    dpeth_t *dep;

    port = mp->DL_PORT;
    count = mp->DL_COUNT;
    if (port < 0 || port >= DE_PORT_NR)
        panic("", "dp8390: illegal port", port);
    dep= &de_table[port];
    dep->de_client= mp->DL_PROC;
    if (dep->de_mode == DEM_SINK)
    {
        reply(dep, OK, FALSE);
        return;
    }
    assert(dep->de_mode == DEM_ENABLED);
    assert(dep->de_flags & DEF_ENABLED);

    if(dep->de_flags & DEF_READING)
        panic("", "dp8390: read already in progress", NO_NUM);

    if (vectored)
    {
        get_userdata(mp->DL_PROC, (vir_bytes) mp->DL_ADDR,
            (count > IOVEC_NR ? IOVEC_NR : count) *
            sizeof(iovec_t), dep->de_read_iovec.iod_iovec);
        dep->de_read_iovec.iod_iovec_s = count;
        dep->de_read_iovec.iod_proc_nr = mp->DL_PROC;
        dep->de_read_iovec.iod_iovec_addr = (vir_bytes) mp->DL_ADDR;

        dep->de_tmp_iovec = dep->de_read_iovec;
    }
}

```

```

        size= calc_iovec_size(&dep->de_tmp_iovec);
    }
    else
    {
        dep->de_read_iovec.iod_iovec[0].iov_addr =
            (vir_bytes) mp->DL_ADDR;
        dep->de_read_iovec.iod_iovec[0].iov_size =
            mp->DL_COUNT;
        dep->de_read_iovec.iod_iovec_s = 1;
        dep->de_read_iovec.iod_proc_nr = mp->DL_PROC;
        dep->de_read_iovec.iod_iovec_addr = 0;
        size= count;
    }
    if (size < ETH_MAX_PACKET_SIZE_TAGGED)
        panic(" ", "dp8390: wrong packet size", size);
    dep->de_flags |= DEF_READING;

    dp_recv(dep);

    if ((dep->de_flags & (DEF_READING|DEF_STOPPED)) ==
        (DEF_READING|DEF_STOPPED))
    {
        /* The chip is stopped, and all arrived packets are
         * delivered.
         */
        dp_reset(dep);
    }
    reply(dep, OK, FALSE);
}

/*=====
 *                               do_init                               *
 *=====*/
static void do_init(mp)
message *mp;
{
    int port;
    dpeth_t *dep;
    message reply_mess;

#ifdef ENABLE_PCI
    pci_conf(); /* Configure PCI devices. */
#endif

    port = mp->DL_PORT;
    if (port < 0 || port >= DE_PORT_NR)
    {
        reply_mess.m_type= DL_INIT_REPLY;
        reply_mess.m3_il= ENXIO;
        mess_reply(mp, &reply_mess);
        return;
    }
    dep= &de_table[port];
    if (dep->de_mode == DEM_DISABLED)
    {
        /* This is the default, try to (re)locate the device. */
        conf_hw(dep);
        if (dep->de_mode == DEM_DISABLED)
        {
            /* Probe failed, or the device is configured off. */
            reply_mess.m_type= DL_INIT_REPLY;
            reply_mess.m3_il= ENXIO;
            mess_reply(mp, &reply_mess);
            return;
        }
        if (dep->de_mode == DEM_ENABLED)
            dp_init(dep);
    }

    if (dep->de_mode == DEM_SINK)
    {
        strncpy((char *) dep->de_address.ea_addr, "ZDP", 6);
        dep->de_address.ea_addr[5] = port;
        dp_confaddr(dep);
    }

```



```

        reply_mess.m_type = DL_INIT_REPLY;
        reply_mess.m3_i1 = mp->DL_PORT;
        reply_mess.m3_i2 = DE_PORT_NR;
        *(ether_addr_t *) reply_mess.m3_cal = dep->de_address;
        mess_reply(mp, &reply_mess);
        return;
    }
    assert(dep->de_mode == DEM_ENABLED);
    assert(dep->de_flags & DEF_ENABLED);

    dep->de_flags &= ~(DEF_PROMISC | DEF_MULTI | DEF_BROAD);

    if (mp->DL_MODE & DL_PROMISC_REQ)
        dep->de_flags |= DEF_PROMISC | DEF_MULTI | DEF_BROAD;
    if (mp->DL_MODE & DL_MULTI_REQ)
        dep->de_flags |= DEF_MULTI;
    if (mp->DL_MODE & DL_BROAD_REQ)
        dep->de_flags |= DEF_BROAD;

    dep->de_client = mp->m_source;
    dp_reinit(dep);

    reply_mess.m_type = DL_INIT_REPLY;
    reply_mess.m3_i1 = mp->DL_PORT;
    reply_mess.m3_i2 = DE_PORT_NR;
    *(ether_addr_t *) reply_mess.m3_cal = dep->de_address;

    mess_reply(mp, &reply_mess);
}

/*=====
 *
 * do_int
 *=====*/
static void do_int(dep)
dpeth_t *dep;
{
    if (dep->de_flags & (DEF_PACK_SEND | DEF_PACK_RECV))
        reply(dep, OK, TRUE);
}

/*=====
 *
 * do_getstat
 *=====*/
static void do_getstat(mp)
message *mp;
{
    int port;
    dpeth_t *dep;

    port = mp->DL_PORT;
    if (port < 0 || port >= DE_PORT_NR)
        panic("", "dp8390: illegal port", port);
    dep = &de_table[port];
    dep->de_client = mp->DL_PROC;
    if (dep->de_mode == DEM_SINK)
    {
        put_userdata(mp->DL_PROC, (vir_bytes) mp->DL_ADDR,
                     (vir_bytes) sizeof(dep->de_stat), &dep->de_stat);
        reply(dep, OK, FALSE);
        return;
    }
    assert(dep->de_mode == DEM_ENABLED);
    assert(dep->de_flags & DEF_ENABLED);

    dep->de_stat.ets_CRCerr += inb_reg0(dep, DP_CNTR0);
    dep->de_stat.ets_frameAll += inb_reg0(dep, DP_CNTR1);
    dep->de_stat.ets_missedP += inb_reg0(dep, DP_CNTR2);

    put_userdata(mp->DL_PROC, (vir_bytes) mp->DL_ADDR,
                 (vir_bytes) sizeof(dep->de_stat), &dep->de_stat);
    reply(dep, OK, FALSE);
}

/*=====

```

```

*                               do_getname                               *
*=====*/
static void do_getname(mp)
message *mp;
{
    int r;

    strncpy(mp->DL_NAME, progname, sizeof(mp->DL_NAME));
    mp->DL_NAME[sizeof(mp->DL_NAME)-1] = '\0';
    mp->m_type= DL_NAME_REPLY;
    r= send(mp->m_source, mp);
    if (r != OK)
        panic("dp8390", "do_getname: send failed: %d\n", r);
}

/*=====*/
*                               do_stop                               *
*=====*/
static void do_stop(mp)
message *mp;
{
    int port;
    dpeth_t *dep;

    port = mp->DL_PORT;

    if (port < 0 || port >= DE_PORT_NR)
        panic("", "dp8390: illegal port", port);
    dep= &de_table[port];
    if (dep->de_mode == DEM_SINK)
        return;
    assert(dep->de_mode == DEM_ENABLED);

    if (!(dep->de_flags & DEF_ENABLED))
        return;

    outb_reg0(dep, DP_CR, CR_STP | CR_DM_ABORT);
    (dep->de_stopf)(dep);

    dep->de_flags= DEF_EMPTY;
}

/*=====*/
*                               dp_init                               *
*=====*/
static void dp_init(dep)
dpeth_t *dep;
{
    int dp_rcr_reg;
    int i, r;

    /* General initialization */
    dep->de_flags = DEF_EMPTY;
    (*dep->de_initf)(dep);

    dp_confaddr(dep);

    if (debug)
    {
        printf("%s: Ethernet address ", dep->de_name);
        for (i= 0; i < 6; i++)
            printf("%x%c", dep->de_address.ea_addr[i],
                i < 5 ? ':' : '\n');
    }

    /* Initialization of the dp8390 following the mandatory procedure
     * in reference manual ("DP8390D/NS32490D NIC Network Interface
     * Controller", National Semiconductor, July 1995, Page 29).
     */
    /* Step 1: */
    outb_reg0(dep, DP_CR, CR_PS_P0 | CR_STP | CR_DM_ABORT);
    /* Step 2: */
    if (dep->de_l6bit)
        outb_reg0(dep, DP_DCR, DCR_WORDWIDE | DCR_8BYTES | DCR_BMS);
}

```

```

else
    outb_reg0(dep, DP_DCR, DCR_BYTEWIDE | DCR_8BYTES | DCR_BMS);
/* Step 3: */
outb_reg0(dep, DP_RBCR0, 0);
outb_reg0(dep, DP_RBCR1, 0);
/* Step 4: */
dp_rcr_reg = 0;
if (dep->de_flags & DEF_PROMISC)
    dp_rcr_reg |= RCR_AB | RCR_PRO | RCR_AM;
if (dep->de_flags & DEF_BROAD)
    dp_rcr_reg |= RCR_AB;
if (dep->de_flags & DEF_MULTI)
    dp_rcr_reg |= RCR_AM;
outb_reg0(dep, DP_RCR, dp_rcr_reg);
/* Step 5: */
outb_reg0(dep, DP_TCR, TCR_INTERNAL);
/* Step 6: */
outb_reg0(dep, DP_BNRY, dep->de_startpage);
outb_reg0(dep, DP_PSTART, dep->de_startpage);
outb_reg0(dep, DP_PSTOP, dep->de_stoppage);
/* Step 7: */
outb_reg0(dep, DP_ISR, 0xFF);
/* Step 8: */
outb_reg0(dep, DP_IMR, IMR_PRXE | IMR_PTXE | IMR_RXEE | IMR_TXEE |
    IMR_OVWE | IMR_CNTE);
/* Step 9: */
outb_reg0(dep, DP_CR, CR_PS_P1 | CR_DM_ABORT | CR_STP);

outb_reg1(dep, DP_PAR0, dep->de_address.ea_addr[0]);
outb_reg1(dep, DP_PAR1, dep->de_address.ea_addr[1]);
outb_reg1(dep, DP_PAR2, dep->de_address.ea_addr[2]);
outb_reg1(dep, DP_PAR3, dep->de_address.ea_addr[3]);
outb_reg1(dep, DP_PAR4, dep->de_address.ea_addr[4]);
outb_reg1(dep, DP_PAR5, dep->de_address.ea_addr[5]);

outb_reg1(dep, DP_MAR0, 0xff);
outb_reg1(dep, DP_MAR1, 0xff);
outb_reg1(dep, DP_MAR2, 0xff);
outb_reg1(dep, DP_MAR3, 0xff);
outb_reg1(dep, DP_MAR4, 0xff);
outb_reg1(dep, DP_MAR5, 0xff);
outb_reg1(dep, DP_MAR6, 0xff);
outb_reg1(dep, DP_MAR7, 0xff);

outb_reg1(dep, DP_CURR, dep->de_startpage + 1);
/* Step 10: */
outb_reg0(dep, DP_CR, CR_DM_ABORT | CR_STA);
/* Step 11: */
outb_reg0(dep, DP_TCR, TCR_NORMAL);

inb_reg0(dep, DP_CNTR0);
inb_reg0(dep, DP_CNTR1);
inb_reg0(dep, DP_CNTR2);

/* Finish the initialization. */
dep->de_flags |= DEF_ENABLED;
for (i = 0; i < dep->de_sendq_nr; i++)
    dep->de_sendq[i].sq_filled = 0;
dep->de_sendq_head = 0;
dep->de_sendq_tail = 0;
if (!dep->de_prog_IO)
{
    dep->de_user2nicf = dp_user2nic;
    dep->de_nic2userf = dp_nic2user;
    dep->de_getblockf = dp_getblock;
}
else if (dep->de_16bit)
{
    dep->de_user2nicf = dp_pi16_user2nic;
    dep->de_nic2userf = dp_pi16_nic2user;
    dep->de_getblockf = dp_pi16_getblock;
}
else
{

```

```

        dep->de_user2nicf= dp_pio8_user2nic;
        dep->de_nic2userf= dp_pio8_nic2user;
        dep->de_getblockf= dp_pio8_getblock;
    }

    /* Set the interrupt handler and policy. Do not automatically
     * reenale interrupts. Return the IRQ line number on interrupts.
     */
    dep->de_hook = dep->de_irq;
    r= sys_irqsetpolicy(dep->de_irq, 0, &dep->de_hook);
    if (r != OK)
        panic("DP8390", "sys_irqsetpolicy failed", r);

    r= sys_irgenable(&dep->de_hook);
    if (r != OK)
    {
        panic("DP8390", "unable enable interrupts", r);
    }
}

/*=====
 *
 *                      dp_confaddr
 *=====*/
static void dp_confaddr(dep)
dpeth_t *dep;
{
    int i;
    char eakey[16];
    static char eafmt[] = "x:x:x:x:x";
    long v;

    /* User defined ethernet address? */
    strcpy(eakey, dp_conf[dep-de_table].dpc_envvar);
    strcat(eakey, "_EA");

    for (i= 0; i < 6; i++)
    {
        v= dep->de_address.ea_addr[i];
        if (env_parse(eakey, eafmt, i, &v, 0x00L, 0xFFL) != EP_SET)
        {
            break;
        }
        dep->de_address.ea_addr[i]= v;
    }

    if (i != 0 && i != 6) env_panic(eakey); /* It's all or nothing */
}

/*=====
 *
 *                      dp_reinit
 *=====*/
static void dp_reinit(dep)
dpeth_t *dep;
{
    int dp_rcr_reg;

    outb_reg0(dep, DP_CR, CR_PS_P0 | CR_EXTRA);

    dp_rcr_reg = 0;
    if (dep->de_flags & DEF_PROMISC)
        dp_rcr_reg |= RCR_AB | RCR_PRO | RCR_AM;
    if (dep->de_flags & DEF_BROAD)
        dp_rcr_reg |= RCR_AB;
    if (dep->de_flags & DEF_MULTI)
        dp_rcr_reg |= RCR_AM;
    outb_reg0(dep, DP_RCR, dp_rcr_reg);
}

/*=====
 *
 *                      dp_reset
 *=====*/
static void dp_reset(dep)
dpeth_t *dep;
{

```

```

    int i;

    /* Stop chip */
    outb_reg0(dep, DP_CR, CR_STP | CR_DM_ABORT);
    outb_reg0(dep, DP_RBCR0, 0);
    outb_reg0(dep, DP_RBCR1, 0);
    for (i= 0; i < 0x1000 && ((inb_reg0(dep, DP_ISR) & ISR_RST) == 0); i++)
        ; /* Do nothing */
    outb_reg0(dep, DP_TCR, TCR_1EXTERNAL|TCR_OFST);
    outb_reg0(dep, DP_CR, CR_STA|CR_DM_ABORT);
    outb_reg0(dep, DP_TCR, TCR_NORMAL);

    /* Acknowledge the ISR_RDC (remote dma) interrupt. */
    for (i= 0; i < 0x1000 && ((inb_reg0(dep, DP_ISR) & ISR_RDC) == 0); i++)
        ; /* Do nothing */
    outb_reg0(dep, DP_ISR, inb_reg0(dep, DP_ISR) & ~ISR_RDC);

    /* Reset the transmit ring. If we were transmitting a packet, we
     * pretend that the packet is processed. Higher layers will
     * retransmit if the packet wasn't actually sent.
     */
    dep->de_sendq_head= dep->de_sendq_tail= 0;
    for (i= 0; i<dep->de_sendq_nr; i++)
        dep->de_sendq[i].sq_filled= 0;
    dp_send(dep);
    dep->de_flags &= ~DEF_STOPPED;
}

/*=====
 *                               dp_check_ints                               *
 *=====*/
static void dp_check_ints(dep)
dpeth_t *dep;
{
    int isr, tsr;
    int size, sendq_tail;

    if (!(dep->de_flags & DEF_ENABLED))
        panic("", "dp8390: got premature interrupt", NO_NUM);

    for(;;)
    {
        isr = inb_reg0(dep, DP_ISR);
        if (!isr)
            break;
        outb_reg0(dep, DP_ISR, isr);
        if (isr & (ISR_PTX|ISR_TXE))
        {
            if (isr & ISR_TXE)
            {
                #if DEBUG
                { printf("%s: got send Error\n", dep->de_name); }
                #endif
                dep->de_stat.ets_sendErr++;
            }
            else
            {
                tsr = inb_reg0(dep, DP_TSR);

                if (tsr & TSR_PTX) dep->de_stat.ets_packetT++;
                #if 0 /* Reserved in later manuals, should be ignored */
                if (!(tsr & TSR_DFR))
                {
                    /* In most (all?) implementations of
                     * the dp8390, this bit is set
                     * when the packet is not deferred
                     */
                    dep->de_stat.ets_transDef++;
                }
            }
            #endif
            if (tsr & TSR_COL) dep->de_stat.ets_collision++;
            if (tsr & TSR_ABT) dep->de_stat.ets_transAb++;
            if (tsr & TSR_CRS) dep->de_stat.ets_carrSense++;
            if (tsr & TSR_FU

```

```

        && ++dep->de_stat.ets_fifoUnder <= 10)
    {
        printf("%s: fifo underrun\n",
            dep->de_name);
    }
    if (tsr & TSR_CDH
        && ++dep->de_stat.ets_CDheartbeat <= 10)
    {
        printf("%s: CD heart beat failure\n",
            dep->de_name);
    }
    if (tsr & TSR_OWC) dep->de_stat.ets_OWC++;
}
sendq_tail= dep->de_sendq_tail;

if (!(dep->de_sendq[sendq_tail].sq_filled))
{
    /* Software bug? */
    assert(!debug);

    /* Or hardware bug? */
    printf(
        "%s: transmit interrupt, but not sending\n",
            dep->de_name);
    continue;
}
dep->de_sendq[sendq_tail].sq_filled= 0;
if (++sendq_tail == dep->de_sendq_nr)
    sendq_tail= 0;
dep->de_sendq_tail= sendq_tail;
if (dep->de_sendq[sendq_tail].sq_filled)
{
    size= dep->de_sendq[sendq_tail].sq_size;
    outb_reg0(dep, DP_TPSR,
        dep->de_sendq[sendq_tail].sq_sendpage);
    outb_reg0(dep, DP_TBCR1, size >> 8);
    outb_reg0(dep, DP_TBCR0, size & 0xff);
    outb_reg0(dep, DP_CR, CR_TXP | CR_EXTRA);
}
if (dep->de_flags & DEF_SEND_AVAIL)
    dp_send(dep);
}

if (isr & ISR_PRX)
{
    /* Only call dp_rcv if there is a read request */
    if (dep->de_flags & DEF_READING)
        dp_rcv(dep);
}

if (isr & ISR_RXE) dep->de_stat.ets_rcvErr++;
if (isr & ISR_CNT)
{
    dep->de_stat.ets_CRCerr += inb_reg0(dep, DP_CNTR0);
    dep->de_stat.ets_frameAll += inb_reg0(dep, DP_CNTR1);
    dep->de_stat.ets_missedP += inb_reg0(dep, DP_CNTR2);
}
if (isr & ISR_OVW)
{
    dep->de_stat.ets_OVW++;
}
#if 0
    { printf(); printf(
        "%s: got overwrite warning\n", dep->de_name); }
#endif

if (dep->de_flags & DEF_READING)
{
    printf(
        "dp_check_ints: strange: overwrite warning and pending read request\n");
    dp_rcv(dep);
}
}
if (isr & ISR_RDC)
{
    /* Nothing to do */

```

```

    }
    if (isr & ISR_RST)
    {
        /* this means we got an interrupt but the ethernet
        * chip is shutdown. We set the flag DEF_STOPPED,
        * and continue processing arrived packets. When the
        * receive buffer is empty, we reset the dp8390.
        */
        #if 0
            { printf(); printf(
                "%s: NIC stopped\n", dep->de_name); }
        #endif
        dep->de_flags |= DEF_STOPPED;
        break;
    }
    if ((dep->de_flags & (DEF_READING|DEF_STOPPED)) ==
        (DEF_READING|DEF_STOPPED))
    {
        /* The chip is stopped, and all arrived packets are
        * delivered.
        */
        dp_reset(dep);
    }
}

/*=====
*                                     dp_rcv
*=====*/
static void dp_rcv(dep)
dpeth_t *dep;
{
    dp_rcvhdr_t header;
    unsigned pageno, curr, next;
    vir_bytes length;
    int packet_processed, r;
    ul6_t eth_type;

    packet_processed = FALSE;
    pageno = inb_reg0(dep, DP_BNRY) + 1;
    if (pageno == dep->de_stoppage) pageno = dep->de_startpage;

    do
    {
        outb_reg0(dep, DP_CR, CR_PS_P1 | CR_EXTRA);
        curr = inb_reg1(dep, DP_CURR);
        outb_reg0(dep, DP_CR, CR_PS_P0 | CR_EXTRA);

        if (curr == pageno) break;

        (dep->de_getblockf)(dep, pageno, (size_t)0, sizeof(header),
            &header);
        (dep->de_getblockf)(dep, pageno, sizeof(header) +
            2*sizeof(ether_addr_t), sizeof(eth_type), &eth_type);

        length = (header.dr_rbc1 | (header.dr_rbc2 << 8)) -
            sizeof(dp_rcvhdr_t);
        next = header.dr_next;
        if (length < ETH_MIN_PACKET_SIZE ||
            length > ETH_MAX_PACKET_SIZE_TAGGED)
        {
            printf("%s: packet with strange length arrived: %d\n",
                dep->de_name, (int) length);
            next = curr;
        }
        else if (next < dep->de_startpage || next >= dep->de_stoppage)
        {
            printf("%s: strange next page\n", dep->de_name);
            next = curr;
        }
        else if (eth_type == eth_ign_proto)
        {
            /* Hack: ignore packets of a given protocol, useful
            * if you share a net with 80 computers sending

```

```

        * Amoeba FLIP broadcasts. (Protocol 0x8146.)
        */
        static int first= 1;
        if (first)
        {
            first= 0;
            printf( "%s: dropping proto 0x%04x packets\n",
                    dep->de_name,
                    ntohs(eth_ign_proto));
        }
        dep->de_stat.ets_packetR++;
    }
    else if (header.dr_status & RSR_FO)
    {
        /* This is very serious, so we issue a warning and
         * reset the buffers */
        printf( "%s: fifo overrun, resetting receive buffer\n",
                dep->de_name);
        dep->de_stat.ets_fifoOver++;
        next = curr;
    }
    else if ((header.dr_status & RSR_PRX) &&
             (dep->de_flags & DEF_ENABLED))
    {
        r = dp_pkt2user(dep, pageno, length);
        if (r != OK)
            return;

        packet_processed = TRUE;
        dep->de_stat.ets_packetR++;
    }
    if (next == dep->de_startpage)
        outb_reg0(dep, DP_BNRY, dep->de_stoppage - 1);
    else
        outb_reg0(dep, DP_BNRY, next - 1);

    pageno = next;
}
while (!packet_processed);
}

/*=====
 *                                dp_send                                *
 *=====*/
static void dp_send(dep)
dpeth_t *dep;
{
    if (!(dep->de_flags & DEF_SEND_AVAIL))
        return;

    dep->de_flags &= ~DEF_SEND_AVAIL;
    switch(dep->de_sendmsg.m_type)
    {
        case DL_WRITE:  do_vwrite(&dep->de_sendmsg, TRUE, FALSE);      break;
        case DL_WRITEV: do_vwrite(&dep->de_sendmsg, TRUE, TRUE);       break;
        default:
            panic(" ", "dp8390: wrong type:", dep->de_sendmsg.m_type);
            break;
    }
}

/*=====
 *                                dp_getblock                            *
 *=====*/
static void dp_getblock(dep, page, offset, size, dst)
dpeth_t *dep;
int page;
size_t offset;
size_t size;
void *dst;
{
    int r;

    offset = page * DP_PAGESIZE + offset;

```



```

    r = sys_vircopy(SELF, BIOS_SEG, dep->de_linmem + offset,
        SELF, D, (vir_bytes)dst, size);

    if (r != OK)
        panic("DP8390", "dp_getblock: sys_vircopy failed", r);
}

/*=====
 *                               dp_pio8_getblock                               *
 *=====*/
static void dp_pio8_getblock(dep, page, offset, size, dst)
dpeth_t *dep;
int page;
size_t offset;
size_t size;
void *dst;
{
    offset = page * DP_PAGESIZE + offset;
    outb_reg0(dep, DP_RBCR0, size & 0xFF);
    outb_reg0(dep, DP_RBCR1, size >> 8);
    outb_reg0(dep, DP_RSAR0, offset & 0xFF);
    outb_reg0(dep, DP_RSAR1, offset >> 8);
    outb_reg0(dep, DP_CR, CR_DM_RR | CR_PS_P0 | CR_STA);

    insb(dep->de_data_port, dst, size);
}

/*=====
 *                               dp_pio16_getblock                              *
 *=====*/
static void dp_pio16_getblock(dep, page, offset, size, dst)
dpeth_t *dep;
int page;
size_t offset;
size_t size;
void *dst;
{
    offset = page * DP_PAGESIZE + offset;
    outb_reg0(dep, DP_RBCR0, size & 0xFF);
    outb_reg0(dep, DP_RBCR1, size >> 8);
    outb_reg0(dep, DP_RSAR0, offset & 0xFF);
    outb_reg0(dep, DP_RSAR1, offset >> 8);
    outb_reg0(dep, DP_CR, CR_DM_RR | CR_PS_P0 | CR_STA);

    assert(!(size & 1));
    insw(dep->de_data_port, dst, size);
}

/*=====
 *                               dp_pkt2user                               *
 *=====*/
static int dp_pkt2user(dep, page, length)
dpeth_t *dep;
int page, length;
{
    int last, count;

    if (!(dep->de_flags & DEF_READING))
        return EGENERIC;

    last = page + (length - 1) / DP_PAGESIZE;
    if (last >= dep->de_stoppage)
    {
        count = (dep->de_stoppage - page) * DP_PAGESIZE -
            sizeof(dp_rcvhdr_t);

        /* Save read_iovec since we need it twice. */
        dep->de_tmp_iovec = dep->de_read_iovec;
        (dep->de_nic2userf)(dep, page * DP_PAGESIZE +
            sizeof(dp_rcvhdr_t), &dep->de_tmp_iovec, 0, count);
        (dep->de_nic2userf)(dep, dep->de_startpage * DP_PAGESIZE,
            &dep->de_read_iovec, count, length - count);
    }
}

```

```

    else
    {
        (dep->de_nic2userf)(dep, page * DP_PAGESIZE +
                           sizeof(dp_rcvhdr_t), &dep->de_read_iovec, 0, length);
    }

    dep->de_read_s = length;
    dep->de_flags |= DEF_PACK_RECV;
    dep->de_flags &= ~DEF_READING;

    return OK;
}

/*=====
 *                               dp_user2nic                               *
 *=====*/
static void dp_user2nic(dep, iovep, offset, nic_addr, count)
dpeth_t *dep;
iovec_dat_t *iovp;
vir_bytes offset;
int nic_addr;
vir_bytes count;
{
    vir_bytes vir_hw, vir_user;
    int bytes, i, r;

    vir_hw = dep->de_linmem + nic_addr;

    i = 0;
    while (count > 0)
    {
        if (i >= IOVEC_NR)
        {
            dp_next_iovec(iovp);
            i = 0;
            continue;
        }
        assert(i < iovep->iod_iovec_s);
        if (offset >= iovep->iod_iovec[i].iov_size)
        {
            offset -= iovep->iod_iovec[i].iov_size;
            i++;
            continue;
        }
        bytes = iovep->iod_iovec[i].iov_size - offset;
        if (bytes > count)
            bytes = count;

        r = sys_vircopy(iovp->iod_proc_nr, D,
                      iovep->iod_iovec[i].iov_addr + offset,
                      SELF, BIOS_SEG, vir_hw, bytes);
        if (r != OK)
            panic("DP8390", "dp_user2nic: sys_vircopy failed", r);

        count -= bytes;
        vir_hw += bytes;
        offset += bytes;
    }
    assert(count == 0);
}

/*=====
 *                               dp_pio8_user2nic                               *
 *=====*/
static void dp_pio8_user2nic(dep, iovep, offset, nic_addr, count)
dpeth_t *dep;
iovec_dat_t *iovp;
vir_bytes offset;
int nic_addr;
vir_bytes count;
{
    phys_bytes phys_user;
    int bytes, i;

```

```

    outb_reg0(dep, DP_ISR, ISR_RDC);

    outb_reg0(dep, DP_RBCR0, count & 0xFF);
    outb_reg0(dep, DP_RBCR1, count >> 8);
    outb_reg0(dep, DP_RSAR0, nic_addr & 0xFF);
    outb_reg0(dep, DP_RSAR1, nic_addr >> 8);
    outb_reg0(dep, DP_CR, CR_DM_RW | CR_PS_P0 | CR_STA);

    i= 0;
    while (count > 0)
    {
        if (i >= IOVEC_NR)
        {
            dp_next_iovec(iovp);
            i= 0;
            continue;
        }
        assert(i < iovp->iod_iovec_s);
        if (offset >= iovp->iod_iovec[i].iov_size)
        {
            offset -= iovp->iod_iovec[i].iov_size;
            i++;
            continue;
        }
        bytes = iovp->iod_iovec[i].iov_size - offset;
        if (bytes > count)
            bytes = count;

        do_vir_outsb(dep->de_data_port, iovp->iod_proc_nr,
                    iovp->iod_iovec[i].iov_addr + offset, bytes);
        count -= bytes;
        offset += bytes;
    }
    assert(count == 0);

    for (i= 0; i<100; i++)
    {
        if (inb_reg0(dep, DP_ISR) & ISR_RDC)
            break;
    }
    if (i == 100)
    {
        panic("", "dp8390: remote dma failed to complete", NO_NUM);
    }
}

/*=====
 *                               dp_piol6_user2nic                               *
 *=====*/
static void dp_piol6_user2nic(dep, iovp, offset, nic_addr, count)
dpeth_t *dep;
iovec_dat_t *iovp;
vir_bytes offset;
int nic_addr;
vir_bytes count;
{
    vir_bytes vir_user;
    vir_bytes ecount;
    int i, r, bytes, user_proc;
    u8_t two_bytes[2];
    int odd_byte;

    ecount= (count+1) & ~1;
    odd_byte= 0;

    outb_reg0(dep, DP_ISR, ISR_RDC);
    outb_reg0(dep, DP_RBCR0, ecount & 0xFF);
    outb_reg0(dep, DP_RBCR1, ecount >> 8);
    outb_reg0(dep, DP_RSAR0, nic_addr & 0xFF);
    outb_reg0(dep, DP_RSAR1, nic_addr >> 8);
    outb_reg0(dep, DP_CR, CR_DM_RW | CR_PS_P0 | CR_STA);

    i= 0;
    while (count > 0)

```

```
{
    if (i >= IOVEC_NR)
    {
        dp_next_iovec(iovp);
        i= 0;
        continue;
    }
    assert(i < iovp->iod_iovec_s);
    if (offset >= iovp->iod_iovec[i].iov_size)
    {
        offset -= iovp->iod_iovec[i].iov_size;
        i++;
        continue;
    }
    bytes = iovp->iod_iovec[i].iov_size - offset;
    if (bytes > count)
        bytes = count;

    user_proc= iovp->iod_proc_nr;
    vir_user= iovp->iod_iovec[i].iov_addr + offset;
    if (odd_byte)
    {
        r= sys_vircopy(user_proc, D, vir_user,
            SELF, D, (vir_bytes)&two_bytes[1], 1);
        if (r != OK)
        {
            panic("DP8390",
                "dp_pio16_user2nic: sys_vircopy failed",
                r);
        }
        outw(dep->de_data_port, *(u16_t *)two_bytes);
        count--;
        offset++;
        bytes--;
        vir_user++;
        odd_byte= 0;
        if (!bytes)
            continue;
    }
    ecount= bytes & ~1;
    if (ecount != 0)
    {
        do_vir_outsw(dep->de_data_port, user_proc, vir_user,
            ecount);
        count -= ecount;
        offset += ecount;
        bytes -= ecount;
        vir_user += ecount;
    }
    if (bytes)
    {
        assert(bytes == 1);
        r= sys_vircopy(user_proc, D, vir_user,
            SELF, D, (vir_bytes)&two_bytes[0], 1);
        if (r != OK)
        {
            panic("DP8390",
                "dp_pio16_user2nic: sys_vircopy failed",
                r);
        }
        count--;
        offset++;
        bytes--;
        vir_user++;
        odd_byte= 1;
    }
}
assert(count == 0);

if (odd_byte)
    outw(dep->de_data_port, *(u16_t *)two_bytes);

for (i= 0; i<100; i++)
{
```

```

        if (inb_reg0(dep, DP_ISR) & ISR_RDC)
            break;
    }
    if (i == 100)
    {
        panic("", "dp8390: remote dma failed to complete", NO_NUM);
    }
}

/*=====
 *                                dp_nic2user                                *
 *=====*/
static void dp_nic2user(dep, nic_addr, iovp, offset, count)
dpeth_t *dep;
int nic_addr;
iovec_dat_t *iovp;
vir_bytes offset;
vir_bytes count;
{
    vir_bytes vir_hw, vir_user;
    int bytes, i, r;

    vir_hw = dep->de_linmem + nic_addr;

    i = 0;
    while (count > 0)
    {
        if (i >= IOVEC_NR)
        {
            dp_next_iovec(iovp);
            i = 0;
            continue;
        }
        assert(i < iovp->iod_iovec_s);
        if (offset >= iovp->iod_iovec[i].iov_size)
        {
            offset -= iovp->iod_iovec[i].iov_size;
            i++;
            continue;
        }
        bytes = iovp->iod_iovec[i].iov_size - offset;
        if (bytes > count)
            bytes = count;

        r = sys_vircopy(SELF, BIOS_SEG, vir_hw,
            iovp->iod_proc_nr, D,
            iovp->iod_iovec[i].iov_addr + offset, bytes);
        if (r != OK)
            panic("DP8390", "dp_nic2user: sys_vircopy failed", r);

        count -= bytes;
        vir_hw += bytes;
        offset += bytes;
    }
    assert(count == 0);
}

/*=====
 *                                dp_pio8_nic2user                                *
 *=====*/
static void dp_pio8_nic2user(dep, nic_addr, iovp, offset, count)
dpeth_t *dep;
int nic_addr;
iovec_dat_t *iovp;
vir_bytes offset;
vir_bytes count;
{
    phys_bytes phys_user;
    int bytes, i;

    outb_reg0(dep, DP_RBCR0, count & 0xFF);
    outb_reg0(dep, DP_RBCR1, count >> 8);
    outb_reg0(dep, DP_RSAR0, nic_addr & 0xFF);
    outb_reg0(dep, DP_RSAR1, nic_addr >> 8);

```

```

    outb_reg0(dep, DP_CR, CR_DM_RR | CR_PS_P0 | CR_STA);

    i= 0;
    while (count > 0)
    {
        if (i >= IOVEC_NR)
        {
            dp_next_iovec(iovp);
            i= 0;
            continue;
        }
        assert(i < iovp->iod_iovec_s);
        if (offset >= iovp->iod_iovec[i].iov_size)
        {
            offset -= iovp->iod_iovec[i].iov_size;
            i++;
            continue;
        }
        bytes = iovp->iod_iovec[i].iov_size - offset;
        if (bytes > count)
            bytes = count;

        do_vir_insb(dep->de_data_port, iovp->iod_proc_nr,
                    iovp->iod_iovec[i].iov_addr + offset, bytes);
        count -= bytes;
        offset += bytes;
    }
    assert(count == 0);
}

/*=====
 *                                dp_piol6_nic2user                                *
 *=====*/
static void dp_piol6_nic2user(dep, nic_addr, iovp, offset, count)
dpeth_t *dep;
int nic_addr;
iovec_dat_t *iovp;
vir_bytes offset;
vir_bytes count;
{
    vir_bytes vir_user;
    vir_bytes ecount;
    int i, r, bytes, user_proc;
    u8_t two_bytes[2];
    int odd_byte;

    ecount= (count+1) & ~1;
    odd_byte= 0;

    outb_reg0(dep, DP_RBCR0, ecount & 0xFF);
    outb_reg0(dep, DP_RBCR1, ecount >> 8);
    outb_reg0(dep, DP_RSAR0, nic_addr & 0xFF);
    outb_reg0(dep, DP_RSAR1, nic_addr >> 8);
    outb_reg0(dep, DP_CR, CR_DM_RR | CR_PS_P0 | CR_STA);

    i= 0;
    while (count > 0)
    {
        if (i >= IOVEC_NR)
        {
            dp_next_iovec(iovp);
            i= 0;
            continue;
        }
        assert(i < iovp->iod_iovec_s);
        if (offset >= iovp->iod_iovec[i].iov_size)
        {
            offset -= iovp->iod_iovec[i].iov_size;
            i++;
            continue;
        }
        bytes = iovp->iod_iovec[i].iov_size - offset;
        if (bytes > count)
            bytes = count;

```

```

    user_proc= iovp->iod_proc_nr;
    vir_user= iovp->iod_iovec[i].iov_addr + offset;
    if (odd_byte)
    {
        r= sys_vircopy(SELFS, D, (vir_bytes)&two_bytes[1],
            user_proc, D, vir_user, 1);
        if (r != OK)
        {
            panic("DP8390",
                "dp_piol6_nic2user: sys_vircopy failed",
                r);
        }
        count--;
        offset++;
        bytes--;
        vir_user++;
        odd_byte= 0;
        if (!bytes)
            continue;
    }
    ecoun= bytes & ~1;
    if (ecoun != 0)
    {
        do_vir_insw(dep->de_data_port, user_proc, vir_user,
            ecoun);
        count -= ecoun;
        offset += ecoun;
        bytes -= ecoun;
        vir_user += ecoun;
    }
    if (bytes)
    {
        assert(bytes == 1);
        *(u16_t *)two_bytes= inw(dep->de_data_port);
        r= sys_vircopy(SELFS, D, (vir_bytes)&two_bytes[0],
            user_proc, D, vir_user, 1);
        if (r != OK)
        {
            panic("DP8390",
                "dp_piol6_nic2user: sys_vircopy failed",
                r);
        }
        count--;
        offset++;
        bytes--;
        vir_user++;
        odd_byte= 1;
    }
}
assert(count == 0);
}

/*=====
 *
 *                                     dp_next_iovec
 *=====*/
static void dp_next_iovec(iovp)
iovec_dat_t *iovp;
{
    assert(iovp->iod_iovec_s > IOVEC_NR);

    iovp->iod_iovec_s -= IOVEC_NR;

    iovp->iod_iovec_addr += IOVEC_NR * sizeof(iovec_t);

    get_userdata(iovp->iod_proc_nr, iovp->iod_iovec_addr,
        (iovp->iod_iovec_s > IOVEC_NR ? IOVEC_NR : iovp->iod_iovec_s) *
        sizeof(iovec_t), iovp->iod_iovec);
}

/*=====
 *
 *                                     conf_hw
 *=====*/
static void conf_hw(dep)

```

```

dpeth_t *dep;
{
    static eth_stat_t empty_stat = {0, 0, 0, 0, 0, 0 /* ,... */ };

    int ifnr;
    dp_conf_t *dcp;

    dep->de_mode= DEM_DISABLED; /* Superfluous */
    ifnr= dep->de_table;

    dcp= &dp_conf[ifnr];
    update_conf(dep, dcp);
    if (dep->de_mode != DEM_ENABLED)
        return;
    if (!wdeth_probe(dep) && !ne_probe(dep) && !el2_probe(dep))
    {
        printf("%s: No ethernet card found at 0x%x\n",
            dep->de_name, dep->de_base_port);
        dep->de_mode= DEM_DISABLED;
        return;
    }

/* XXX */ if (dep->de_linmem == 0) dep->de_linmem= 0xFFFFF0000;

    dep->de_flags = DEF_EMPTY;
    dep->de_stat = empty_stat;
}

/*=====
 *                               update_conf                               *
 *=====*/
static void update_conf(dep, dcp)
dpeth_t *dep;
dp_conf_t *dcp;
{
    long v;
    static char dpc_fmt[] = "x:d:x:x";

#if ENABLE_PCI
    if (dep->de_pci)
    {
        if (dep->de_pci == 1)
        {
            /* PCI device is present */
            dep->de_mode= DEM_ENABLED;
        }
        return; /* Already configured */
    }
#endif

    /* Get the default settings and modify them from the environment. */
    dep->de_mode= DEM_SINK;
    v= dcp->dpc_port;
    switch (env_parse(dcp->dpc_envvar, dpc_fmt, 0, &v, 0x0000L, 0xFFFFL)) {
    case EP_OFF:
        dep->de_mode= DEM_DISABLED;
        break;
    case EP_ON:
    case EP_SET:
        dep->de_mode= DEM_ENABLED; /* Might become disabled if
                                   * all probes fail */
        break;
    }
    dep->de_base_port= v;

    v= dcp->dpc_irq | DEI_DEFAULT;
    (void) env_parse(dcp->dpc_envvar, dpc_fmt, 1, &v, 0L,
        (long) NR_IRQ_VECTORS - 1);
    dep->de_irq= v;

    v= dcp->dpc_mem;
    (void) env_parse(dcp->dpc_envvar, dpc_fmt, 2, &v, 0L, 0xFFFFFL);
    dep->de_linmem= v;

```



```

    v= 0;
    (void) env_parse(dcp->dpc_envvar, dpc_fmt, 3, &v, 0x2000L, 0x8000L);
    dep->de_ramsize= v;
}

/*=====
 *                               calc_iovec_size                               *
 *=====*/
static int calc_iovec_size(iovp)
iovec_dat_t *iovp;
{
    /* Calculate the size of a request. Note that the iovec_dat
     * structure will be unusable after calc_iovec_size.
     */
    int size;
    int i;

    size= 0;
    i= 0;
    while (i < iovp->iod_iovec_s)
    {
        if (i >= IOVEC_NR)
        {
            dp_next_iovec(iovp);
            i= 0;
            continue;
        }
        size += iovp->iod_iovec[i].iov_size;
        i++;
    }
    return size;
}

/*=====
 *                               reply                               *
 *=====*/
static void reply(dep, err, may_block)
dpeth_t *dep;
int err;
int may_block;
{
    message reply;
    int status;
    int r;

    status = 0;
    if (dep->de_flags & DEF_PACK_SEND)
        status |= DL_PACK_SEND;
    if (dep->de_flags & DEF_PACK_RECV)
        status |= DL_PACK_RECV;

    reply.m_type = DL_TASK_REPLY;
    reply.DL_PORT = dep - de_table;
    reply.DL_PROC = dep->de_client;
    reply.DL_STAT = status | ((u32_t) err << 16);
    reply.DL_COUNT = dep->de_read_s;
    reply.DL_CLCK = 0; /* Don't know */
    r= send(dep->de_client, &reply);

    if (r == ELOCKED && may_block)
    {
#if 0
        printf("send locked\n");
#endif
        return;
    }

    if (r < 0)
        panic("", "dp8390: send failed:", r);

    dep->de_read_s = 0;
    dep->de_flags &= ~(DEF_PACK_SEND | DEF_PACK_RECV);
}

```

```

/*=====
 *                                mess_reply                                *
 *=====*/
static void mess_reply(req, reply_mess)
message *req;
message *reply_mess;
{
    if (send(req->m_source, reply_mess) != OK)
        panic("", "dp8390: unable to mess_reply", NO_NUM);
}

/*=====
 *                                get_userdata                                *
 *=====*/
static void get_userdata(user_proc, user_addr, count, loc_addr)
int user_proc;
vir_bytes user_addr;
vir_bytes count;
void *loc_addr;
{
    int r;

    r= sys_vircopy(user_proc, D, user_addr,
        SELF, D, (vir_bytes)loc_addr, count);
    if (r != OK)
        panic("DP8390", "get_userdata: sys_vircopy failed", r);
}

/*=====
 *                                put_userdata                                *
 *=====*/
static void put_userdata(user_proc, user_addr, count, loc_addr)
int user_proc;
vir_bytes user_addr;
vir_bytes count;
void *loc_addr;
{
    int r;

    r= sys_vircopy(SELF, D, (vir_bytes)loc_addr,
        user_proc, D, user_addr, count);
    if (r != OK)
        panic("DP8390", "put_userdata: sys_vircopy failed", r);
}

u8_t inb(port_t port)
{
    int r;
    u32_t value;

    r= sys_inb(port, &value);
    if (r != OK)
        panic("DP8390", "sys_inb failed", r);
    return value;
}

u16_t inw(port_t port)
{
    int r;
    unsigned long value;

    r= sys_inw(port, &value);
    if (r != OK)
        panic("DP8390", "sys_inw failed", r);
    return (u16_t) value;
}

void outb(port_t port, u8_t value)
{
    int r;

    r= sys_outb(port, value);
    if (r != OK)
        panic("DP8390", "sys_outb failed", r);
}

```

```
}

void outw(port_t port, ul6_t value)
{
    int r;

    r= sys_outw(port, value);
    if (r != OK)
        panic("DP8390", "sys_outw failed", r);
}

static void insb(port_t port, void *buf, size_t size)
{
    do_vir_insb(port, SELF, (vir_bytes)buf, size);
}

static void insw(port_t port, void *buf, size_t size)
{
    do_vir_insw(port, SELF, (vir_bytes)buf, size);
}

static void do_vir_insb(port_t port, int proc, vir_bytes buf, size_t size)
{
    int r;

    r= sys_sdevio(DIO_INPUT, port, DIO_BYTE, proc, (void *)buf, size);
    if (r != OK)
        panic("DP8390", "sys_sdevio failed", r);
}

static void do_vir_insw(port_t port, int proc, vir_bytes buf, size_t size)
{
    int r;

    r= sys_sdevio(DIO_INPUT, port, DIO_WORD, proc, (void *)buf, size);
    if (r != OK)
        panic("DP8390", "sys_sdevio failed", r);
}

static void do_vir_outsb(port_t port, int proc, vir_bytes buf, size_t size)
{
    int r;

    r= sys_sdevio(DIO_OUTPUT, port, DIO_BYTE, proc, (void *)buf, size);
    if (r != OK)
        panic("DP8390", "sys_sdevio failed", r);
}

static void do_vir_outsw(port_t port, int proc, vir_bytes buf, size_t size)
{
    int r;

    r= sys_sdevio(DIO_OUTPUT, port, DIO_WORD, proc, (void *)buf, size);
    if (r != OK)
        panic("DP8390", "sys_sdevio failed", r);
}

/*
 * $PchId: dp8390.c,v 1.25 2005/02/10 17:32:07 philip Exp $
 */
```

```
/*
dp8390.h
```

```
Created:          before Dec 28, 1992 by Philip Homburg
*/
```

```
/* National Semiconductor DP8390 Network Interface Controller. */
```

```
/* Page 0, for reading ----- */
#define DP_CR          0x0    /* Read side of Command Register */
#define DP_CLDA0       0x1    /* Current Local Dma Address 0 */
#define DP_CLDA1       0x2    /* Current Local Dma Address 1 */
#define DP_BNRY        0x3    /* Boundary Pointer */
#define DP_TSR          0x4    /* Transmit Status Register */
#define DP_NCR          0x5    /* Number of Collisions Register */
#define DP_FIFO        0x6    /* Fifo ?? */
#define DP_ISR         0x7    /* Interrupt Status Register */
#define DP_CRDA0       0x8    /* Current Remote Dma Address 0 */
#define DP_CRDA1       0x9    /* Current Remote Dma Address 1 */
#define DP_DUM1        0xA    /* unused */
#define DP_DUM2        0xB    /* unused */
#define DP_RSR         0xC    /* Receive Status Register */
#define DP_CNTR0       0xD    /* Tally Counter 0 */
#define DP_CNTR1       0xE    /* Tally Counter 1 */
#define DP_CNTR2       0xF    /* Tally Counter 2 */
```

```
/* Page 0, for writing ----- */
#define DP_CR          0x0    /* Write side of Command Register */
#define DP_PSTART      0x1    /* Page Start Register */
#define DP_PSTOP       0x2    /* Page Stop Register */
#define DP_BNRY        0x3    /* Boundary Pointer */
#define DP_TPSR        0x4    /* Transmit Page Start Register */
#define DP_TBCR0       0x5    /* Transmit Byte Count Register 0 */
#define DP_TBCR1       0x6    /* Transmit Byte Count Register 1 */
#define DP_ISR         0x7    /* Interrupt Status Register */
#define DP_RSAR0       0x8    /* Remote Start Address Register 0 */
#define DP_RSAR1       0x9    /* Remote Start Address Register 1 */
#define DP_RBCR0       0xA    /* Remote Byte Count Register 0 */
#define DP_RBCR1       0xB    /* Remote Byte Count Register 1 */
#define DP_RCR         0xC    /* Receive Configuration Register */
#define DP_TCR         0xD    /* Transmit Configuration Register */
#define DP_DCR         0xE    /* Data Configuration Register */
#define DP_IMR         0xF    /* Interrupt Mask Register */
```

```
/* Page 1, read/write ----- */
#define DP_CR          0x0    /* Command Register */
#define DP_PAR0        0x1    /* Physical Address Register 0 */
#define DP_PAR1        0x2    /* Physical Address Register 1 */
#define DP_PAR2        0x3    /* Physical Address Register 2 */
#define DP_PAR3        0x4    /* Physical Address Register 3 */
#define DP_PAR4        0x5    /* Physical Address Register 4 */
#define DP_PAR5        0x6    /* Physical Address Register 5 */
#define DP_CURR        0x7    /* Current Page Register */
#define DP_MAR0        0x8    /* Multicast Address Register 0 */
#define DP_MAR1        0x9    /* Multicast Address Register 1 */
#define DP_MAR2        0xA    /* Multicast Address Register 2 */
#define DP_MAR3        0xB    /* Multicast Address Register 3 */
#define DP_MAR4        0xC    /* Multicast Address Register 4 */
#define DP_MAR5        0xD    /* Multicast Address Register 5 */
#define DP_MAR6        0xE    /* Multicast Address Register 6 */
#define DP_MAR7        0xF    /* Multicast Address Register 7 */
```

```
/* Bits in dp_cr */
#define CR_STP          0x01    /* Stop: software reset */
#define CR_STA          0x02    /* Start: activate NIC */
#define CR_TXP          0x04    /* Transmit Packet */
#define CR_DMA          0x38    /* Mask for DMA control */
#define CR_DM_NOP       0x00    /* DMA: No Operation */
#define CR_DM_RR        0x08    /* DMA: Remote Read */
#define CR_DM_RW        0x10    /* DMA: Remote Write */
#define CR_DM_SP        0x18    /* DMA: Send Packet */
#define CR_DM_ABORT     0x20    /* DMA: Abort Remote DMA Operation */
#define CR_PS           0xC0    /* Mask for Page Select */
#define CR_PS_P0        0x00    /* Register Page 0 */
```

```

#define CR_PS_P1      0x40    /* Register Page 1 */
#define CR_PS_P2      0x80    /* Register Page 2 */
#define CR_PS_T1      0xC0    /* Test Mode Register Map */

/* Bits in dp_isr */
#define ISR_PRX        0x01    /* Packet Received with no errors */
#define ISR_PTX        0x02    /* Packet Transmitted with no errors */
#define ISR_RXE        0x04    /* Receive Error */
#define ISR_TXE        0x08    /* Transmit Error */
#define ISR_OVW        0x10    /* Overwrite Warning */
#define ISR_CNT        0x20    /* Counter Overflow */
#define ISR_RDC        0x40    /* Remote DMA Complete */
#define ISR_RST        0x80    /* Reset Status */

/* Bits in dp_imr */
#define IMR_PRXE       0x01    /* Packet Received iEnable */
#define IMR_PTXE       0x02    /* Packet Transmitted iEnable */
#define IMR_RXEE       0x04    /* Receive Error iEnable */
#define IMR_TXEE       0x08    /* Transmit Error iEnable */
#define IMR_OVWE       0x10    /* Overwrite Warning iEnable */
#define IMR_CNTE       0x20    /* Counter Overflow iEnable */
#define IMR_RDCE       0x40    /* DMA Complete iEnable */

/* Bits in dp_dcr */
#define DCR_WTS        0x01    /* Word Transfer Select */
#define DCR_BYTEWIDE   0x00    /* WTS: byte wide transfers */
#define DCR_WORDWIDE   0x01    /* WTS: word wide transfers */
#define DCR_BOS        0x02    /* Byte Order Select */
#define DCR_LITTLENDIAN 0x00    /* BOS: Little Endian */
#define DCR_BIGENDIAN  0x02    /* BOS: Big Endian */
#define DCR_LAS        0x04    /* Long Address Select */
#define DCR_BMS        0x08    /* Burst Mode Select
                                * Called Loopback Select (LS) in
                                * later manuals. Should be set. */
#define DCR_AR         0x10    /* Autoinitialize Remote */
#define DCR_FTS        0x60    /* Fifo Threshold Select */
#define DCR_2BYTES     0x00    /* 2 bytes */
#define DCR_4BYTES     0x40    /* 4 bytes */
#define DCR_8BYTES     0x20    /* 8 bytes */
#define DCR_12BYTES    0x60    /* 12 bytes */

/* Bits in dp_tcr */
#define TCR_CRC        0x01    /* Inhibit CRC */
#define TCR_ELC        0x06    /* Encoded Loopback Control */
#define TCR_NORMAL     0x00    /* ELC: Normal Operation */
#define TCR_INTERNAL   0x02    /* ELC: Internal Loopback */
#define TCR_0EXTERNAL  0x04    /* ELC: External Loopback LPBK=0 */
#define TCR_1EXTERNAL  0x06    /* ELC: External Loopback LPBK=1 */
#define TCR_ATD        0x08    /* Auto Transmit Disable */
#define TCR_OFST       0x10    /* Collision Offset Enable (be nice) */

/* Bits in dp_tsr */
#define TSR_PTX        0x01    /* Packet Transmitted (without error) */
#define TSR_DFR        0x02    /* Transmit Deferred, reserved in
                                * later manuals. */
#define TSR_COL        0x04    /* Transmit Collided */
#define TSR_ABT        0x08    /* Transmit Aborted */
#define TSR_CRS        0x10    /* Carrier Sense Lost */
#define TSR_FU         0x20    /* FIFO Underrun */
#define TSR_CDH        0x40    /* CD Heartbeat */
#define TSR_OWC        0x80    /* Out of Window Collision */

/* Bits in tp_rcr */
#define RCR_SEP        0x01    /* Save Errored Packets */
#define RCR_AR         0x02    /* Accept Runt Packets */
#define RCR_AB         0x04    /* Accept Broadcast */
#define RCR_AM         0x08    /* Accept Multicast */
#define RCR_PRO        0x10    /* Physical Promiscuous */
#define RCR_MON        0x20    /* Monitor Mode */

/* Bits in dp_rsr */
#define RSR_PRX        0x01    /* Packet Received Intact */
#define RSR_CRC        0x02    /* CRC Error */
#define RSR_FAE        0x04    /* Frame Alignment Error

```

```

#define RSR_FO      0x08      /* FIFO Overrun */
#define RSR_MPA     0x10      /* Missed Packet */
#define RSR_PHY     0x20      /* Multicast Address Match */
#define RSR_DIS     0x40      /* Receiver Disabled */
#define RSR_DFR     0x80      /* In later manuals: Deferring */

typedef struct dp_rcvhdr
{
    u8_t dr_status;          /* Copy of rsr */
    u8_t dr_next;            /* Pointer to next packet */
    u8_t dr_rbcl;            /* Receive Byte Count Low */
    u8_t dr_rבח;            /* Receive Byte Count High */
} dp_rcvhdr_t;

#define DP_PAGESIZE      256

/* Some macros to simplify accessing the dp8390 */
#define inb_reg0(dep, reg)      (inb(dep->de_dp8390_port+reg))
#define outb_reg0(dep, reg, data) (outb(dep->de_dp8390_port+reg, data))
#define inb_reg1(dep, reg)      (inb(dep->de_dp8390_port+reg))
#define outb_reg1(dep, reg, data) (outb(dep->de_dp8390_port+reg, data))

/* Software interface to the dp8390 driver */

struct dpeth;
struct iovec_dat;
_PROTOTYPE( typedef void (*dp_initf_t), (struct dpeth *dep) );
_PROTOTYPE( typedef void (*dp_stopf_t), (struct dpeth *dep) );
_PROTOTYPE( typedef void (*dp_user2nicf_t), (struct dpeth *dep,
        struct iovec_dat *iovp, vir_bytes offset,
        int nic_addr, vir_bytes count) );
_PROTOTYPE( typedef void (*dp_nic2userf_t), (struct dpeth *dep,
        int nic_addr, struct iovec_dat *iovp,
        vir_bytes offset, vir_bytes count) );

#if 0
_PROTOTYPE( typedef void (*dp_getheaderf_t), (struct dpeth *dep,
        int page, struct dp_rcvhdr *h, ul6_t *eth_type) );
#endif
_PROTOTYPE( typedef void (*dp_getblock_t), (struct dpeth *dep,
        int page, size_t offset, size_t size, void *dst) );

/* iovectors are handled IOVEC_NR entries at a time. */
#define IOVEC_NR      16

typedef int irq_hook_t;

typedef struct iovec_dat
{
    iovec_t iod_iovec[IOVEC_NR];
    int iod_iovec_s;
    int iod_proc_nr;
    vir_bytes iod_iovec_addr;
} iovec_dat_t;

#define SENDQ_NR      2      /* Maximum size of the send queue */
#define SENDQ_PAGES      6      /* 6 * DP_PAGESIZE >= 1514 bytes */

typedef struct dpeth
{
    /* The de_base_port field is the starting point of the probe.
     * The conf routine also fills de_linmem and de_irq. If the probe
     * routine knows the irq and/or memory address because they are
     * hardwired in the board, the probe should modify these fields.
     * Furthermore, the probe routine should also fill in de_initf and
     * de_stopf fields with the appropriate function pointers and set
     * de_prog_IO iff programmed I/O is to be used.
     */
    port_t de_base_port;
    phys_bytes de_linmem;
    int de_irq;
    int de_int_pending;
    irq_hook_t de_hook;
    dp_initf_t de_initf;
    dp_stopf_t de_stopf;

```

```

int de_prog_IO;
char de_name[sizeof("dp8390#n")];

/* The initf function fills the following fields. Only cards that do
 * programmed I/O fill in the de_pata_port field.
 * In addition, the init routine has to fill in the sendq data
 * structures.
 */
ether_addr_t de_address;
port_t de_dp8390_port;
port_t de_data_port;
int de_16bit;
int de_ramsize;
int de_offset_page;
int de_startpage;
int de_stoppage;

#if ENABLE_PCI
/* PCI config */
char de_pci; /* TRUE iff PCI device */
u8_t de_pcibus;
u8_t de_pcidev;
u8_t de_pcifunc;
#endif

/* Do it yourself send queue */
struct sendq
{
    int sq_filled; /* this buffer contains a packet */
    int sq_size; /* with this size */
    int sq_sendpage; /* starting page of the buffer */
} de_sendq[SENDQ_NR];
int de_sendq_nr;
int de_sendq_head; /* Enqueue at the head */
int de_sendq_tail; /* Dequeue at the tail */

/* Fields for internal use by the dp8390 driver. */
int de_flags;
int de_mode;
eth_stat_t de_stat;
iovec_dat_t de_read_iovec;
iovec_dat_t de_write_iovec;
iovec_dat_t de_tmp_iovec;
vir_bytes de_read_s;
int de_client;
message de_sendmsg;
dp_user2nicf_t de_user2nicf;
dp_nic2userf_t de_nic2userf;
dp_getblock_t de_getblockf;
} dpeth_t;

#define DEI_DEFAULT 0x8000

#define DEF_EMPTY 0x000
#define DEF_PACK_SEND 0x001
#define DEF_PACK_RECV 0x002
#define DEF_SEND_AVAIL 0x004
#define DEF_READING 0x010
#define DEF_PROMISC 0x040
#define DEF_MULTI 0x080
#define DEF_BROAD 0x100
#define DEF_ENABLED 0x200
#define DEF_STOPPED 0x400

#define DEM_DISABLED 0x0
#define DEM_SINK 0x1
#define DEM_ENABLED 0x2

#if !__minix_vmd
#define debug 0 /* Standard Minix lacks debug variable */
#endif

/*
 * $PchId: dp8390.h,v 1.10 2005/02/10 17:26:06 philip Exp $

```

\* /



```
/*
local.h
*/

#define ENABLE_WDETH 1
#define ENABLE_NE2000 1
#define ENABLE_3C503 1
#define ENABLE_PCI 1

struct dpeth;

/* 3c503.c */
_PROTOTYPE( int el2_probe, (struct dpeth* dep) );

/* dp8390.c */
_PROTOTYPE( u8_t inb, (port_t port) );
_PROTOTYPE( u16_t inw, (port_t port) );
_PROTOTYPE( void outb, (port_t port, u8_t v) );
_PROTOTYPE( void outw, (port_t port, u16_t v) );

/* ne2000.c */
_PROTOTYPE( int ne_probe, (struct dpeth *dep) );
_PROTOTYPE( void ne_init, (struct dpeth *dep) );

/* rtl8029.c */
_PROTOTYPE( int rtl_probe, (struct dpeth *dep) );

/* wdeth.c */
_PROTOTYPE( int wdeth_probe, (struct dpeth* dep) );
```

```

/*
ne2000.c

Driver for the ne2000 ethernet cards. This file contains only the ne2000
specific code, the rest is in dp8390.c

Created:      March 15, 1994 by Philip Homburg <philip@f-mnx.phicoh.com>
*/

#include "../drivers.h"

#include <net/gen/ether.h>
#include <net/gen/eth_io.h>
#if __minix_vmd
#include "config.h"
#endif

#include "local.h"
#include "dp8390.h"
#include "ne2000.h"

#if ENABLE_NE2000

#define N 100

#define MILLIS_TO_TICKS(m)  (((m)*HZ/1000)+1)

_PROTOTYPE( typedef int (*testf_t), (dpeth_t *dep, int pos, u8_t *pat) );

u8_t pat0[] = { 0x00, 0x00, 0x00, 0x00 };
u8_t pat1[] = { 0xFF, 0xFF, 0xFF, 0xFF };
u8_t pat2[] = { 0xA5, 0x5A, 0x69, 0x96 };
u8_t pat3[] = { 0x96, 0x69, 0x5A, 0xA5 };

_PROTOTYPE( static int test_8, (dpeth_t *dep, int pos, u8_t *pat) );
_PROTOTYPE( static int test_16, (dpeth_t *dep, int pos, u8_t *pat) );
_PROTOTYPE( static void ne_stop, (dpeth_t *dep) );
_PROTOTYPE( static void milli_delay, (unsigned long millis) );

/*=====
*
* ne_probe
*=====*/
int ne_probe(dep)
dpeth_t *dep;
{
    int byte;
    int i;
    int loc1, loc2;
    testf_t f;

    dep->de_dp8390_port = dep->de_base_port + NE_DP8390;

    /* We probe for an ne1000 or an ne2000 by testing whether the
    * on board is reachable through the dp8390. Note that the
    * ne1000 is an 8bit card and has a memory region distict from
    * the 16bit ne2000
    */

    for (dep->de_16bit = 0; dep->de_16bit < 2; dep->de_16bit++)
    {
        /* Reset the ethernet card */
        byte = inb_ne(dep, NE_RESET);
        milli_delay(2);
        outb_ne(dep, NE_RESET, byte);
        milli_delay(2);

        /* Reset the dp8390 */
        outb_reg0(dep, DP_CR, CR_STP | CR_DM_ABORT);
        for (i = 0; i < 0x1000 && ((inb_reg0(dep, DP_ISR) & ISR_RST) == 0); i++)
            ; /* Do nothing */

        /* Check if the dp8390 is really there */
        if ((inb_reg0(dep, DP_CR) & (CR_STP | CR_DM_ABORT)) !=
            (CR_STP | CR_DM_ABORT))
    }
}

```

```

    {
        return 0;
    }

    /* Disable the receiver and init TCR and DCR. */
    outb_reg0(dep, DP_RCR, RCR_MON);
    outb_reg0(dep, DP_TCR, TCR_NORMAL);
    if (dep->de_16bit)
    {
        outb_reg0(dep, DP_DCR, DCR_WORDWIDE | DCR_8BYTES |
                    DCR_BMS);
    }
    else
    {
        outb_reg0(dep, DP_DCR, DCR_BYTEWIDE | DCR_8BYTES |
                    DCR_BMS);
    }

    if (dep->de_16bit)
    {
        loc1= NE2000_START;
        loc2= NE2000_START + NE2000_SIZE - 4;
        f= test_16;
    }
    else
    {
        loc1= NE1000_START;
        loc2= NE1000_START + NE1000_SIZE - 4;
        f= test_8;
    }
    if (f(dep, loc1, pat0) && f(dep, loc1, pat1) &&
        f(dep, loc1, pat2) && f(dep, loc1, pat3) &&
        f(dep, loc2, pat0) && f(dep, loc2, pat1) &&
        f(dep, loc2, pat2) && f(dep, loc2, pat3))
    {
        /* We don't need a memory segment */
        dep->de_linmem= 0;
        if (!dep->de_pci)
            dep->de_initf= ne_init;
        dep->de_stopf= ne_stop;
        dep->de_prog_IO= 1;
        return 1;
    }
}
return 0;
}

/*=====
 *                               ne_init                               *
 *=====*/
void ne_init(dep)
dpeth_t *dep;
{
    int i;
    int word, sendq_nr;

    /* Setup a transfer to get the ethernet address. */
    if (dep->de_16bit)
        outb_reg0(dep, DP_RBCR0, 6*2);
    else
        outb_reg0(dep, DP_RBCR0, 6);
    outb_reg0(dep, DP_RBCR1, 0);
    outb_reg0(dep, DP_RSAR0, 0);
    outb_reg0(dep, DP_RSAR1, 0);
    outb_reg0(dep, DP_CR, CR_DM_RR | CR_PS_P0 | CR_STA);

    for (i= 0; i<6; i++)
    {
        if (dep->de_16bit)
        {
            word= inw_ne(dep, NE_DATA);
            dep->de_address.ea_addr[i]= word;
        }
        else

```

```

        {
            dep->de_address.ea_addr[i] = inb_ne(dep, NE_DATA);
        }
    }
    dep->de_data_port= dep->de_base_port + NE_DATA;
    if (dep->de_16bit)
    {
        dep->de_ramsize= NE2000_SIZE;
        dep->de_offset_page= NE2000_START / DP_PAGESIZE;
    }
    else
    {
        dep->de_ramsize= NE1000_SIZE;
        dep->de_offset_page= NE1000_START / DP_PAGESIZE;
    }

    /* Allocate one send buffer (1.5KB) per 8KB of on board memory. */
    sendq_nr= dep->de_ramsize / 0x2000;
    if (sendq_nr < 1)
        sendq_nr= 1;
    else if (sendq_nr > SENDQ_NR)
        sendq_nr= SENDQ_NR;
    dep->de_sendq_nr= sendq_nr;
    for (i= 0; i<sendq_nr; i++)
    {
        dep->de_sendq[i].sq_sendpage= dep->de_offset_page +
            i*SENDQ_PAGES;
    }

    dep->de_startpage= dep->de_offset_page + i*SENDQ_PAGES;
    dep->de_stoppage= dep->de_offset_page + dep->de_ramsize / DP_PAGESIZE;

    /* Can't override the default IRQ. */
    dep->de_irq &= ~DEI_DEFAULT;

    if (!debug)
    {
        printf("%s: NE%d000 at %X:%d\n",
            dep->de_name, dep->de_16bit ? 2 : 1,
            dep->de_base_port, dep->de_irq);
    }
    else
    {
        printf("%s: Novell NE%d000 ethernet card at I/O address "
            "0x%X, memory size 0x%X, irq %d\n",
            dep->de_name, dep->de_16bit ? 2 : 1,
            dep->de_base_port, dep->de_ramsize, dep->de_irq);
    }
}

/*=====
*                                     test_8                                     *
*=====*/
static int test_8(dep, pos, pat)
dpeth_t *dep;
int pos;
u8_t *pat;
{
    u8_t buf[4];
    int i;
    int r;

    outb_reg0(dep, DP_ISR, 0xFF);

    /* Setup a transfer to put the pattern. */
    outb_reg0(dep, DP_RBCR0, 4);
    outb_reg0(dep, DP_RBCR1, 0);
    outb_reg0(dep, DP_RSAR0, pos & 0xFF);
    outb_reg0(dep, DP_RSAR1, pos >> 8);
    outb_reg0(dep, DP_CR, CR_DM_RW | CR_PS_P0 | CR_STA);

    for (i= 0; i<4; i++)
        outb_ne(dep, NE_DATA, pat[i]);
}

```

```

    for (i= 0; i<N; i++)
    {
        if (inb_reg0(dep, DP_ISR) & ISR_RDC)
            break;
    }
    if (i == N)
    {
        if (debug)
        {
            printf("%s: NE1000 remote DMA test failed\n",
                dep->de_name);
        }
        return 0;
    }

    outb_reg0(dep, DP_RBCR0, 4);
    outb_reg0(dep, DP_RBCR1, 0);
    outb_reg0(dep, DP_RSAR0, pos & 0xFF);
    outb_reg0(dep, DP_RSAR1, pos >> 8);
    outb_reg0(dep, DP_CR, CR_DM_RR | CR_PS_P0 | CR_STA);

    for (i= 0; i<4; i++)
        buf[i]= inb_ne(dep, NE_DATA);

    r= (memcmp(buf, pat, 4) == 0);
    return r;
}

/*=====
 *                               test_16                               *
 *=====*/
static int test_16(dep, pos, pat)
dpeth_t *dep;
int pos;
u8_t *pat;
{
    u8_t buf[4];
    int i;
    int r;

    outb_reg0(dep, DP_ISR, 0xFF);

    /* Setup a transfer to put the pattern. */
    outb_reg0(dep, DP_RBCR0, 4);
    outb_reg0(dep, DP_RBCR1, 0);
    outb_reg0(dep, DP_RSAR0, pos & 0xFF);
    outb_reg0(dep, DP_RSAR1, pos >> 8);
    outb_reg0(dep, DP_CR, CR_DM_RW | CR_PS_P0 | CR_STA);

    for (i= 0; i<4; i += 2)
    {
        outw_ne(dep, NE_DATA, *(u16_t *)(pat+i));
    }

    for (i= 0; i<N; i++)
    {
        if (inb_reg0(dep, DP_ISR) & ISR_RDC)
            break;
    }
    if (i == N)
    {
        if (debug)
        {
            printf("%s: NE2000 remote DMA test failed\n",
                dep->de_name);
        }
        return 0;
    }

    outb_reg0(dep, DP_RBCR0, 4);
    outb_reg0(dep, DP_RBCR1, 0);
    outb_reg0(dep, DP_RSAR0, pos & 0xFF);
    outb_reg0(dep, DP_RSAR1, pos >> 8);
    outb_reg0(dep, DP_CR, CR_DM_RR | CR_PS_P0 | CR_STA);

```

```
    for (i= 0; i<4; i += 2)
    {
        *(u16_t *) (buf+i)= inw_ne(dep, NE_DATA);
    }

    r= (memcmp(buf, pat, 4) == 0);
    return r;
}

/*=====
 *                               ne_stop                               *
 *=====*/
static void ne_stop(dep)
dpeth_t *dep;
{
    int byte;

    /* Reset the ethernet card */
    byte= inb_ne(dep, NE_RESET);
    milli_delay(2);
    outb_ne(dep, NE_RESET, byte);
}

static void milli_delay(unsigned long millis)
{
    tickdelay(MILLIS_TO_TICKS(millis));
}

#endif /* ENABLE_NE2000 */

/*
 * $PchId: ne2000.c,v 1.10 2004/08/03 12:03:00 philip Exp $
 */
```

```
/*
ne2000.h

Created:      March 15, 1994 by Philip Homburg <philip@f-mnx.phicoh.com>
*/

#ifndef NE2000_H
#define NE2000_H

#define NE_DP8390      0x00
#define NE_DATA        0x10
#define NE_RESET       0x1F

#define NE1000_START   0x2000
#define NE1000_SIZE     0x2000
#define NE2000_START   0x4000
#define NE2000_SIZE     0x4000

#define inb_ne(dep, reg)      (inb(dep->de_base_port+reg))
#define outb_ne(dep, reg, data) (outb(dep->de_base_port+reg, data))
#define inw_ne(dep, reg)      (inw(dep->de_base_port+reg))
#define outw_ne(dep, reg, data) (outw(dep->de_base_port+reg, data))

#endif /* NE2000_H */

/*
 * $PchId: ne2000.h,v 1.4 2004/08/03 12:03:20 philip Exp $
 */
```

```

/*
rtl8029.c

Initialization of PCI DP8390-based ethernet cards

Created:      April 2000 by Philip Homburg <philip@f-mnx.phicoh.com>
*/

#include "../drivers.h"

#include <stdlib.h>
#include <sys/types.h>
#include <net/gen/ether.h>
#include <net/gen/eth_io.h>
#include <ibm/pci.h>

#include "assert.h"

#include "local.h"
#include "dp8390.h"
#include "rtl8029.h"

#if ENABLE_PCI

#define MICROS_TO_TICKS(m)  (((m)*HZ/1000000)+1)

PRIVATE struct pcitab
{
    u16_t vid;
    u16_t did;
    int checkclass;
} pcitab[] =
{
    { 0x10ec, 0x8029, 0 },          /* Realtek RTL8029 */
    { 0x0000, 0x0000, 0 }
};

_PROTOTYPE( static void rtl_init, (struct dpeth *dep) ) ;
_PROTOTYPE( static u16_t get_ee_word, (dpeth_t *dep, int a) ) ;
_PROTOTYPE( static void ee_wen, (dpeth_t *dep) ) ;
_PROTOTYPE( static void set_ee_word, (dpeth_t *dep, int a, U16_t w) ) ;
_PROTOTYPE( static void ee_wds, (dpeth_t *dep) ) ;
_PROTOTYPE( static void micro_delay, (unsigned long usecs) ) ;

PUBLIC int rtl_probe(dep)
struct dpeth *dep;
{
    int i, r, devind, just_one;
    u16_t vid, did;
    u32_t bar;
    u8_t ilr;
    char *dname;

    pci_init();

    if ((dep->de_pcibus | dep->de_pcidev | dep->de_pcifunc) != 0)
    {
        /* Look for specific PCI device */
        r= pci_find_dev(dep->de_pcibus, dep->de_pcidev,
                        dep->de_pcifunc, &devind);
        if (r == 0)
        {
            printf("%s: no PCI found at %d.%d.%d\n",
                    dep->de_name, dep->de_pcibus,
                    dep->de_pcidev, dep->de_pcifunc);
            return 0;
        }
        pci_ids(devind, &vid, &did);
        just_one= TRUE;
    }
    else
    {
        r= pci_first_dev(&devind, &vid, &did);
    }
}

```



```

        if (r == 0)
            return 0;
        just_one= FALSE;
    }

    for(;;)
    {
        for (i= 0; pcitab[i].vid != 0; i++)
        {
            if (pcitab[i].vid != vid)
                continue;
            if (pcitab[i].did != did)
                continue;
            if (pcitab[i].checkclass)
            {
                panic(" ",
                    "rtl_probe: class check not implemented" ,
                    NO_NUM);
            }
            break;
        }
        if (pcitab[i].vid != 0)
            break;

        if (just_one)
        {
            printf(
                "%s: wrong PCI device (%04X/%04X) found at %d.%d.%d\n",
                dep->de_name, vid, did,
                dep->de_pcibus,
                dep->de_pcidev, dep->de_pcifunc);
            return 0;
        }

        r= pci_next_dev(&devind, &vid, &did);
        if (!r)
            return 0;
    }

    dname= pci_dev_name(vid, did);
    if (!dname)
        dname= "unknown device";
    printf("%s: %s (%04X/%04X) at %s\n",
        dep->de_name, dname, vid, did, pci_slot_name(devind));
    pci_reserve(devind);
    /* printf("cr = 0x%x\n", pci_attr_r16(devind, PCI_CR)); */
    bar= pci_attr_r32(devind, PCI_BAR) & 0xffffffe0;

    if (bar < 0x400)
        panic(" ", "base address is not properly configured" , NO_NUM);

    dep->de_base_port= bar;

    ilr= pci_attr_r8(devind, PCI_ILR);
    dep->de_irq= ilr;
    if (debug)
    {
        printf("%s: using I/O address 0x%lx, IRQ %d\n",
            dep->de_name, (unsigned long)bar, ilr);
    }
    dep->de_initf= rtl_init;

    return TRUE;
}

static void rtl_init(dep)
dpeth_t *dep;
{
    u8_t reg_a, reg_b, cr, config0, config2, config3;
    int i;

#ifdef DEBUG
    printf("rtl_init called\n");
#endif

```

```
ne_init(dep);

/* ID */
outb_reg0(dep, DP_CR, CR_PS_P0);
reg_a = inb_reg0(dep, DP_DUM1);
reg_b = inb_reg0(dep, DP_DUM2);

#if DEBUG
printf("rtl_init: '%c', '%c'\n", reg_a, reg_b);
#endif

outb_reg0(dep, DP_CR, CR_PS_P3);
config0 = inb_reg3(dep, 3);
config2 = inb_reg3(dep, 5);
config3 = inb_reg3(dep, 6);
outb_reg0(dep, DP_CR, CR_PS_P0);

#if DEBUG
printf("rtl_init: config 0/2/3 = %x/%x/%x\n",
        config0, config2, config3);
#endif

if (getenv("RTL8029FD"))
{
    printf("rtl_init: setting full-duplex mode\n");
    outb_reg0(dep, DP_CR, CR_PS_P3);

    cr = inb_reg3(dep, 1);
    outb_reg3(dep, 1, cr | 0xc0);

    outb_reg3(dep, 6, config3 | 0x40);
    config3 = inb_reg3(dep, 6);

    config2 = inb_reg3(dep, 5);
    outb_reg3(dep, 5, config2 | 0x20);
    config2 = inb_reg3(dep, 5);

    outb_reg3(dep, 1, cr);

    outb_reg0(dep, DP_CR, CR_PS_P0);

#if DEBUG
printf("rtl_init: config 2 = %x\n", config2);
printf("rtl_init: config 3 = %x\n", config3);
#endif
}

#if DEBUG
for (i = 0; i < 64; i++)
    printf("%x ", get_ee_word(dep, i));
printf("\n");
#endif

if (getenv("RTL8029MN"))
{
    ee_wen(dep);

    set_ee_word(dep, 0x78/2, 0x10ec);
    set_ee_word(dep, 0x7A/2, 0x8029);
    set_ee_word(dep, 0x7C/2, 0x10ec);
    set_ee_word(dep, 0x7E/2, 0x8029);

    ee_wds(dep);

    assert(get_ee_word(dep, 0x78/2) == 0x10ec);
    assert(get_ee_word(dep, 0x7A/2) == 0x8029);
    assert(get_ee_word(dep, 0x7C/2) == 0x10ec);
    assert(get_ee_word(dep, 0x7E/2) == 0x8029);
}

if (getenv("RTL8029XXX"))
{
    ee_wen(dep);
```

```

        set_ee_word(dep, 0x76/2, 0x8029);

        ee_wds(dep);

        assert(get_ee_word(dep, 0x76/2) == 0x8029);
    }
}

static ul6_t get_ee_word(dep, a)
dpeth_t *dep;
int a;
{
    int b, i, cmd;
    ul6_t w;

    outb_reg0(dep, DP_CR, CR_PS_P3);          /* Bank 3 */

    /* Switch to 9346 mode and enable CS */
    outb_reg3(dep, 1, 0x80 | 0x8);

    cmd= 0x180 | (a & 0x3f);          /* 1 1 0 a5 a4 a3 a2 a1 a0 */
    for (i= 8; i >= 0; i--)
    {
        b= (cmd & (1 << i));
        b= (b ? 2 : 0);

        /* Cmd goes out on the rising edge of the clock */
        outb_reg3(dep, 1, 0x80 | 0x8 | b);
        outb_reg3(dep, 1, 0x80 | 0x8 | 0x4 | b);
    }
    outb_reg3(dep, 1, 0x80 | 0x8); /* End of cmd */

    w= 0;
    for (i= 0; i<16; i++)
    {
        w <= 1;

        /* Data is shifted out on the rising edge. Read at the
         * falling edge.
         */
        outb_reg3(dep, 1, 0x80 | 0x8 | 0x4);
        outb_reg3(dep, 1, 0x80 | 0x8 | b);
        b= inb_reg3(dep, 1);
        w |= (b & 1);
    }

    outb_reg3(dep, 1, 0x80);          /* drop CS */
    outb_reg3(dep, 1, 0x00);          /* back to normal */
    outb_reg0(dep, DP_CR, CR_PS_P0);  /* back to bank 0 */

    return w;
}

static void ee_wen(dep)
dpeth_t *dep;
{
    int b, i, cmd;

    outb_reg0(dep, DP_CR, CR_PS_P3);          /* Bank 3 */

    /* Switch to 9346 mode and enable CS */
    outb_reg3(dep, 1, 0x80 | 0x8);

    cmd= 0x130;          /* 1 0 0 1 1 x x x x */
    for (i= 8; i >= 0; i--)
    {
        b= (cmd & (1 << i));
        b= (b ? 2 : 0);

        /* Cmd goes out on the rising edge of the clock */
        outb_reg3(dep, 1, 0x80 | 0x8 | b);
        outb_reg3(dep, 1, 0x80 | 0x8 | 0x4 | b);
    }
    outb_reg3(dep, 1, 0x80 | 0x8); /* End of cmd */
}

```

```

    outb_reg3(dep, 1, 0x80);          /* Drop CS */
    micro_delay(1);                   /* Is this required? */
}

static void set_ee_word(dep, a, w)
dpeth_t *dep;
int a;
ul6_t w;
{
    int b, i, cmd;

    outb_reg3(dep, 1, 0x80 | 0x8);    /* Set CS */

    cmd= 0x140 | (a & 0x3f);          /* 1 0 1 a5 a4 a3 a2 a1 a0 */
    for (i= 8; i >= 0; i--)
    {
        b= (cmd & (1 << i));
        b= (b ? 2 : 0);

        /* Cmd goes out on the rising edge of the clock */
        outb_reg3(dep, 1, 0x80 | 0x8 | b);
        outb_reg3(dep, 1, 0x80 | 0x8 | 0x4 | b);
    }
    for (i= 15; i >= 0; i--)
    {
        b= (w & (1 << i));
        b= (b ? 2 : 0);

        /* Cmd goes out on the rising edge of the clock */
        outb_reg3(dep, 1, 0x80 | 0x8 | b);
        outb_reg3(dep, 1, 0x80 | 0x8 | 0x4 | b);
    }
    outb_reg3(dep, 1, 0x80 | 0x8);    /* End of data */
    outb_reg3(dep, 1, 0x80);          /* Drop CS */
    micro_delay(1);                   /* Is this required? */
    outb_reg3(dep, 1, 0x80 | 0x8);    /* Set CS */
    for (i= 0; i<10000; i++)
    {
        if (inb_reg3(dep, 1) & 1)
            break;
        micro_delay(1);
    }
    if (!(inb_reg3(dep, 1) & 1))
        panic("", "set_ee_word: device remains busy", NO_NUM);
}

static void ee_wds(dep)
dpeth_t *dep;
{
    int b, i, cmd;

    outb_reg0(dep, DP_CR, CR_PS_P3); /* Bank 3 */

    /* Switch to 9346 mode and enable CS */
    outb_reg3(dep, 1, 0x80 | 0x8);

    cmd= 0x100;                       /* 1 0 0 0 0 x x x x */
    for (i= 8; i >= 0; i--)
    {
        b= (cmd & (1 << i));
        b= (b ? 2 : 0);

        /* Cmd goes out on the rising edge of the clock */
        outb_reg3(dep, 1, 0x80 | 0x8 | b);
        outb_reg3(dep, 1, 0x80 | 0x8 | 0x4 | b);
    }
    outb_reg3(dep, 1, 0x80 | 0x8);    /* End of cmd */
    outb_reg3(dep, 1, 0x80);          /* Drop CS */
    outb_reg3(dep, 1, 0x00);          /* back to normal */
    outb_reg0(dep, DP_CR, CR_PS_P0); /* back to bank 0 */
}

static void micro_delay(unsigned long uses)
{

```

```
        tickdelay(MICROS_TO_TICKS(usecs));  
    }  
  
#endif /* ENABLE_PCI */  
  
/*  
 * $PchId: rtl8029.c,v 1.7 2004/08/03 12:16:58 philip Exp $  
 */
```

```
/*
rtl8029.h

Created:      Sep 2003 by Philip Homburg <philip@f-mnx.phicoh.com>
*/

/* Bits in dp_cr */
#define CR_PS_P3      0xC0      /* Register Page 3 */

#define inb_reg3(dep, reg)      (inb (dep->de_dp8390_port+reg))
#define outb_reg3(dep, reg, data) (outb(dep->de_dp8390_port+reg, data))

/*
 * $PchId: rtl8029.h,v 1.3 2004/08/03 15:11:06 philip Exp $
 */
```

```

/*
wdeth.c

Created:      March 14, 1994 by Philip Homburg
*/

#include "../drivers.h"

#include <net/gen/ether.h>
#include <net/gen/eth_io.h>
#include "assert.h"

#include "local.h"
#include "dp8390.h"
#include "wdeth.h"

#if ENABLE_WDETH

#define WET_ETHERNET      0x01          /* Ethernet transceiver */
#define WET_STARLAN      0x02          /* Starlan transceiver */
#define WET_INTERF_CHIP  0x04          /* has a WD83C583 interface chip */
#define WET_BRD_16BIT     0x08          /* 16 bit board */
#define WET_SLT_16BIT     0x10          /* 16 bit slot */
#define WET_790           0x20          /* '790 chip */

static int we_int_table[8]= { 9, 3, 5, 7, 10, 11, 15, 4 };
static int we_790int_table[8]= { 0, 9, 3, 5, 7, 10, 11, 15 };

_PROTOTYPE( static void we_init, (dpeth_t *dep) )
_PROTOTYPE( static void we_stop, (dpeth_t *dep) )
_PROTOTYPE( static int we_aliasing, (dpeth_t *dep) )
_PROTOTYPE( static int we_interface_chip, (dpeth_t *dep) )
_PROTOTYPE( static int we_16bitboard, (dpeth_t *dep) )
_PROTOTYPE( static int we_16bitslot, (dpeth_t *dep) )
_PROTOTYPE( static int we_ultra, (dpeth_t *dep) )

/*=====
 *                               wdeth_probe                               *
 *=====*/
int wdeth_probe(dep)
dpeth_t *dep;
{
    int sum;

    if (dep->de_linmem == 0)
        return 0;          /* No shared memory, so no WD board */

    sum = inb_we(dep, EPL_EA0) + inb_we(dep, EPL_EA1) +
          inb_we(dep, EPL_EA2) + inb_we(dep, EPL_EA3) +
          inb_we(dep, EPL_EA4) + inb_we(dep, EPL_EA5) +
          inb_we(dep, EPL_TLB) + inb_we(dep, EPL_CHKSUM);
    if ((sum & 0xFF) != 0xFF)
        return 0;          /* No ethernet board at this address */

    dep->de_initf= we_init;
    dep->de_stopf= we_stop;
    dep->de_prog_IO= 0;
    return 1;
}

/*=====
 *                               we_init                               *
 *=====*/
static void we_init(dep)
dpeth_t *dep;
{
    int i, int_idx, int_nr;
    int tlb, rambit, revision;
    int icr, irr, hwr, b, gcr;
    int we_type;
    int sendq_nr;

    assert(dep->de_mode == DEM_ENABLED);
    assert(!(dep->de_flags & DEF_ENABLED));

```

```
dep->de_address.ea_addr[0] = inb_we(dep, EPL_EA0);
dep->de_address.ea_addr[1] = inb_we(dep, EPL_EA1);
dep->de_address.ea_addr[2] = inb_we(dep, EPL_EA2);
dep->de_address.ea_addr[3] = inb_we(dep, EPL_EA3);
dep->de_address.ea_addr[4] = inb_we(dep, EPL_EA4);
dep->de_address.ea_addr[5] = inb_we(dep, EPL_EA5);

dep->de_dp8390_port= dep->de_base_port + EPL_DP8390;

dep->de_16bit= 0;

we_type= 0;
we_type |= WET_ETHERNET;          /* assume ethernet */
if (we_ultra(dep))
    we_type |= WET_790;
if (!we_aliasing(dep))
{
    if (we_interface_chip(dep))
        we_type |= WET_INTERF_CHIP;
    if (we_16bitboard(dep))
    {
        we_type |= WET_BRD_16BIT;
        if (we_16bitslot(dep))
            we_type |= WET_SLT_16BIT;
    }
}
if (we_type & WET_SLT_16BIT)
    dep->de_16bit= 1;

/* look at the on board ram size. */
tlb= inb_we(dep, EPL_TLB);
revision= tlb & E_TLB_REV;
rambit= tlb & E_TLB_RAM;

if (dep->de_ramsize != 0)
{
    /* size set from boot environment. */
}
else if (revision < 2)
{
    dep->de_ramsize= 0x2000;          /* 8K */
    if (we_type & WET_BRD_16BIT)
        dep->de_ramsize= 0x4000;    /* 16K */
    else if ((we_type & WET_INTERF_CHIP) &&
             inb_we(dep, EPL_ICR) & E_ICR_MEMBIT)
    {
        dep->de_ramsize= 0x8000;    /* 32K */
    }
}
else
{
    if (we_type & WET_BRD_16BIT)
    {
        /* 32K or 16K */
        dep->de_ramsize= rambit ? 0x8000 : 0x4000;
    }
    else
    {
        /* 32K or 8K */
        dep->de_ramsize= rambit ? 0x8000 : 0x2000;
    }
}

if (we_type & WET_790)
{
    outb_we(dep, EPL_MSR, E_MSR_RESET);
    if ((we_type & (WET_BRD_16BIT|WET_SLT_16BIT)) ==
        (WET_BRD_16BIT|WET_SLT_16BIT))
    {
        outb_we(dep, EPL_LAAR, E_LAAR_LAN16E | E_LAAR_MEM16E);
    }
}
else if (we_type & WET_BRD_16BIT)
```



```

    {
        if (we_type & WET_SLT_16BIT)
        {
            outb_we(dep, EPL_LAAR, E_LAAR_A19 | E_LAAR_SOFTINT |
                    E_LAAR_LAN16E | E_LAAR_MEM16E);
        }
        else
        {
            outb_we(dep, EPL_LAAR, E_LAAR_A19 | E_LAAR_SOFTINT |
                    E_LAAR_LAN16E);
        }
    }

    if (we_type & WET_790)
    {
        outb_we(dep, EPL_MSR, E_MSR_MENABLE);
        hwr= inb_we(dep, EPL_790_HWR);
        outb_we(dep, EPL_790_HWR, hwr | E_790_HWR_SWH);
        b= inb_we(dep, EPL_790_B);
        outb_we(dep, EPL_790_B, ((dep->de_linmem >> 13) & 0x0f) |
            ((dep->de_linmem >> 11) & 0x40) | (b & 0xb0));
        outb_we(dep, EPL_790_HWR, hwr & ~E_790_HWR_SWH);
    }
    else
    {
        outb_we(dep, EPL_MSR, E_MSR_RESET);
        outb_we(dep, EPL_MSR, E_MSR_MENABLE |
            ((dep->de_linmem >> 13) & E_MSR_MEMADDR));
    }

    if ((we_type & WET_INTERF_CHIP) && !(we_type & WET_790))
    {
        icr= inb_we(dep, EPL_ICR);
        irr= inb_we(dep, EPL_IRR);
        int_indx= (icr & E_ICR_IR2) |
            ((irr & (E_IRR_IR0|E_IRR_IR1)) >> 5);
        int_nr= we_int_table[int_indx];

    #if DEBUG
    { printf("%s: encoded irq= %d\n", dep->de_name, int_nr); }
    #endif

        if (dep->de_irq & DEI_DEFAULT) dep->de_irq= int_nr;

        outb_we(dep, EPL_IRR, irr | E_IRR_IEN);
    }
    if (we_type & WET_790)
    {
        hwr= inb_we(dep, EPL_790_HWR);
        outb_we(dep, EPL_790_HWR, hwr | E_790_HWR_SWH);

        gcr= inb_we(dep, EPL_790_GCR);

        outb_we(dep, EPL_790_HWR, hwr & ~E_790_HWR_SWH);

        int_indx= ((gcr & E_790_GCR_IR2) >> 4) |
            ((gcr & (E_790_GCR_IR1|E_790_GCR_IR0)) >> 2);
        int_nr= we_790int_table[int_indx];

    #if DEBUG
    { printf("%s: encoded irq= %d\n", dep->de_name, int_nr); }
    #endif

        if (dep->de_irq & DEI_DEFAULT) dep->de_irq= int_nr;

        icr= inb_we(dep, EPL_790_ICR);
        outb_we(dep, EPL_790_ICR, icr | E_790_ICR_EIL);
    }

    /* Strip the "default flag." */
    dep->de_irq &= ~DEI_DEFAULT;

    if (!debug)
    {
        printf("%s: WD80%d3 at %X:%d:%lX\n",
            dep->de_name, we_type & WET_BRD_16BIT ? 1 : 0,
            dep->de_base_port, dep->de_irq, dep->de_linmem);
    }
}

```

```

    else
    {
        printf( "%s: Western Digital %s%s card %s%s at I/O "
                "address 0x%X, memory address 0x%IX, "
                "memory size 0x%X, irq %d\n",
                dep->de_name,
                we_type & WET_BRD_16BIT ? "16-bit " : "",
                we_type & WET_ETHERNET ? "Ethernet" :
                we_type & WET_STARLAN ? "Starlan" : "Network",
                we_type & WET_INTERF_CHIP ? "with an interface chip " : "",
                we_type & WET_SLT_16BIT ? "in a 16-bit slot " : "",
                dep->de_base_port, dep->de_linmem, dep->de_ramsize,
                dep->de_irq);
    }

    dep->de_offset_page= 0;          /* Shared memory starts at 0 */

    /* Allocate one send buffer (1.5KB) per 8KB of on board memory. */
    sendq_nr= dep->de_ramsize / 0x2000;
    if (sendq_nr < 1)
        sendq_nr= 1;
    else if (sendq_nr > SENDQ_NR)
        sendq_nr= SENDQ_NR;
    dep->de_sendq_nr= sendq_nr;
    for (i= 0; i<sendq_nr; i++)
        dep->de_sendq[i].sq_sendpage= i*SENDQ_PAGES;

    dep->de_startpage= i*SENDQ_PAGES;
    dep->de_stoppage= dep->de_ramsize / DP_PAGESIZE;
}

/*=====
 *                               we_stop                               *
 *=====*/
static void we_stop(dep)
dpeth_t *dep;
{
    if (dep->de_16bit)
        outb_we(dep, EPL_LAAR, E_LAAR_A19 | E_LAAR_LAN16E);
    outb_we(dep, EPL_MSR, E_MSR_RESET);
    outb_we(dep, EPL_MSR, 0);
}

/*=====
 *                               we_aliasing                           *
 *=====*/
static int we_aliasing(dep)
dpeth_t *dep;
{
    /* Determine whether wd8003 hardware performs register aliasing. This implies
     * an old WD8003E board. */

    if (inb_we(dep, EPL_REG1) != inb_we(dep, EPL_EA1))
        return 0;
    if (inb_we(dep, EPL_REG2) != inb_we(dep, EPL_EA2))
        return 0;
    if (inb_we(dep, EPL_REG3) != inb_we(dep, EPL_EA3))
        return 0;
    if (inb_we(dep, EPL_REG4) != inb_we(dep, EPL_EA4))
        return 0;
    if (inb_we(dep, EPL_REG7) != inb_we(dep, EPL_CHKSUM))
        return 0;
    return 1;
}

/*=====
 *                               we_interface_chip                       *
 *=====*/
static int we_interface_chip(dep)
dpeth_t *dep;
{
    /* Determine if the board has an interface chip. */

    outb_we(dep, EPL_GP2, 0x35);

```

```
    if (inb_we(dep, EPL_GP2) != 0x35)
        return 0;
    outb_we(dep, EPL_GP2, 0x3A);
    if (inb_we(dep, EPL_GP2) != 0x3A)
        return 0;
    return 1;
}

/*=====
 *                               we_16bitboard
 *=====*/
static int we_16bitboard(dep)
dpeth_t *dep;
{
    /* Determine whether the board is capable of doing 16 bit memory moves.
     * If the 16 bit enable bit is unchangable by software we'll assume an
     * 8 bit board.
     */
    int icr;
    u8_t tlb;

    icr= inb_we(dep, EPL_ICR);

    outb_we(dep, EPL_ICR, icr ^ E_ICR_16BIT);
    if (inb_we(dep, EPL_ICR) == icr)
    {
        tlb= inb_we(dep, EPL_TLB);
#ifdef DEBUG
        printf("%s: tlb= 0x%x\n", dep->de_name, tlb);
#endif
        return tlb == E_TLB_EB || tlb == E_TLB_E ||
            tlb == E_TLB_SMCE || tlb == E_TLB_SMC8216T ||
            tlb == E_TLB_SMC8216C;
    }
    outb_we(dep, EPL_ICR, icr);
    return (icr & E_ICR_16BIT);
}

/*=====
 *                               we_16bitslot
 *=====*/
static int we_16bitslot(dep)
dpeth_t *dep;
{
    /* Determine if the 16 bit board is plugged into a 16 bit slot. */
    return !(inb_we(dep, EPL_ICR) & E_ICR_16BIT);
}

/*=====
 *                               we_ultra
 *=====*/
static int we_ultra(dep)
dpeth_t *dep;
{
    /* Determine if we has an '790 chip. */
    u8_t tlb;

    tlb= inb_we(dep, EPL_TLB);
    return tlb == E_TLB_SMC8216T || tlb == E_TLB_SMC8216C;
}

#endif /* ENABLE_WDETH */

/*
 * $PchId: wdeth.c,v 1.10 2003/09/10 19:31:50 philip Exp $
 */
```

```

/*
wdeth.h

Created:          before Dec 28, 1992 by Philip Homburg
*/

#ifndef WDETH_H
#define WDETH_H

/* Western Digital Ethercard Plus, or WD8003E card. */

#define EPL_REG0          0x0          /* Control(write) and status(read) */
#define EPL_REG1          0x1
#define EPL_REG2          0x2
#define EPL_REG3          0x3
#define EPL_REG4          0x4
#define EPL_REG5          0x5
#define EPL_REG6          0x6
#define EPL_REG7          0x7
#define EPL_EA0           0x8          /* Most significant eaddr byte */
#define EPL_EA1           0x9
#define EPL_EA2           0xA
#define EPL_EA3           0xB
#define EPL_EA4           0xC
#define EPL_EA5           0xD          /* Least significant eaddr byte */
#define EPL_TLB           0xE
#define EPL_CHKSUM        0xF          /* sum from epl_ea0 upto here is 0xFF */
#define EPL_DP8390        0x10         /* NatSemi chip */

#define EPL_MSR            EPL_REG0    /* memory select register */
#define EPL_ICR            EPL_REG1    /* interface configuration register */
#define EPL_IRR            EPL_REG4    /* interrupt request register (IRR) */
#define EPL_790_HWR        EPL_REG4    /* '790 hardware support register */
#define EPL_LAAR           EPL_REG5    /* LA address register (write only) */
#define EPL_790_ICR        EPL_REG6    /* '790 interrupt control register */
#define EPL_GP2            EPL_REG7    /* general purpose register 2 */
#define EPL_790_B          EPL_EA3     /* '790 memory register */
#define EPL_790_GCR        EPL_EA5     /* '790 General Control Register */

/* Bits in EPL_MSR */
#define E_MSR_MEMADDR      0x3F        /* Bits SA18-SA13, SA19 implicit 1 */
#define E_MSR_MENABLE      0x40        /* Memory Enable */
#define E_MSR_RESET        0x80        /* Software Reset */

/* Bits in EPL_ICR */
#define E_ICR_16BIT        0x01        /* 16 bit bus */
#define E_ICR_IR2          0x04        /* bit 2 of encoded IRQ */
#define E_ICR_MEMBIT       0x08        /* 583 mem size mask */

/* Bits in EPL_IRR */
#define E_IRR_IR0          0x20        /* bit 0 of encoded IRQ */
#define E_IRR_IR1          0x40        /* bit 1 of encoded IRQ */
#define E_IRR_IEN          0x80        /* enable interrupts */

/* Bits in EPL_LAAR */
#define E_LAAR_A19         0x01        /* address lines for above 1M ram */
#define E_LAAR_A20         0x02        /* address lines for above 1M ram */
#define E_LAAR_A21         0x04        /* address lines for above 1M ram */
#define E_LAAR_A22         0x08        /* address lines for above 1M ram */
#define E_LAAR_A23         0x10        /* address lines for above 1M ram */
#define E_LAAR_SOFTINT     0x20        /* enable software interrupt */
#define E_LAAR_LAN16E      0x40        /* enables 16 bit RAM for LAN */
#define E_LAAR_MEM16E      0x80        /* enables 16 bit RAM for host */

/* Bits and values in EPL_TLB */
#define E_TLB_EB           0x05        /* WD8013EB */
#define E_TLB_E            0x27        /* WD8013 Elite */
#define E_TLB_SMCE         0x29        /* SMC Elite 16 */
#define E_TLB_SMC8216T     0x2A        /* SMC 8216 T */
#define E_TLB_SMC8216C     0x2B        /* SMC 8216 C */

#define E_TLB_REV          0x1F        /* revision mask */
#define E_TLB_SOFT         0x20        /* soft config */
#define E_TLB_RAM          0x40        /* extra ram bit */

```

```
/* Bits in EPL_790_HWR */
#define E_790_HWR_SWH 0x80 /* switch register set */

/* Bits in EPL_790_ICR */
#define E_790_ICR_EIL 0x01 /* enable interrupts */

/* Bits in EPL_790_GCR when E_790_HWR_SWH is set in EPL_790_HWR */
#define E_790_GCR_IR0 0x04 /* bit 0 of encoded IRQ */
#define E_790_GCR_IR1 0x08 /* bit 1 of encoded IRQ */
#define E_790_GCR_IR2 0x40 /* bit 2 of encoded IRQ */

#define inb_we(dep, reg) (inb(dep->de_base_port+reg))
#define outb_we(dep, reg, data) (outb(dep->de_base_port+reg, data))

#endif /* WDETH_H */

/*
 * $PchId: wdeth.h,v 1.6 2003/09/10 19:29:52 philip Exp $
 */
```

```
/*
** File:          3c501.c          Jan. 14, 1997
**
** Author:        Giovanni Falzoni <gfalzoni@inwind.it>
**
** This file contains specific implementation of the ethernet
** device driver for 3Com Etherlink (3c501) boards. This is a
** very old board and its performances are very poor for today
** network environments.
**
** $Id: 3c501.c,v 1.3 2005/08/05 19:08:43 beng Exp $
*/

#include "drivers.h"
#include <minix/com.h>
#include <net/hton.h>
#include <net/gen/ether.h>
#include <net/gen/eth_io.h>
#include "dp.h"

#if (ENABLE_3C501 == 1)

#include "3c501.h"

static unsigned char StationAddress[SA_ADDR_LEN] = {0, 0, 0, 0, 0, 0,};
static buff_t *TxBuff = NULL;

/*
** Name:          void ell_getstats(dpeth_t *dep)
** Function:      Reads statistics counters from board.
**/
static void ell_getstats(dpeth_t * dep)
{
    return;                /* Nothing to do */
}

/*
** Name:          void ell_reset(dpeth_t *dep)
** Function:      Reset function specific for Etherlink hardware.
**/
static void ell_reset(dpeth_t * dep)
{
    int ix;

    for (ix = 0; ix < 8; ix += 1) /* Resets the board */
        outb_ell(dep, EL1_CSR, ECSR_RESET);
    outb_ell(dep, EL1_CSR, ECSR_RIDE | ECSR_SYS);

    /* Set Ethernet Address on controller */
    outb_ell(dep, EL1_CSR, ECSR_LOOP); /* Loopback mode */
    for (ix = EL1_ADDRESS; ix < SA_ADDR_LEN; ix += 1)
        outb_ell(dep, ix, StationAddress[ix]);

    lock();
    /* Enable DMA/Interrupt, gain control of Buffer */
    outb_ell(dep, EL1_CSR, ECSR_RIDE | ECSR_SYS);
    /* Clear RX packet area */
    outw_ell(dep, EL1_RECVPTR, 0);
    /* Enable transmit/receive configuration and flush pending interrupts */
    outb_ell(dep, EL1_XMIT, EXSR_IDLE | EXSR_16JAM | EXSR_JAM | EXSR_UNDER);
    outb_ell(dep, EL1_RECV, dep->de_recv_mode);
    inb_ell(dep, EL1_RECV);
    inb_ell(dep, EL1_XMIT);
    dep->de_flags &= NOT(DEF_XMIT_BUSY);
    unlock();
    return;                /* Done */
}

/*
** Name:          void ell_dumpstats(dpeth_t *dep, int port, vir_bytes size)
** Function:      Dumps counter on screen (support for console display).
**/
static void ell_dumpstats(dpeth_t * dep)
```

```

{
    return;
}

/*
** Name:      void ell_mode_init(dpeth_t *dep)
** Function:  Initializes receiver mode
*/
static void ell_mode_init(dpeth_t * dep)
{
    if (dep->de_flags & DEF_BROAD) {
        dep->de_rcv_mode = ERSR_BROAD | ERSR_RMASK;
    } else if (dep->de_flags & DEF_PROMISC) {
        dep->de_rcv_mode = ERSR_ALL | ERSR_RMASK;
    } else if (dep->de_flags & DEF_MULTI) {
        dep->de_rcv_mode = ERSR_MULTI | ERSR_RMASK;
    } else {
        dep->de_rcv_mode = ERSR_NONE | ERSR_RMASK;
    }
    outb_ell(dep, EL1_RECV, dep->de_rcv_mode);
    inb_ell(dep, EL1_RECV);
    return;
}

/*
** Name:      void ell_rcv(dpeth_t *dep, int from, int size)
** Function:  Receive function. Called from interrupt handler to
**           unload rcv. buffer or from main (packet to client)
*/
static void ell_rcv(dpeth_t * dep, int from, int size)
{
    buff_t *rxptr;

    while ((dep->de_flags & DEF_READING) && (rxptr = dep->de_rcvq_head)) {
        /* Remove buffer from queue and free buffer */
        lock();
        if (dep->de_rcvq_tail == dep->de_rcvq_head)
            dep->de_rcvq_head = dep->de_rcvq_tail = NULL;
        else
            dep->de_rcvq_head = rxptr->next;
        unlock();

        /* Copy buffer to user area */
        mem2user(dep, rxptr);

        /* Reply information */
        dep->de_read_s = rxptr->size;
        dep->de_flags |= DEF_ACK_RECV;
        dep->de_flags &= NOT(DEF_READING);

        /* Return buffer to the idle pool */
        free_buff(dep, rxptr);
    }
    return;
}

/*
** Name:      void ell_send(dpeth_t *dep, int from_int, int pktsize)
** Function:  Send function. Called from main to transit a packet or
**           from interrupt handler when a new packet was queued.
*/
static void ell_send(dpeth_t * dep, int from_int, int pktsize)
{
    buff_t *txbuff;
    clock_t now;

    if (from_int == FALSE) {

```

```

    if ((txbuff = alloc_buff(dep, pktsize + sizeof(buff_t))) != NULL) {

        /* Fill transmit buffer from user area */
        txbuff->next = NULL;
        txbuff->size = pktsize;
        txbuff->client = dep->de_client;
        user2mem(dep, txbuff);
    } else
        panic(dep->de_name, "out of memory for Tx", NO_NUM);

} else if ((txbuff = dep->de_xmitq_head) != NULL) {

    /* Get first packet in queue */
    lock();
    if (dep->de_xmitq_tail == dep->de_xmitq_head)
        dep->de_xmitq_head = dep->de_xmitq_tail = NULL;
    else
        dep->de_xmitq_head = txbuff->next;
    unlock();
    pktsize = txbuff->size;

} else
    panic(dep->de_name, "should not be sending ", NO_NUM);

if ((dep->de_flags & DEF_XMIT_BUSY)) {
    if (from_int) panic(dep->de_name, "should not be sending ", NO_NUM);
    getuptime(&now);
    if ((now - dep->de_xmit_start) > 4) {
        /* Transmitter timed out */
        DEBUG printf("3c501: transmitter timed out ... \n");
        dep->de_stat.ets_sendErr += 1;
        dep->de_flags &= NOT(DEF_XMIT_BUSY);
        ell_reset(dep);
    }

    /* Queue packet */
    lock();
    /* Queue packet to receive queue */
    if (dep->de_xmitq_head == NULL)
        dep->de_xmitq_head = txbuff;
    else
        dep->de_xmitq_tail->next = txbuff;
    dep->de_xmitq_tail = txbuff;
    unlock();
} else {
    /* Save for retransmission */
    TxBuff = txbuff;
    dep->de_flags |= (DEF_XMIT_BUSY | DEF_ACK_SEND);

    /* Setup board for packet loading */
    lock();
    /* Buffer to processor */
    outb_ell(dep, EL1_CSR, ECSR_RIDE | ECSR_SYS);
    inb_ell(dep, EL1_RECV); /* Clears any spurious interrupt */
    inb_ell(dep, EL1_XMIT);
    outw_ell(dep, EL1_RECVPTR, 0); /* Clears RX packet area */

    /* Loads packet */
    outw_ell(dep, EL1_XMITPTR, (EL1_BFRSIZ - pktsize));
    outsb(dep->de_data_port, SELF, txbuff->buffer, pktsize);
    /* Starts transmitter */
    outw_ell(dep, EL1_XMITPTR, (EL1_BFRSIZ - pktsize));
    outb_ell(dep, EL1_CSR, ECSR_RIDE | ECSR_XMIT); /* There it goes... */
    unlock();

    getuptime(&dep->de_xmit_start);
    dep->de_flags &= NOT(DEF_SENDING);
}
return;
}

/*
** Name:      void ell_stop(dpeth_t *dep)
** Function:  Stops board and disable interrupts.
*/
static void ell_stop(dpeth_t * dep)

```



```

{
    int ix;

    DEBUG(printf("%s: stopping Etherlink ....\n", dep->de_name));
    for (ix = 0; ix < 8; ix += 1) /* Reset board */
        outb_ell(dep, EL1_CSR, ECSR_RESET);
    outb_ell(dep, EL1_CSR, ECSR_SYS);
    sys_irqdisable(&dep->de_hook);          /* Disable interrupt */
    return;
}

/*
** Name:          void ell_interrupt(dpeth_t *dep)
** Function:      Interrupt handler. Acknowledges transmit interrupts
**                or unloads receive buffer to memory queue.
*/
static void ell_interrupt(dpeth_t * dep)
{
    ul6_t csr, isr;
    int pktsize;
    buff_t *rxptra;

    csr = inb_ell(dep, EL1_CSR);
    if ((csr & ECSR_XMIT) && (dep->de_flags & DEF_XMIT_BUSY)) {

        /* Got a transmit interrupt */
        isr = inb_ell(dep, EL1_XMIT);
        if ((isr & (EXSR_16JAM | EXSR_UNDER | EXSR_JAM)) || !(isr & EXSR_IDLE)) {
            DEBUG(printf("3c501: got xmit interrupt (ASR=0x%02X XSR=0x%02X)\n", csr, isr));
            if (isr & EXSR_JAM) {
                /* Sending, packet got a collision */
                dep->de_stat.ets_collision += 1;
                /* Put pointer back to beginning of packet */
                outb_ell(dep, EL1_CSR, ECSR_RIDE | ECSR_SYS);
                outw_ell(dep, EL1_XMITPTR, (EL1_BFRSIZ - TxBuff->size));
                /* And retrigger transmission */
                outb_ell(dep, EL1_CSR, ECSR_RIDE | ECSR_XMIT);
                return;
            } else if ((isr & EXSR_16JAM) || !(isr & EXSR_IDLE)) {
                dep->de_stat.ets_sendErr += 1;
            } else if (isr & EXSR_UNDER) {
                dep->de_stat.ets_fifoUnder += 1;
            }
            DEBUG(printf("3c501: got xmit interrupt (0x%02X)\n", isr));
            ell_reset(dep);
        } else {
            /** if (inw_ell(dep, EL1_XMITPTR) == EL1_BFRSIZ) **/
            /* Packet transmitted successfully */
            dep->de_stat.ets_packetT += 1;
            dep->bytes_Tx += (long) (TxBuff->size);
            free_buff(dep, TxBuff);
            dep->de_flags &= NOT(DEF_XMIT_BUSY);
            if ((dep->de_flags & DEF_SENDING) && dep->de_xmitq_head) {
                /* Pending transmit request available in queue */
                ell_send(dep, TRUE, 0);
                if (dep->de_flags & (DEF_XMIT_BUSY | DEF_ACK_SEND))
                    return;
            }
        }
    } else if ((csr & (ECSR_RECV | ECSR_XMTBSY)) == (ECSR_RECV | ECSR_XMTBSY)) {

        /* Got a receive interrupt */
        isr = inb_ell(dep, EL1_RECV);
        pktsize = inw_ell(dep, EL1_RECVPTR);
        if ((isr & ERSR_ERROR) || (isr & ERSR_STALE)) {
            DEBUG(printf("Rx0 (ASR=0x%02X RSR=0x%02X size=%d)\n", csr, isr, pktsize));
            dep->de_stat.ets_recvErr += 1;
        } else if (pktsize < ETH_MIN_PACK_SIZE || pktsize > ETH_MAX_PACK_SIZE) {
            DEBUG(printf("Rx1 (ASR=0x%02X RSR=0x%02X size=%d)\n", csr, isr, pktsize));

```

```

        dep->de_stat.ets_recvErr += 1;

    } else if ((rxptring = alloc_buff(dep, pktsize + sizeof(buff_t))) == NULL) {
        /* Memory not available. Drop packet */
        dep->de_stat.ets_fifoOver += 1;

    } else if (isr & (ERSR_GOOD | ERSR_ANY)) {
        /* Got a good packet. Read it from buffer */
        outb_ell(dep, EL1_CSR, ECSR_RIDE | ECSR_SYS);
        outw_ell(dep, EL1_XMITPTR, 0);
        insb(dep->de_data_port, SELF, rxptring->buffer, pktsize);
        rxptring->next = NULL;
        rxptring->size = pktsize;
        dep->de_stat.ets_packetR += 1;
        dep->bytes_Rx += (long) pktsize;
        lock();
        /* Queue packet to receive queue */
        if (dep->de_recvq_head == NULL)
            dep->de_recvq_head = rxptring;
        else
            dep->de_recvq_tail->next = rxptring;
        dep->de_recvq_tail = rxptring;
        unlock();

        /* Reply to pending Receive requests, if any */
        ell_recv(dep, TRUE, 0);
    }
} else {
    /* Nasty condition, should never happen */
    DEBUG(
        printf("3c501: got interrupt with status 0x%02X\n"
            "    de_flags=0x%04X XSR=0x%02X RSR=0x%02X\n"
            "    xmit buffer = 0x%4X recv buffer = 0x%4X\n",
            csr, dep->de_flags,
            inb_ell(dep, EL1_RECV),
            inb_ell(dep, EL1_XMIT),
            inw_ell(dep, EL1_XMITPTR),
            inw_ell(dep, EL1_RECVPTR));
    );
    ell_reset(dep);
}

/* Move into receive mode */
outb_ell(dep, EL1_CSR, ECSR_RIDE | ECSR_RECV);
outw_ell(dep, EL1_RECVPTR, 0);
/* Be sure that interrupts are cleared */
inb_ell(dep, EL1_RECV);
inb_ell(dep, EL1_XMIT);
return;
}

/*
** Name:      void ell_init(dpeth_t *dep)
** Function:  Initializes board hardware and driver data structures.
*/
static void ell_init(dpeth_t * dep)
{
    int ix;

    dep->de_irq &= NOT(DEI_DEFAULT);      /* Strip the default flag. */
    dep->de_offset_page = 0;
    dep->de_data_port = dep->de_base_port + EL1_DATAPORT;

    ell_reset(dep);                      /* Reset and initialize board */

    /* Start receiver (default mode) */
    outw_ell(dep, EL1_RECVPTR, 0);
    outb_ell(dep, EL1_CSR, ECSR_RIDE | ECSR_RECV);

    /* Initializes buffer pool */
    init_buff(dep, NULL);
    ell_mode_init(dep);

    printf("%s: Etherlink (%s) at %X:%d - ",
        dep->de_name, "3c501", dep->de_base_port, dep->de_irq);
    for (ix = 0; ix < SA_ADDR_LEN; ix += 1)

```

```
    printf("%02X%c", (dep->de_address.ea_addr[ix] = StationAddress[ix]),
           ix < SA_ADDR_LEN - 1 ? ':' : '\n');

/* Device specific functions */
dep->de_rcvtf = ell_rcvf;
dep->de_sndtf = ell_snd;
dep->de_flagsf = ell_mode_init;
dep->de_resettf = ell_reset;
dep->de_getstatsf = ell_getstats;
dep->de_dumpstatsf = ell_dumpstats;
dep->de_interruptf = ell_interrupt;

return;                                /* Done */
}

/*
** Name:      int ell_probe(dpeth_t *dep)
** Function:  Checks for presence of the board.
*/
PUBLIC int ell_probe(dpeth_t * dep)
{
    int ix;

    for (ix = 0; ix < 8; ix += 1) /* Reset the board */
        outb_ell(dep, EL1_CSR, ECSR_RESET);
    outb_ell(dep, EL1_CSR, ECSR_SYS); /* Leaves buffer to system */

    /* Check station address */
    for (ix = 0; ix < SA_ADDR_LEN; ix += 1) {
        outw_ell(dep, EL1_XMITPTR, ix);
        StationAddress[ix] = inb_ell(dep, EL1_SAPROM);
    }
    if (StationAddress[0] != 0x02 || /* Etherlink Station address */
        StationAddress[1] != 0x60 || /* MUST be 02:60:8c:xx:xx:xx */
        StationAddress[2] != 0x8C)
        return FALSE; /* No Etherlink board at this address */

    dep->de_ramsize = 0; /* RAM size is meaningless */
    dep->de_linmem = 0L; /* Access is via I/O port */

    /* Device specific functions */
    dep->de_initf = ell_init;
    dep->de_stopf = ell_stop;

    return TRUE; /* Etherlink board found */
}

#endif                                /* ENABLE_3C501 */

/** 3c501.c **/
```

```

/*
** File:          3c501.h          Jan. 14, 1997
**
** Author:       Giovanni Falzoni <gfalzoni@inwind.it>
**
** Interface description for 3Com Etherlink boards
**
** $Log: 3c501.h,v $
** Revision 1.1  2005/06/29 10:16:46  beng
** Import of dpeth 3c501/3c509b/.. ethernet driver by
** Giovanni Falzoni <fgalzoni@inwind.it>.
**
** Revision 2.0  2005/06/26 16:16:46  lsodgf0
** Initial revision for Minix 3.0.6
**
** $Id: 3c501.h,v 1.1 2005/06/29 10:16:46 beng Exp $
*/

/* The various board command registers */
#define EL1_ADDRESS      0x00    /* Board station address, 6 bytes */
#define EL1_RECV         0x06    /* Board Receive Config/Status Reg. */
#define EL1_XMIT         0x07    /* Board Transmit Config/Status Reg. */
#define EL1_XMITPTR      0x08    /* Transmit buffer pointer (word access) */
#define EL1_RECVPTR      0x0A    /* Receive buffer pointer (word access) */
#define EL1_SAPROM        0x0C    /* Window on Station Addr prom */
#define EL1_CSR           0x0E    /* Board Command/Status Register */
#define EL1_DATAPORT     0x0F    /* Window on packet buffer (Data Port) */

/* Bits in EL1_RECV, interrupt enable on write, status when read */
#define ERSR_NONE        0x00    /* Match mode in bits 5-6 (wo) */
#define ERSR_ALL         0x40    /* Promiscuous receive (wo) */
#define ERSR_BROAD       0x80    /* Station address plus broadcast (wo) */
#define ERSR_MULTI       0x80    /* Station address plus multicast 0xC0 */
#define ERSR_STALE       0x80    /* Receive status previously read (ro) */
#define ERSR_GOOD        0x20    /* Well formed packets only (rw) */
#define ERSR_ANY         0x10    /* Any packet, even with errors (rw) */
#define ERSR_SHORT       0x08    /* Short frame (rw) */
#define ERSR_DRIBBLE     0x04    /* Dribble error (rw) */
#define ERSR_FCS         0x02    /* CRC error (rw) */
#define ERSR_OVER        0x01    /* Data overflow (rw) */

#define ERSR_ERROR       (ERSR_SHORT|ERSR_DRIBBLE|ERSR_FCS|ERSR_OVER)
#define ERSR_RMASK       (ERSR_GOOD|ERSR_ERROR)/*(ERSR_GOOD|ERSR_ANY|ERSR_ERROR)*/

/* Bits in EL1_XMIT, interrupt enable on write, status when read */
#define EXSR_IDLE        0x08    /* Transmit idle (send completed) */
#define EXSR_16JAM       0x04    /* Packet sending got 16 collisions */
#define EXSR_JAM         0x02    /* Packet sending got a collision */
#define EXSR_UNDER       0x01    /* Data underflow in sending */

/* Bits in EL1_CSR (Configuration Status Register) */
#define ECSR_RESET       0x80    /* Reset the controller (wo) */
#define ECSR_XMTBSY      0x80    /* Transmitter busy (ro) */
#define ECSR_RIDE        0x01    /* Request interrupt/DMA enable (rw) */
#define ECSR_DMA         0x20    /* DMA request (rw) */
#define ECSR_EDMA        0x10    /* DMA done (ro) */
#define ECSR_CRC         0x02    /* Causes CRC error on transmit (wo) */
#define ECSR_RCVBSY      0x01    /* Receive in progress (ro) */
#define ECSR_LOOP        (3<<2) /* 2 bit field in bits 2,3, loopback */
#define ECSR_RECV        (2<<2) /* Gives buffer to receiver (rw) */
#define ECSR_XMIT        (1<<2) /* Gives buffer to transmit (rw) */
#define ECSR_SYS         (0<<2) /* Gives buffer to processor (wo) */

#define EL1_BFRSIZ       2048    /* Number of bytes in board buffer */

#define inb_ell(dep,reg) (inb(dep->de_base_port+(reg)))
#define inw_ell(dep,reg) (inw(dep->de_base_port+(reg)))
#define outb_ell(dep,reg,data) (outb(dep->de_base_port+(reg),data))
#define outw_ell(dep,reg,data) (outw(dep->de_base_port+(reg),data))

/** 3c501.h */

```

```
/*
** File:          3c503.c          Dec. 20, 1996
**
** Author:       Giovanni Falzoni <gfalzoni@inwind.it>
**
** Driver for the Etherlink II boards. Works in shared memory mode.
** Programmed I/O could be used as well but would result in poor
** performances. This file contains only the board specific code,
** the rest is in 8390.c          Code specific for ISA bus only
**
** $Id: 3c503.c,v 1.3 2005/08/05 19:08:43 beng Exp $
*/

#include "drivers.h"
#include <net/gen/ether.h>
#include <net/gen/eth_io.h>
#include "dp.h"

#if (ENABLE_3C503 == 1)

#include "8390.h"
#include "3c503.h"

/*
** Name:          void el2_init(dpeth_t *dep);
** Function:      Initialize hardware and data structures.
*/
static void el2_init(dpeth_t * dep)
{
    int ix, irq;
    int sendq_nr;
    int cntr;

    /* Map the address PROM to lower I/O address range */
    cntr = inb_el2(dep, EL2_CNTR);
    outb_el2(dep, EL2_CNTR, cntr | ECNTR_SAPROM);

    /* Read station address from PROM */
    for (ix = EL2_EA0; ix <= EL2_EA5; ix += 1)
        dep->de_address.ea_addr[ix] = inb_el2(dep, ix);

    /* Map the 8390 back to lower I/O address range */
    outb_el2(dep, EL2_CNTR, cntr);

    /* Enable memory, but turn off interrupts until we are ready */
    outb_el2(dep, EL2_CFGR, ECFGR_IRQOFF);

    dep->de_data_port = dep->de_dp8390_port = dep->de_base_port;
    dep->de_prog_IO = FALSE;          /* Programmed I/O not yet available */

    /* Check width of data bus */
    outb_el2(dep, DP_CR, CR_PS_P0 | CR_NO_DMA | CR_STP);
    outb_el2(dep, DP_DCR, 0);
    outb_el2(dep, DP_CR, CR_PS_P2 | CR_NO_DMA | CR_STP);
    dep->de_16bit = (inb_el2(dep, DP_DCR) & DCR_WTS) != 0;
    outb_el2(dep, DP_CR, CR_PS_P0 | CR_NO_DMA | CR_STP);

    /* Allocate one send buffer (1.5kb) per 8kb of on board memory. */
    /* Only 8kb of 3c503/16 boards are used to avoid specific routines */
    sendq_nr = dep->de_ramsize / 0x2000;
    if (sendq_nr < 1)
        sendq_nr = 1;
    else if (sendq_nr > SENDQ_NR)
        sendq_nr = SENDQ_NR;

    dep->de_sendq_nr = sendq_nr;
    for (ix = 0; ix < sendq_nr; ix++)
        dep->de_sendq[ix].sq_sendpage = (ix * SENDQ_PAGES) + EL2_SM_START_PG;

    dep->de_startpage = (ix * SENDQ_PAGES) + EL2_SM_START_PG;
    dep->de_stoppage = EL2_SM_STOP_PG;

    outb_el2(dep, EL2_STARTPG, dep->de_startpage);
    outb_el2(dep, EL2_STOPPG, dep->de_stoppage);
}
```

```

/* Point the vector pointer registers somewhere ?harmless?. */
outb_el2(dep, EL2_VP2, 0xFF); /* Point at the ROM restart location */
outb_el2(dep, EL2_VP1, 0xFF); /* 0xFFFF:0000 (from original sources) */
outb_el2(dep, EL2_VP0, 0x00); /* - What for protected mode? */

/* Set interrupt level for 3c503 */
irq = (dep->de_irq & ~DEI_DEFAULT); /* Strip the default flag. */
if (irq == 9) irq = 2;
if (irq < 2 || irq > 5) panic(dep->de_name, "bad 3c503 irq configuration", irq);
outb_el2(dep, EL2_IDCFG, (0x04 << irq));

outb_el2(dep, EL2_DRQCNT, 0x08); /* Set burst size to 8 */
outb_el2(dep, EL2_DMAAH, EL2_SM_START_PG); /* Put start of TX */
outb_el2(dep, EL2_DMAAL, 0x00); /* buffer in the GA DMA reg */

outb_el2(dep, EL2_CFGR, ECFGR_NORM); /* Enable shared memory */

ns_init(dep); /* Initialize DP controller */

printf("%s: Etherlink II%s (%s) at %X:%d:%05IX - ",
        dep->de_name, dep->de_16bit ? "/16" : "", "3c503",
        dep->de_base_port, dep->de_irq,
        dep->de_linmem + dep->de_offset_page);
for (ix = 0; ix < SA_ADDR_LEN; ix += 1)
    printf("%02X%c", dep->de_address.ea_addr[ix],
           ix < SA_ADDR_LEN - 1 ? ':' : '\n');
return;
}

/*
** Name:      void el2_stop(dpeth_t *dep);
** Function:  Stops board by disabling interrupts.
*/
static void el2_stop(dpeth_t * dep)
{
    outb_el2(dep, EL2_CFGR, ECFGR_IRQOFF);
    sys_irqdisable(&dep->de_hook); /* disable interrupts */
    return;
}

/*
** Name:      void el2_probe(dpeth_t *dep);
** Function:  Probe for the presence of an EtherLink II card.
**           Initialize memory addressing if card detected.
*/
int el2_probe(dpeth_t * dep)
{
    int iobase, membase;
    int thin;

    /* Thin ethernet or AUI? */
    thin = (dep->de_linmem & 1) ? ECNTR_AUI : ECNTR_THIN;

    /* Location registers should have 1 bit set */
    if (!(iobase = inb_el2(dep, EL2_IOBASE))) return FALSE;
    if (!(membase = inb_el2(dep, EL2_MEMBASE)) & 0xF0) return FALSE;
    if ((iobase & (iobase - 1)) || (membase & (membase - 1))) return FALSE;

    /* Resets board */
    outb_el2(dep, EL2_CNTR, ECNTR_RESET | thin);
    milli_delay(1);
    outb_el2(dep, EL2_CNTR, thin);
    milli_delay(5);

    /* Map the address PROM to lower I/O address range */
    outb_el2(dep, EL2_CNTR, ECNTR_SAPROM | thin);
    if (inb_el2(dep, EL2_EA0) != 0x02 || /* Etherlink II Station address */
        inb_el2(dep, EL2_EA1) != 0x60 || /* MUST be 02:60:8c:xx:xx:xx */
        inb_el2(dep, EL2_EA2) != 0x8C)
        return FALSE; /* No Etherlink board at this address */

    /* Map the 8390 back to lower I/O address range */

```

```
outb_el2(dep, EL2_CNTR, thin);

/* Setup shared memory addressing for 3c503 */
dep->de_linmem = ((membase & 0xC0) ? EL2_BASE_0D8000 : EL2_BASE_0C8000) +
    ((membase & 0xA0) ? (EL2_BASE_0CC000 - EL2_BASE_0C8000) : 0x0000);

/* Shared memory starts at 0x2000 (8kb window) */
dep->de_offset_page = (EL2_SM_START_PG * DP_PAGESIZE);
dep->de_linmem -= dep->de_offset_page;
dep->de_ramsize = (EL2_SM_STOP_PG - EL2_SM_START_PG) * DP_PAGESIZE;

/* Board initialization and stop functions */
dep->de_initf = el2_init;
dep->de_stopf = el2_stop;
return TRUE;
}
#endif                                /* ENABLE_3C503 */

/** 3c503.c **/
```

```
/*
** File:          3c503.h          Dec. 20, 1996
**
** Author:       Giovanni Falzoni <gfalzoni@inwind.it>
**
** Interface description for 3Com Etherlink II boards
**
** $Log: 3c503.h,v $
** Revision 1.1  2005/06/29 10:16:46  beng
** Import of dpeth 3c501/3c509b/.. ethernet driver by
** Giovanni Falzoni <fgalzoni@inwind.it>.
**
** Revision 2.0  2005/06/26 16:16:46  lsodgf0
** Initial revision for Minix 3.0.6
**
** $Id: 3c503.h,v 1.1 2005/06/29 10:16:46  beng Exp $
*/

#define EL2_MEMTEST      0          /* Set to 1 for on board memory test */

#define EL2_GA           0x0400    /* Offset of registers in Gate Array */

/* EtherLink II card */

#define EL2_STARTPG      (EL2_GA+0x00) /* Start page matching DP_PSTARTPG */
#define EL2_STOPPG       (EL2_GA+0x01) /* Stop page matching DP_PSTOPPG */
#define EL2_DRQCNT       (EL2_GA+0x02) /* DMA burst count */
#define EL2_IOBASE       (EL2_GA+0x03) /* I/O base jumpers (bit coded) */
#define EL2_MEMBASE      (EL2_GA+0x04) /* Memory base jumpers (bit coded) */
#define EL2_CFGR         (EL2_GA+0x05) /* Configuration Register for GA */
#define EL2_CNTR         (EL2_GA+0x06) /* Control(write) and status(read) */
#define EL2_STATUS       (EL2_GA+0x07)
#define EL2_IDCFG        (EL2_GA+0x08) /* Interrupt/DMA configuration reg */
#define EL2_DMAAH        (EL2_GA+0x09) /* DMA address register (High byte) */
#define EL2_DMAAL        (EL2_GA+0x0A) /* DMA address register (Low byte) */
#define EL2_VP2          (EL2_GA+0x0B) /* Vector pointer - set to */
#define EL2_VP1          (EL2_GA+0x0C) /* reset address (0xFFFF:0) */
#define EL2_VP0          (EL2_GA+0x0D) /* */
#define EL2_FIFOH        (EL2_GA+0x0E) /* FIFO for progr. I/O (High byte) */
#define EL2_FIFOL        (EL2_GA+0x0F) /* FIFO for progr. I/O (Low byte) */

#define EL2_EA0          0x00      /* Most significant byte of ethernet address */
#define EL2_EA1          0x01
#define EL2_EA2          0x02
#define EL2_EA3          0x03
#define EL2_EA4          0x04
#define EL2_EA5          0x05      /* Least significant byte of ethernet address */

/* Bits in EL2_CNTR register */
#define ECNTR_RESET      0x01      /* Software Reset */
#define ECNTR_THIN       0x02      /* Onboard transceiver enable */
#define ECNTR_AUI        0x00      /* Onboard transceiver disable */
#define ECNTR_SAPROM     0x04      /* Map the station address prom */

/* Bits in EL2_CFGR register */
#define ECFGR_NORM       0x49      /* Enable 8k shared memory, no DMA, TC int */
#define ECFGR_IRQOFF     0xC9      /* As above, disable 8390 IRQ */

/* Shared memory management parameters */
#define EL2_SM_START_PG  0x20      /* First page of TX buffer */
#define EL2_SM_STOP_PG   0x40      /* Last page +1 of RX ring */

/* Physical addresses where an Etherlink board can be configured */
#define EL2_BASE_0C8000  0x0C8000
#define EL2_BASE_0CC000  0x0CC000
#define EL2_BASE_0D8000  0x0D8000
#define EL2_BASE_0DC000  0x0DC000

#define inb_el2(dep,reg) (inb((dep)->de_base_port+(reg)))
#define outb_el2(dep,reg,data) (outb((dep)->de_base_port+(reg),(data)))

/** 3c503.h **/
```



```
/*
** File:          3c509.c          Jun. 01, 2000
**
** Author:       Giovanni Falzoni <gfalzoni@inwind.it>
**
** This file contains specific implementation of the ethernet
** device driver for 3Com Etherlink III (3c509) boards.
** NOTE: The board has to be setup to disable PnP and to assign
**       I/O base and IRQ. The driver is for ISA bus only
**
** $Id: 3c509.c,v 1.3 2005/08/05 19:08:43 beng Exp $
*/

#include "drivers.h"
#include <minix/com.h>
#include <net/hton.h>
#include <net/gen/ether.h>
#include <net/gen/eth_io.h>

#include "dp.h"

#if (ENABLE_3C509 == 1)

#include "3c509.h"

static const char *const IfNamesMsg[] = {
    "10BaseT", "AUI", "unknown", "BNC",
};

/*
** Name:          void el3_update_stats(dpeth_t *dep)
** Function:      Reads statistic counters from board
**                and updates local counters.
*/
static void el3_update_stats(dpeth_t * dep)
{
    /* Disables statistics while reading and switches to the correct window */
    outw_el3(dep, REG_CmdStatus, CMD_StatsDisable);
    SetWindow(WNO_Statistics);

    /* Reads everything, adding values to the local counters */
    dep->de_stat.ets_sendErr += inb_el3(dep, REG_TxCARRIERLost); /* Reg. 00 */
    dep->de_stat.ets_sendErr += inb_el3(dep, REG_TxNoCD);          /* Reg. 01 */
    dep->de_stat.ets_collision += inb_el3(dep, REG_TxMultColl);    /* Reg. 02 */
    dep->de_stat.ets_collision += inb_el3(dep, REG_TxSingleColl);  /* Reg. 03 */
    dep->de_stat.ets_collision += inb_el3(dep, REG_TxLate);        /* Reg. 04 */
    dep->de_stat.ets_recvErr += inb_el3(dep, REG_RxDiscarded);     /* Reg. 05 */
    dep->de_stat.ets_packetT += inb_el3(dep, REG_TxFrames);        /* Reg. 06 */
    dep->de_stat.ets_packetR += inb_el3(dep, REG_RxFrames);        /* Reg. 07 */
    dep->de_stat.ets_transDef += inb_el3(dep, REG_TxDefer);         /* Reg. 08 */
    dep->bytes_Rx += (unsigned) inw_el3(dep, REG_RxBytes);          /* Reg. 10 */
    dep->bytes_Tx += (unsigned) inw_el3(dep, REG_TxBytes);          /* Reg. 12 */

    /* Goes back to operating window and enables statistics */
    SetWindow(WNO_Operating);
    outw_el3(dep, REG_CmdStatus, CMD_StatsEnable);

    return;
}

/*
** Name:          void el3_getstats(dpeth_t *dep)
** Function:      Reads statistics counters from board.
*/
static void el3_getstats(dpeth_t * dep)
{
    lock();
    el3_update_stats(dep);
    unlock();
    return;
}
```

```
/*
** Name:      void el3_dodump(dpeth_t *dep)
** Function:   Dumps counter on screen (support for console display).
*/
static void el3_dodump(dpeth_t * dep)
{
    el3_getstats(dep);
    return;
}

/*
** Name:      void el3_rx_mode(dpeth_t *dep)
** Function:   Initializes receiver mode
*/
static void el3_rx_mode(dpeth_t * dep)
{
    dep->de_rcv_mode = FilterIndividual;
    if (dep->de_flags & DEF_BROAD) dep->de_rcv_mode |= FilterBroadcast;
    if (dep->de_flags & DEF_MULTI) dep->de_rcv_mode |= FilterMulticast;
    if (dep->de_flags & DEF_PROMISC) dep->de_rcv_mode |= FilterPromiscuous;

    outw_el3(dep, REG_CmdStatus, CMD_RxReset);
    outw_el3(dep, REG_CmdStatus, CMD_SetRxFilter | dep->de_rcv_mode);
    outw_el3(dep, REG_CmdStatus, CMD_RxEnable);

    return;
}

/*
** Name:      void el3_reset(dpeth_t *dep)
** Function:   Reset function specific for Etherlink hardware.
*/
static void el3_reset(dpeth_t * dep)
{
    return;                /* Done */
}

/*
** Name:      void el3_write_fifo(dpeth_t * dep, int pktsize);
** Function:   Writes a packet from user area to board.
** Remark:     Writing a word/dword at a time may result faster
**             but is a lot more complicated. Let's go simpler way.
*/
static void el3_write_fifo(dpeth_t * dep, int pktsize)
{
    phys_bytes phys_user;
    int bytes, ix = 0;
    iovec_dat_t *iovp = &dep->de_write_iovec;
    int padding = pktsize;

    do {
        /* Writes chunks of packet from user buffers */

        bytes = iovp->iod_iovec[ix].iov_size;    /* Size of buffer */
        if (bytes > pktsize) bytes = pktsize;
        /* Writes from user buffer to Tx FIFO */
        outsb(dep->de_data_port, iovp->iod_proc_nr,
            (void*)(iovp->iod_iovec[ix].iov_addr), bytes);
        if (++ix >= IOVEC_NR) { /* Next buffer of IO vector */
            dp_next_iovec(iovp);
            ix = 0;
        }
        /* Till packet done */
    } while ((pktsize -= bytes) > 0);
    while ((padding++ % sizeof(long)) != 0) outb(dep->de_data_port, 0x00);
    return;
}

/*
** Name:      void el3_rcv(dpeth_t *dep, int fromint, int size)
** Function:   Receive function. Called from interrupt handler or
**             from main to unload rcv. buffer (packet to client)
*/
```

```

*/
static void el3_rcv(dpeth_t *dep, int fromint, int size)
{
    buff_t *rxp;

    while ((dep->de_flags & DEF_READING) && (rxp = dep->de_rcvq_head)) {
        lock();
        /* Remove buffer from queue */
        if (dep->de_rcvq_tail == dep->de_rcvq_head)
            dep->de_rcvq_head = dep->de_rcvq_tail = NULL;
        else
            dep->de_rcvq_head = rxp->next;
        unlock();

        /* Copy buffer to user area and free it */
        mem2user(dep, rxp);

        dep->de_read_s = rxp->size;
        dep->de_flags |= DEF_ACK_RECV;
        dep->de_flags &= NOT(DEF_READING);

        /* Return buffer to the idle pool */
        free_buff(dep, rxp);
    }
    return;
}

/*
** Name:      void el3_rx_complete(dpeth_t * dep);
** Function:  Upon receiving a packet, provides status checks
**            and if packet is OK copies it to local buffer.
*/
static void el3_rx_complete(dpeth_t * dep)
{
    short int RxStatus;
    int pktsize;
    buff_t *rxp;

    RxStatus = inw_el3(dep, REG_RxStatus);
    pktsize = RxStatus & RXS_Length; /* Mask off packet length */

    if (RxStatus & RXS_Error) {
        /* First checks for receiving errors */
        RxStatus &= RXS_ErrType;
        switch (RxStatus) { /* Bad packet (see error type) */
            case RXS_Dribble:
            case RXS_Oversize:
            case RXS_Runt:
                dep->de_stat.ets_rcvErr += 1; break;
            case RXS_Overrun:
                dep->de_stat.ets_OVW += 1; break;
            case RXS_Framing:
                dep->de_stat.ets_frameAll += 1; break;
            case RXS_CRC:
                dep->de_stat.ets_CRCerr += 1; break;
        }
    }
    else if ((rxp = alloc_buff(dep, pktsize + sizeof(buff_t))) == NULL) {
        /* Memory not available. Drop packet */
        dep->de_stat.ets_fifoOver += 1;
    }
    else {
        /* Good packet. Read it from FIFO */
        insb(dep->de_data_port, SELF, rxp->buffer, pktsize);
        rxp->next = NULL;
        rxp->size = pktsize;

        lock(); /* Queue packet to receive queue */
        if (dep->de_rcvq_head == NULL)
            dep->de_rcvq_head = rxp;
        else
            dep->de_rcvq_tail->next = rxp;
        dep->de_rcvq_tail = rxp;
        unlock();

        /* Reply to pending Receive requests, if any */
        el3_rcv(dep, TRUE, pktsize);
    }
}

```

```
}

/* Discard top packet from queue */
outw_el3(dep, REG_CmdStatus, CMD_RxDiscard);

return;
}

/*
** Name:      void el3_send(dpeth_t *dep, int count)
** Function:  Send function. Called from main to transit a packet or
**           from interrupt handler when Tx FIFO gets available.
*/
static void el3_send(dpeth_t * dep, int from_int, int count)
{
    clock_t now;
    int ix;
    short int TxStatus;

    getuptime(&now);
    if ((dep->de_flags & DEF_XMIT_BUSY) &&
        (now - dep->de_xmit_start) > 4) {

        DEBUG(sprintf("3c509: Transmitter timed out. Resetting ....\n"));
        dep->de_stat.ets_sendErr += 1;
        /* Resets and restarts the transmitter */
        outw_el3(dep, REG_CmdStatus, CMD_TxReset);
        outw_el3(dep, REG_CmdStatus, CMD_TxEnable);
        dep->de_flags &= NOT(DEF_XMIT_BUSY);
    }
    if (!(dep->de_flags & DEF_XMIT_BUSY)) {

        /* Writes Transmitter preamble 1st Word (packet len, no ints) */
        outw_el3(dep, REG_TxFIFO, count);
        /* Writes Transmitter preamble 2nd Word (all zero) */
        outw_el3(dep, REG_TxFIFO, 0);
        /* Writes packet */
        el3_write_fifo(dep, count);

        getuptime(&dep->de_xmit_start);
        dep->de_flags |= (DEF_XMIT_BUSY | DEF_ACK_SEND);
        if (inw_el3(dep, REG_TxFree) > ETH_MAX_PACK_SIZE) {
            /* Tx has enough room for a packet of maximum size */
            dep->de_flags &= NOT(DEF_XMIT_BUSY | DEF_SENDING);
        } else {
            /* Interrupt driver when enough room is available */
            outw_el3(dep, REG_CmdStatus, CMD_SetTxAvailable | ETH_MAX_PACK_SIZE);
            dep->de_flags &= NOT(DEF_SENDING);
        }

        /* Pops Tx status stack */
        for (ix = 4; --ix && (TxStatus = inb_el3(dep, REG_TxStatus)) > 0;) {
            if (TxStatus & 0x38) dep->de_stat.ets_sendErr += 1;
            if (TxStatus & 0x30)
                outw_el3(dep, REG_CmdStatus, CMD_TxReset);
            if (TxStatus & 0x3C)
                outw_el3(dep, REG_CmdStatus, CMD_TxEnable);
            outb_el3(dep, REG_TxStatus, 0);
        }
    }
    return;
}

/*
** Name:      void el3_close(dpeth_t *dep)
** Function:  Stops board and makes it ready to shut down.
*/
static void el3_close(dpeth_t * dep)
{
    /* Disables statistics, Receiver and Transmitter */
    outw_el3(dep, REG_CmdStatus, CMD_StatsDisable);
    outw_el3(dep, REG_CmdStatus, CMD_RxDisable);
    outw_el3(dep, REG_CmdStatus, CMD_TxDisable);
}
```

```

if (dep->de_if_port == BNC_XCVR) {
    outw_el3(dep, REG_CmdStatus, CMD_StopIntXcvr);
    /* milli_delay(5); */
} else if (dep->de_if_port == TP_XCVR) {
    SetWindow(WNO_Diagnostics);
    outw_el3(dep, REG_MediaStatus, inw_el3(dep, REG_MediaStatus) &
        NOT((MediaLBeatEnable | MediaJabberEnable)));
    /* milli_delay(5); */
}
DEBUG(printf("%s: stopping Etherlink ... \n", dep->de_name));
/* Issues a global reset
outw_el3(dep, REG_CmdStatus, CMD_GlobalReset); */
sys_irqdisable(&dep->de_hook); /* Disable interrupt */

return;
}

/*
** Name:          void el3_interrupt(dpeth_t *dep)
** Function:      Interrupt handler. Acknowledges transmit interrupts
**               or unloads receive buffer to memory queue.
*/
static void el3_interrupt(dpeth_t * dep)
{
    int loop;
    unsigned short isr;

    for (loop = 5; loop > 0 && ((isr = inw_el3(dep, REG_CmdStatus)) &
        (INT_Latch | INT_RxComplete | INT_UpdateStats)); loop -= 1) {

        if (isr & INT_RxComplete) /* Got a new packet */
            el3_rx_complete(dep);

        if (isr & INT_TxAvailable) { /* Tx has room for big packets */
            DEBUG(printf("3c509: got Tx interrupt, Status=0x%04x\n", isr));
            dep->de_flags &= NOT(DEF_XMIT_BUSY);
            outw_el3(dep, REG_CmdStatus, CMD_Acknowledge | INT_TxAvailable);
            if (dep->de_flags & DEF_SENDING) /* Send pending */
                el3_send(dep, TRUE, dep->de_send_s);
        }
        if (isr & (INT_AdapterFail | INT_RxEarly | INT_UpdateStats)) {

            if (isr & INT_UpdateStats) /* Empties statistics */
                el3_getstats(dep);

            if (isr & INT_RxEarly) /* Not really used. Do nothing */
                outw_el3(dep, REG_CmdStatus, CMD_Acknowledge | (INT_RxEarly));

            if (isr & INT_AdapterFail) {
                /* Adapter error. Reset and re-enable receiver */
                DEBUG(printf("3c509: got Rx fail interrupt, Status=0x%04x\n", isr));
                el3_rx_mode(dep);
                outw_el3(dep, REG_CmdStatus, CMD_Acknowledge | INT_AdapterFail);
            }
        }

        /* Acknowledge interrupt */
        outw_el3(dep, REG_CmdStatus, CMD_Acknowledge | (INT_Latch | INT_Requested));
    }
    return;
}

/*
** Name:          unsigned el3_read_eeprom(port_t port, unsigned address);
** Function:      Reads the EEPROM at specified address
*/
static unsigned el3_read_eeprom(port_t port, unsigned address)
{
    unsigned int result;
    int bit;

    address |= EL3_READ_EEPROM;

```

```
    outb(port, address);
    milli_delay(5);                /* Allows EEPROM reads */
    for (result = 0, bit = 16; bit > 0; bit -= 1) {
        result = (result << 1) | (inb(port) & 0x0001);
    }
    return result;
}

/*
** Name:          void el3_read_StationAddress(dpeth_t *dep)
** Function:      Reads station address from board
*/
static void el3_read_StationAddress(dpeth_t * dep)
{
    unsigned int ix, rc;

    for (ix = EE_3COM_NODE_ADDR; ix < SA_ADDR_LEN;) {
        /* Accesses with word No. */
        rc = el3_read_eeprom(dep->de_id_port, ix / 2);
        /* Swaps bytes of word */
        dep->de_address.ea_addr[ix++] = (rc >> 8) & 0xFF;
        dep->de_address.ea_addr[ix++] = rc & 0xFF;
    }
    return;
}

/*
** Name:          void el3_open(dpeth_t *dep)
** Function:      Initializes board hardware and driver data structures.
*/
static void el3_open(dpeth_t * dep)
{
    unsigned int AddrCfgReg, ResCfgReg;
    unsigned int ix;

    el3_read_StationAddress(dep); /* Get ethernet address */

    /* Get address and resource configurations */
    AddrCfgReg = el3_read_eeprom(dep->de_id_port, EE_ADDR_CFG);
    ResCfgReg = el3_read_eeprom(dep->de_id_port, EE_RESOURCE_CFG);
    outb(dep->de_id_port, EL3_ACTIVATE); /* Activate the board */

    /* Gets xcvr configuration */
    dep->de_if_port = AddrCfgReg & EL3_CONFIG_XCVR_MASK;

    AddrCfgReg = ((AddrCfgReg & EL3_CONFIG_IOBASE_MASK) << 4) + EL3_IO_BASE_ADDR;
    if (AddrCfgReg != dep->de_base_port)
        panic(dep->de_name, "Bad I/O port for Etherlink board", NO_NUM);

    ResCfgReg >>= 12;
    dep->de_irq &= NOT(DEI_DEFAULT); /* Strips the default flag */
    if (ResCfgReg != dep->de_irq) panic(dep->de_name, "Bad IRQ for Etherlink board", NO_NUM);

    SetWindow(WNO_Setup);

    /* Reset transmitter and receiver */
    outw_el3(dep, REG_CmdStatus, CMD_TxReset);
    outw_el3(dep, REG_CmdStatus, CMD_RxReset);

    /* Enable the adapter */
    outb_el3(dep, REG_CfgControl, EL3_EnableAdapter);
    /* Disable Status bits */
    outw_el3(dep, REG_CmdStatus, CMD_SetStatusEnab + 0x00);

    /* Set "my own" address */
    SetWindow(WNO_StationAddress);
    for (ix = 0; ix < 6; ix += 1)
        outb_el3(dep, REG_SA0_1 + ix, dep->de_address.ea_addr[ix]);

    /* Start Transceivers as required */
    if (dep->de_if_port == BNC_XCVR) {
        /* Start internal transceiver for Coaxial cable */
        outw_el3(dep, REG_CmdStatus, CMD_StartIntXcvr);
        milli_delay(5);
    }
}
```

```

} else if (dep->de_if_port == TP_XCVR) {
    /* Start internal transceiver for Twisted pair cable */
    SetWindow(WNO_Diagnostics);
    outw_el3(dep, REG_MediaStatus,
              inw_el3(dep, REG_MediaStatus) | (MediaLBeatEnable | MediaJabberEnable));
}

/* Switch to the statistic window, and clear counts (by reading) */
SetWindow(WNO_Statistics);
for (ix = REG_TxCarrierLost; ix <= REG_TxDefer; ix += 1) inb_el3(dep, ix);
inw_el3(dep, REG_RxBytes);
inw_el3(dep, REG_TxBytes);

/* Switch to operating window for normal use */
SetWindow(WNO_Operating);

/* Receive individual address & broadcast. (Mofified later by rx_mode) */
outw_el3(dep, REG_CmdStatus, CMD_SetRxFilter |
          (FilterIndividual | FilterBroadcast));

/* Turn on statistics */
outw_el3(dep, REG_CmdStatus, CMD_StatsEnable);

/* Enable transmitter and receiver */
outw_el3(dep, REG_CmdStatus, CMD_TxEnable);
outw_el3(dep, REG_CmdStatus, CMD_RxEnable);

/* Enable all the status bits */
outw_el3(dep, REG_CmdStatus, CMD_SetStatusEnab | 0xFF);

/* Acknowledge all interrupts to clear adapter. Enable interrupts */
outw_el3(dep, REG_CmdStatus, CMD_Acknowledge | 0xFF);
outw_el3(dep, REG_CmdStatus, CMD_SetIntMask |
          (INT_Latch | INT_TxAvailable | INT_RxComplete | INT_UpdateStats));

/* Ready to operate, sets the environment for eth_task */
dep->de_data_port = dep->de_base_port;
/* Allocates Rx/Tx buffers */
init_buff(dep, NULL);

/* Device specific functions */
dep->de_rcv = el3_rcv;
dep->de_snd = el3_snd;
dep->de_flags = el3_rx_mode;
dep->de_reset = el3_reset;
dep->de_getstats = el3_getstats;
dep->de_dumpstats = el3_dodump;
dep->de_interrupt = el3_interrupt;

printf("%s: Etherlink III (%s) at %X:%d, %s port - ",
        dep->de_name, "3c509", dep->de_base_port, dep->de_irq,
        IfNamesMsg[dep->de_if_port >> 14]);
for (ix = 0; ix < SA_ADDR_LEN; ix += 1)
    printf("%02X%c", dep->de_address.ea_addr[ix],
           ix < SA_ADDR_LEN - 1 ? ':' : '\n');

return; /* Done */
}

/*
** Name:      unsigned int el3_checksum(port_t port);
** Function:  Reads EEPROM and computes checksum.
*/
static unsigned short el3_checksum(port_t port)
{
    unsigned short rc, checksum, address;
    unsigned char lo, hi;

    for (checksum = address = 0; address < 15; address += 1) {
        rc = el3_read_eeprom(port, address);
        lo = rc & 0xFF;
        hi = (rc >> 8) & 0xFF;
        if ((address == EE_PROD_ID && (rc & EE_PROD_ID_MASK) != EL3_PRODUCT_ID) ||

```

```

        (address == EE_3COM_CODE && rc != EL3_3COM_CODE))
            return address;
    if (address == EE_ADDR_CFG ||
        address == EE_RESOURCE_CFG ||
        address == EE_SW_CONFIG_INFO) {
        lo ^= hi;
        hi = 0;
    } else {
        hi ^= lo;
        lo = 0;
    }
    rc = ((unsigned) hi << 8) + lo;
    checksum ^= rc;
}
rc = el3_read_eeprom(port, address);
return(checksum ^= rc);    /* If OK checksum is 0 */
}

/*
** Name:      void el3_write_id(port_t port);
** Function:  Writes the ID sequence to the board.
*/
static void el3_write_id(port_t port)
{
    int ix, pattern;

    outb(port, 0);    /* Selects the ID port */
    outb(port, 0);    /* Resets hardware pattern generator */
    for (pattern = ix = 0x00FF; ix > 0; ix -= 1) {
        outb(port, pattern);
        pattern <= 1;
        pattern = (pattern & 0x0100) ? pattern ^ 0xCF : pattern;
    }
    return;
}

/*
** Name:      int el3_probe(dpeth_t *dep)
** Function:  Checks for presence of the board.
*/
PUBLIC int el3_probe(dpeth_t * dep)
{
    port_t id_port;

    /* Don't ask me what is this for !! */
    outb(0x0279, 0x02);    /* Select PnP config control register. */
    outb(0x0A79, 0x02);    /* Return to WaitForKey state. */
    /* Tests I/O ports in the 0x1xF range for a valid ID port */
    for (id_port = 0x110; id_port < 0x200; id_port += 0x10) {
        outb(id_port, 0x00);
        outb(id_port, 0xFF);
        if (inb(id_port) & 0x01) break;
    }
    if (id_port == 0x200) return 0;    /* No board responding */

    el3_write_id(id_port);
    outb(id_port, EL3_ID_GLOBAL_RESET);    /* Reset the board */
    milli_delay(5);    /* Technical reference says 162 micro sec. */
    el3_write_id(id_port);
    outb(id_port, EL3_SET_TAG_REGISTER);
    milli_delay(5);

    dep->de_id_port = id_port;    /* Stores ID port No. */
    dep->de_ramsize = 0;    /* RAM size is meaningless */
    dep->de_offset_page = 0;
    dep->de_linmem = 0L;    /* Access is via I/O port */

    /* Device specific functions */
    dep->de_initf = el3_open;
    dep->de_stopf = el3_close;

    return(el3_checksum(id_port) == 0);    /* Etherlink board found/not found */
}

```



```
#endif                                /* ENABLE_3C509 */  
  
/** 3c509.c */
```

```
/*
** File:          3c509.h          Jun. 01, 2000
**
** Author:       Giovanni Falzoni <gfalzoni@inwind.it>
**
** Interface description for 3Com Etherlink III board.
**
** $Log: 3c509.h,v $
** Revision 1.1 2005/06/29 10:16:46 beng
** Import of dpeth 3c501/3c509b/.. ethernet driver by
** Giovanni Falzoni <fgalzoni@inwind.it>.
**
** Revision 2.0 2005/06/26 16:16:46 lsodgf0
** Initial revision for Minix 3.0.6
**
** $Id: 3c509.h,v 1.1 2005/06/29 10:16:46 beng Exp $
*/

/* Command codes */
#define CMD_GlobalReset      0x0000 /* resets adapter (power up status) */
#define CMD_SelectWindow    (1<<11) /* select register window */
#define CMD_StartIntXcvr    (2<<11) /* start internal transceiver */
#define CMD_RxDisable       (3<<11) /* rx disable */
#define CMD_RxEnable        (4<<11) /* rx enable */
#define CMD_RxReset         (5<<11) /* rx reset */
#define CMD_RxDiscard       (8<<11) /* rx discard top packet */
#define CMD_TxEnable        (9<<11) /* tx enable */
#define CMD_TxDisable       (10<<11) /* tx disable */
#define CMD_TxReset         (11<<11) /* tx reset */
#define CMD_Acknowledge     (13<<11) /* acknowledge interrupt */
#define CMD_SetIntMask      (14<<11) /* set interrupt mask */
#define CMD_SetStatusEnab   (15<<11) /* set read zero mask */
#define CMD_SetRxFilter     (16<<11) /* set rx filter */
#define CMD_SetTxAvailable  (18<<11) /* set tx available threshold */
#define CMD_StatsEnable     (21<<11) /* statistics enable */
#define CMD_StatsDisable    (22<<11) /* statistics disable */
#define CMD_StopIntXcvr     (23<<11) /* start internal transceiver */

/* Status register bits (INT for interrupt sources, ST for the rest) */
#define INT_Latch           0x0001 /* interrupt latch */
#define INT_AdapterFail     0x0002 /* adapter failure */
#define INT_TxComplete      0x0004 /* tx complete */
#define INT_TxAvailable     0x0008 /* tx available */
#define INT_RxComplete      0x0010 /* rx complete */
#define INT_RxEarly         0x0020 /* rx early */
#define INT_Requested       0x0040 /* interrupt requested */
#define INT_UpdateStats     0x0080 /* update statistics */

/* Rx Status register bits */
#define RXS_Error           0x4000 /* error in packet */
#define RXS_Length          0x07FF /* bytes in Rx FIFO */
#define RXS_ErrType         0x3800 /* Rx error type, bit 13-11 */
#define RXS_Overrun         0x0000 /* overrun error */
#define RXS_Oversize        0x0800 /* oversize packet error */
#define RXS_Dribble         0x1000 /* dribble bit (not an error) */
#define RXS_Runt            0x1800 /* runt packet error */
#define RXS_Framing         0x2000 /* framing error */
#define RXS_CRC             0x2800 /* CRC error */

/* Tx Status register bits */

/* Window Numbers */
#define WNO_Setup           0x0000 /* setup/configuration */
#define WNO_Operating       0x0001 /* operating set */
#define WNO_StationAddress  0x0002 /* station address setup/read */
#define WNO_Diagnostics     0x0004 /* diagnostics */
#define WNO_Statistics      0x0006 /* statistics */

/* Register offsets - Window 1 (WNO_Operating) */
#define REG_CmdStatus       0x000E /* command/status */
#define REG_TxFree          0x000C /* free transmit bytes */
#define REG_TxStatus        0x000B /* transmit status (byte) */
#define REG_RxStatus        0x0008 /* receive status */
#define REG_RxFIFO          0x0000 /* Rx FIFO read */
```

```
#define REG_TxFIFO          0x0000  /* TxFIFO write */

/* Register offsets - Window 0 (WNO_Setup) */
#define REG_CfgControl      0x0004  /* configuration control */

/* Register offsets - Window 2 (WNO_StationAddress) */
#define REG_SA0_1          0x0000  /* station address bytes 0,1 */

/* Register offsets - Window 3 (WNO_FIFO) */

/* Register offsets - Window 4 (WNO_Diagnostics) */
#define REG_MediaStatus     0x000A  /* media type/status */

/* Register offsets - Window 5 (WNO_Readable) */

/* Register offsets - Window 6 (WNO_Statistics) */
#define REG_TxBytes         0x000C  /* tx bytes ok */
#define REG_RxBytes         0x000A  /* rx bytes ok */
#define REG_TxDefer         0x0008  /* tx frames deferred (byte) */
#define REG_RxFrames        0x0007  /* rx frames ok (byte) */
#define REG_TxFrames        0x0006  /* tx frames ok (byte) */
#define REG_RxDiscarded     0x0005  /* rx frames discarded (byte) */
#define REG_TxLate          0x0004  /* tx frames late coll. (byte) */
#define REG_TxSingleColl    0x0003  /* tx frames one coll. (byte) */
#define REG_TxMultColl      0x0002  /* tx frames mult. coll. (byte) */
#define REG_TxNoCD          0x0001  /* tx frames no CDheartbt (byte) */
#define REG_TxCarrierLost   0x0000  /* tx frames carrier lost (byte) */

/* Various command arguments */

#define FilterIndividual     0x0001  /* individual address */
#define FilterMulticast     0x0002  /* multicast/group addresses */
#define FilterBroadcast     0x0004  /* broadcast address */
#define FilterPromiscuous   0x0008  /* promiscuous mode */

/* Resource Configuration Register bits */
#define EL3_CONFIG_IRQ_MASK 0xF000

/* Address Configuration Register bits */
#define EL3_CONFIG_XCVR_MASK 0xC000
#define EL3_CONFIG_IOBASE_MASK 0x001F

#define TP_XCVR              0x0000
#define BNC_XCVR             0xC000
#define AUI_XCVR             0x4000

#define EL3_IO_BASE_ADDR     0x200

/* Transmit Preamble */

/* Bits in various diagnostics registers */
#define MediaBeatEnable      0x0080  /* link beat enable (TP) */
#define MediaJabberEnable    0x0040  /* jabber enable (TP) */

/* Board identification codes, byte swapped in Rev 0 */
#define EL3_3COM_CODE        0x6D50  /* EISA manufacturer code */
#define EL3_PRODUCT_ID       0x9050  /* Product ID for ISA board */

/* EEPROM access */
#define EE_3COM_NODE_ADDR    0x00
#define EE_PROD_ID           0x03
#define EE_MANUFACTURING_DATA 0x04
#define EE_3COM_CODE         0x07
#define EE_ADDR_CFG          0x08
#define EE_RESOURCE_CFG      0x09
#define EE_SW_CONFIG_INFO    0x0D
#define EE_PROD_ID_MASK      0xF0FF  /* Mask off revision nibble */

/* Contention logic */
#define EL3_READ_EEPROM      0x80
#define EL3_ID_GLOBAL_RESET  0xC0
#define EL3_SET_TAG_REGISTER 0xD0
#define EL3_ACTIVATE_AND_SET_IO 0xE0
#define EL3_ACTIVATE         0xFF
```

```
/* Software Configuration Register bits */

/* Configuration Control Register bits */
#define EL3_EnableAdapter      0x01

/* EL3 access macros */
#define inb_el3(dep,reg) (inb((dep)->de_base_port+(reg)))
#define inw_el3(dep,reg) (inw((dep)->de_base_port+(reg)))
#define outb_el3(dep,reg,data) (outb((dep)->de_base_port+(reg),(data)))
#define outw_el3(dep,reg,data) (outw((dep)->de_base_port+(reg),(data)))

#define SetWindow(win) \
    outw(dep->de_base_port+REG_CmdStatus,CMD_SelectWindow|(win))

/** 3c509.h **/
```

```

/*
** File:      8390.c          May 02, 2000
**
** Author:    Giovanni Falzoni <gfalzoni@inwind.it>
**
** This file contains an ethernet device driver for NICs
** equipped with the National Semiconductor NS 8390 chip.
** It has to be associated with the board specific driver.
** Rewritten from Minix 2.0.0 ethernet driver dp8390.c
** to extract the NS 8390 common functions.
**
** $Id: 8390.c,v 1.4 2005/09/04 18:52:16 beng Exp $
*/

#include "drivers.h"
#include <minix/com.h>
#include <net/hton.h>
#include <net/gen/ether.h>
#include <net/gen/eth_io.h>
#include "dp.h"

#if (ENABLE_DP8390 == 1)

#define PIO16    0          /* NOTE: pio 16 functions missing */

#include "8390.h"

#define sys_nic2mem(srcOffs,dstProc,dstOffs,length) \
    sys_vircopy(SELF,dep->de_memsegm,(vir_bytes)(srcOffs),\
                (dstProc),D,(vir_bytes)(dstOffs),length)
#define sys_user2nic(srcProc,srcOffs,dstOffs,length) \
    sys_vircopy((srcProc),D,(vir_bytes)(srcOffs),\
                SELF,dep->de_memsegm,(vir_bytes)(dstOffs),length)

static const char RdmaErrMsg[] = "remote dma failed to complete";

/*
** Name:      void ns_rw_setup(dpeth_t *dep, int mode, int size, ul6_t offset);
** Function:   Sets the board for reading/writing.
*/
static void ns_rw_setup(dpeth_t *dep, int mode, int size, ul6_t offset)
{
    if (mode == CR_DM_RW) outb_reg0(dep, DP_ISR, ISR_RDC);
    outb_reg0(dep, DP_RBCR0, size & 0xFF);
    outb_reg0(dep, DP_RBCR1, (size >> 8) & 0xFF);
    outb_reg0(dep, DP_RSAR0, offset & 0xFF);
    outb_reg0(dep, DP_RSAR1, (offset >> 8) & 0xFF);
    mode |= (CR_PS_P0 | CR_STA);
    outb_reg0(dep, DP_CR, mode);
    return;
}

/*
** Name:      void ns_start_xmit(dpeth_t *dep, int size, int pageno);
** Function:   Sets the board for for transmitting and fires it.
*/
static void ns_start_xmit(dpeth_t * dep, int size, int pageno)
{
    outb_reg0(dep, DP_TPSR, pageno);
    outb_reg0(dep, DP_TBCR1, size >> 8);
    outb_reg0(dep, DP_TBCR0, size & 0xFF);
    outb_reg0(dep, DP_CR, CR_NO_DMA | CR_STA | CR_TXP);    /* Fires transmission */
    return;
}

/*
** Name:      void mem_getblock(dpeth_t *dep, ul6_t offset,
**                               int size, void *dst)
** Function:   Reads a block of packet from board (shared memory).
*/
static void mem_getblock(dpeth_t *dep, ul6_t offset, int size, void *dst)
{

```

```

sys_nic2mem(dep->de_linmem + offset, SELF, dst, size);
return;
}

/*
** Name:      void mem_nic2user(dpeth_t *dep, int pageno, int pktsize);
** Function:  Copies a packet from board to user area (shared memory).
*/
static void mem_nic2user(dpeth_t *dep, int pageno, int pktsize)
{
    phys_bytes offset, phys_user;
    iovec_dat_t *iovp = &dep->de_read_iovec;
    int bytes, ix = 0;

    /* Computes shared memory address (skipping receive header) */
    offset = pageno * DP_PAGESIZE + sizeof(dp_rcvhdr_t);

    do {
        /* Reads chunks of packet into user area */

        bytes = iovp->iod_iovec[ix].iov_size; /* Size of a chunk */
        if (bytes > pktsize) bytes = pktsize;

        /* Reads from board to user area */
        if ((offset + bytes) > (dep->de_stoppage * DP_PAGESIZE)) {

            /* Circular buffer wrap-around */
            bytes = dep->de_stoppage * DP_PAGESIZE - offset;
            sys_nic2mem(dep->de_linmem + offset, iovp->iod_proc_nr,
                iovp->iod_iovec[ix].iov_addr, bytes);
            pktsize -= bytes;
            phys_user += bytes;
            bytes = iovp->iod_iovec[ix].iov_size - bytes;
            if (bytes > pktsize) bytes = pktsize;
            offset = dep->de_startpage * DP_PAGESIZE;
        }
        sys_nic2mem(dep->de_linmem + offset, iovp->iod_proc_nr,
            iovp->iod_iovec[ix].iov_addr, bytes);
        offset += bytes;

        if (++ix >= IOVEC_NR) { /* Next buffer of IO vector */
            dp_next_iovec(iovp);
            ix = 0;
        }
        /* Till packet done */
    } while ((pktsize -= bytes) > 0);
    return;
}

/*
** Name:      void mem_user2nic(dpeth_t *dep, int pageno, int pktsize)
** Function:  Copies a packet from user area to board (shared memory).
*/
static void mem_user2nic(dpeth_t *dep, int pageno, int pktsize)
{
    phys_bytes offset, phys_user;
    iovec_dat_t *iovp = &dep->de_write_iovec;
    int bytes, ix = 0;

    /* Computes shared memory address */
    offset = pageno * DP_PAGESIZE;

    do {
        /* Reads chunks of packet from user area */

        bytes = iovp->iod_iovec[ix].iov_size; /* Size of chunk */
        if (bytes > pktsize) bytes = pktsize;

        /* Reads from user area to board (shared memory) */
        sys_user2nic(iovp->iod_proc_nr, iovp->iod_iovec[ix].iov_addr,
            dep->de_linmem + offset, bytes);
        offset += bytes;

        if (++ix >= IOVEC_NR) { /* Next buffer of IO vector */
            dp_next_iovec(iovp);

```

```

        ix = 0;
    }
    /* Till packet done */
} while ((pktsize -= bytes) > 0);
return;
}

/*
** Name:          void pio_getblock(dpeth_t *dep, u16_t offset,
**               int size, void *dst)
** Function:      Reads a block of packet from board (Prog. I/O).
*/
static void pio_getblock(dpeth_t *dep, u16_t offset, int size, void *dst)
{
    /* Sets up board for reading */
    ns_rw_setup(dep, CR_DM_RR, size, offset);

#if PIO16 == 0
    insb(dep->de_data_port, SELF, dst, size);
#else
    if (dep->de_16bit == TRUE) {
        insw(dep->de_data_port, dst, size);
    } else {
        insb(dep->de_data_port, dst, size);
    }
#endif
    return;
}

/*
** Name:          void pio_nic2user(dpeth_t *dep, int pageno, int pktsize)
** Function:      Copies a packet from board to user area (Prog. I/O).
*/
static void pio_nic2user(dpeth_t *dep, int pageno, int pktsize)
{
    phys_bytes phys_user;
    iovec_dat_t *iovp = &dep->de_read_iovec;
    unsigned offset; int bytes, ix = 0;

    /* Computes memory address (skipping receive header) */
    offset = pageno * DP_PAGESIZE + sizeof(dp_rcvhdr_t);
    /* Sets up board for reading */
    ns_rw_setup(dep, CR_DM_RR, ((offset + pktsize) > (dep->de_stoppage * DP_PAGESIZE)) ?
        (dep->de_stoppage * DP_PAGESIZE) - offset : pktsize, offset);

    do {
        /* Reads chunks of packet into user area */

        bytes = iovp->iod_iovec[ix].iov_size; /* Size of a chunk */
        if (bytes > pktsize) bytes = pktsize;

        if ((offset + bytes) > (dep->de_stoppage * DP_PAGESIZE)) {
            /* Circular buffer wrap-around */
            bytes = dep->de_stoppage * DP_PAGESIZE - offset;
            insb(dep->de_data_port, iovp->iod_proc_nr, (void*)(iovp->iod_iovec[ix].io
v_addr), bytes);
            pktsize -= bytes;
            iovp->iod_iovec[ix].iov_addr += bytes;
            bytes = iovp->iod_iovec[ix].iov_size - bytes;
            if (bytes > pktsize) bytes = pktsize;
            offset = dep->de_startpage * DP_PAGESIZE;
            ns_rw_setup(dep, CR_DM_RR, pktsize, offset);
        }
        insb(dep->de_data_port, iovp->iod_proc_nr, (void*)(iovp->iod_iovec[ix].iov_addr),
bytes);
        offset += bytes;

        if (++ix >= IOVEC_NR) { /* Next buffer of IO vector */
            dp_next_iovec(iovp);
            ix = 0;
        }
    } while ((pktsize -= bytes) > 0);
}

```

```

    return;
}

/*
** Name:      void pio_user2nic(dpeth_t *dep, int pageno, int pktsize)
** Function:   Copies a packet from user area to board (Prog. I/O).
*/
static void pio_user2nic(dpeth_t *dep, int pageno, int pktsize)
{
    phys_bytes phys_user;
    iovec_dat_t *iovp = &dep->de_write_iovec;
    int bytes, ix = 0;

    /* Sets up board for writing */
    ns_rw_setup(dep, CR_DM_RW, pktsize, pageno * DP_PAGESIZE);

    do {
        /* Reads chunks of packet from user area */

        bytes = iovp->iod_iovec[ix].iov_size; /* Size of chunk */
        if (bytes > pktsize) bytes = pktsize;
        outsb(dep->de_data_port, iovp->iod_proc_nr,
              (void*)(iovp->iod_iovec[ix].iov_addr), bytes);

        if (++ix >= IOVEC_NR) { /* Next buffer of I/O vector */
            dp_next_iovec(iovp);
            ix = 0;
        }
        /* Till packet done */
    } while ((pktsize -= bytes) > 0);

    for (ix = 0; ix < 100; ix += 1) {
        if (inb_reg0(dep, DP_ISR) & ISR_RDC) break;
    }
    if (ix == 100) {
        panic(dep->de_name, RdmaErrMsg, NO_NUM);
    }
    return;
}

/*
** Name:      void ns_stats(dpeth_t * dep)
** Function:   Updates counters reading from device
*/
static void ns_stats(dpeth_t * dep)
{
    dep->de_stat.ets_CRCerr += inb_reg0(dep, DP_CNTR0);
    dep->de_stat.ets_recvErr += inb_reg0(dep, DP_CNTR1);
    dep->de_stat.ets_fifoOver += inb_reg0(dep, DP_CNTR2);
    return;
}

/*
** Name:      void ns_dodump(dpeth_t * dep)
** Function:   Displays statistics (a request from F5 key).
*/
static void ns_dodump(dpeth_t * dep)
{
    ns_stats(dep);
    /* Forces reading fo counters from board */
    return;
}

/*
** Name:      void ns_reinit(dpeth_t *dep)
** Function:   Updates receiver configuration.
*/
static void ns_reinit(dpeth_t * dep)
{
    int dp_reg = 0;

    if (dep->de_flags & DEF_PROMISC) dp_reg |= RCR_AB | RCR_PRO | RCR_AM;
    if (dep->de_flags & DEF_BROAD) dp_reg |= RCR_AB;
    if (dep->de_flags & DEF_MULTI) dp_reg |= RCR_AM;

```



```

outb_reg0(dep, DP_CR, CR_PS_P0);
outb_reg0(dep, DP_RCR, dp_reg);
return;
}

/*
** Name:      void ns_send(dpeth_t * dep, int from_int, int size)
** Function:  Transfers packet to device and starts sending.
*/
static void ns_send(dpeth_t * dep, int from_int, int size)
{
    int queue;

    if (queue = dep->de_sendq_head, dep->de_sendq[queue].sq_filled) {
        if (from_int) panic(dep->de_name, "should not be sending ", NO_NUM);
        dep->de_send_s = size;
        return;
    }
    (dep->de_user2nicf) (dep, dep->de_sendq[queue].sq_sendpage, size);
    dep->bytes_Tx += (long) size;
    dep->de_sendq[queue].sq_filled = TRUE;
    dep->de_flags |= (DEF_XMIT_BUSY | DEF_ACK_SEND);
    if (dep->de_sendq_tail == queue) { /* there it goes.. */
        ns_start_xmit(dep, size, dep->de_sendq[queue].sq_sendpage);
    } else
        dep->de_sendq[queue].sq_size = size;

    if (++queue == dep->de_sendq_nr) queue = 0;
    dep->de_sendq_head = queue;
    dep->de_flags &= NOT(DEF_SENDING);

    return;
}

/*
** Name:      void ns_reset(dpeth_t *dep)
** Function:  Resets device.
*/
static void ns_reset(dpeth_t * dep)
{
    int ix;

    /* Stop chip */
    outb_reg0(dep, DP_CR, CR_STP | CR_NO_DMA);
    outb_reg0(dep, DP_RBCR0, 0);
    outb_reg0(dep, DP_RBCR1, 0);
    for (ix = 0; ix < 0x1000 && ((inb_reg0(dep, DP_ISR) & ISR_RST) == 0); ix += 1)
        /* Do nothing */ ;
    outb_reg0(dep, DP_TCR, TCR_1EXTERNAL | TCR_OFST);
    outb_reg0(dep, DP_CR, CR_STA | CR_NO_DMA);
    outb_reg0(dep, DP_TCR, TCR_NORMAL | TCR_OFST);

    /* Acknowledge the ISR_RDC (remote dma) interrupt. */
    for (ix = 0; ix < 0x1000 && ((inb_reg0(dep, DP_ISR) & ISR_RDC) == 0); ix += 1)
        /* Do nothing */ ;
    outb_reg0(dep, DP_ISR, inb_reg0(dep, DP_ISR) & NOT(ISR_RDC));

    /* Reset the transmit ring. If we were transmitting a packet, we
     * pretend that the packet is processed. Higher layers will
     * retransmit if the packet wasn't actually sent. */
    dep->de_sendq_head = dep->de_sendq_tail = 0;
    for (ix = 0; ix < dep->de_sendq_nr; ix++)
        dep->de_sendq[ix].sq_filled = FALSE;
    ns_send(dep, TRUE, dep->de_send_s);
    return;
}

/*
** Name:      void ns_rcv(dpeth_t *dep, int fromint, int size)
** Function:  Gets a packet from device
*/
static void ns_rcv(dpeth_t *dep, int fromint, int size)
{
    dp_rcvhdr_t header;

```

```

dp_rcvhdr_t dummy;
unsigned pageno, curr, next;
vir_bytes length;
int packet_processed = FALSE;
#ifdef ETH_IGN_PROTO
    ul6_t eth_type;
#endif

pageno = inb_reg0(dep, DP_BNRY) + 1;
if (pageno == dep->de_stoppage) pageno = dep->de_startpage;

do {
    /* */
    outb_reg0(dep, DP_CR, CR_PS_P1);
    curr = inb_reg1(dep, DP_CURR);
    outb_reg0(dep, DP_CR, CR_PS_P0 | CR_NO_DMA | CR_STA);

    if (curr == pageno) break;

    (dep->de_getblockf) (dep, pageno * DP_PAGESIZE, sizeof(header), &header);
#ifdef ETH_IGN_PROTO
    (dep->de_getblockf) (dep, pageno * DP_PAGESIZE + sizeof(header) + 2 * sizeof(ether_addr_t), sizeof(eth_type), &eth_type);
#endif
    length = (header.dr_rbc1 | (header.dr_rbc0 << 8)) - sizeof(dp_rcvhdr_t);
    next = header.dr_next;

    if (length < ETH_MIN_PACK_SIZE || length > ETH_MAX_PACK_SIZE) {
        printf("%s: packet with strange length arrived: %d\n", dep->de_name, length);
        dep->de_stat.ets_recvErr += 1;
        next = curr;
    } else if (next < dep->de_startpage || next >= dep->de_stoppage) {
        printf("%s: strange next page\n", dep->de_name);
        dep->de_stat.ets_recvErr += 1;
        next = curr;
    }

#ifdef ETH_IGN_PROTO
    } else if (eth_type == eth_ign_proto) {
        /* Hack: ignore packets of a given protocol */
        static int first = TRUE;
        if (first) {
            first = FALSE;
            printf("%s: dropping proto %04x packet\n", dep->de_name, ntohs(eth_ign_proto));
        }
        next = curr;
    }
#endif
    } else if (header.dr_status & RSR_FO) {
        /* This is very serious, issue a warning and reset buffers */
        printf("%s: fifo overrun, resetting receive buffer\n", dep->de_name);
        dep->de_stat.ets_fifoOver += 1;
        next = curr;
    }

    } else if ((header.dr_status & RSR_PRX) && (dep->de_flags & DEF_ENABLED)) {

        if (!(dep->de_flags & DEF_READING)) break;

        (dep->de_nic2userf) (dep, pageno, length);
        dep->de_read_s = length;
        dep->de_flags |= DEF_ACK_RECV;
        dep->de_flags &= NOT(DEF_READING);
        packet_processed = TRUE;
    }
    dep->bytes_Rx += (long) length;
    dep->de_stat.ets_packetR += 1;
    outb_reg0(dep, DP_BNRY, (next == dep->de_startpage ? dep->de_stoppage : next) - 1);
    pageno = next;

} while (!packet_processed);
if 0
    if ((dep->de_flags & (DEF_READING | DEF_STOPPED)) == (DEF_READING | DEF_STOPPED))
        /* The chip is stopped, and all arrived packets delivered */

```

```

    (*dep->de_resetf) (dep);
    dep->de_flags &= NOT(DEF_STOPPED);
#endif
    return;
}

/*
** Name:      void ns_interrupt(dpeth_t * dep)
** Function:  Handles interrupt.
*/
static void ns_interrupt(dpeth_t * dep)
{
    int isr, tsr;
    int size, queue;

    while ((isr = inb_reg0(dep, DP_ISR)) != 0) {

        outb_reg0(dep, DP_ISR, isr);
        if (isr & (ISR_PTX | ISR_TXE)) {

            tsr = inb_reg0(dep, DP_TSR);
            if (tsr & TSR_PTX) {
                dep->de_stat.ets_packetT++;
            }
            if (tsr & TSR_COL) dep->de_stat.ets_collision++;
            if (tsr & (TSR_ABT | TSR_FU)) {
                dep->de_stat.ets_fifoUnder++;
            }
            if ((isr & ISR_TXE) || (tsr & (TSR_CRS | TSR_CDH | TSR_OWC))) {
                printf("%s: got send Error (0x%02X)\n", dep->de_name, tsr);
                dep->de_stat.ets_sendErr++;
            }
            queue = dep->de_sendq_tail;

            if (!(dep->de_sendq[queue].sq_filled)) {
                /* Hardware bug? */
                printf("%s: transmit interrupt, but not sending\n", dep->de_name);
                continue;
            }
            dep->de_sendq[queue].sq_filled = FALSE;
            if (++queue == dep->de_sendq_nr) queue = 0;
            dep->de_sendq_tail = queue;
            if (dep->de_sendq[queue].sq_filled) {
                ns_start_xmit(dep, dep->de_sendq[queue].sq_size,
                    dep->de_sendq[queue].sq_sendpage);
            }
            if (dep->de_flags & DEF_SENDING) {
                ns_send(dep, TRUE, dep->de_send_s);
            }
        }
        if (isr & ISR_PRX) {
            ns_recv(dep, TRUE, 0);
        }
        if (isr & ISR_RXE) {
            printf("%s: got rcv Error (0x%04X)\n", dep->de_name, inb_reg0(dep, DP_RSR));
            dep->de_stat.ets_rcvErr++;
        }
        if (isr & ISR_CNT) {
            dep->de_stat.ets_CRCerr += inb_reg0(dep, DP_CNTR0);
            dep->de_stat.ets_rcvErr += inb_reg0(dep, DP_CNTR1);
            dep->de_stat.ets_fifoOver += inb_reg0(dep, DP_CNTR2);
        }
        if (isr & ISR_OVW) {
            printf("%s: got overwrite warning\n", dep->de_name);
        }
        if (isr & ISR_RDC) {
            /* Nothing to do */
        }
        if (isr & ISR_RST) {
            /* This means we got an interrupt but the ethernet
             * chip is shutdown. We set the flag DEF_STOPPED, and
             * continue processing arrived packets. When the
             * receive buffer is empty, we reset the dp8390. */
            printf("%s: network interface stopped\n", dep->de_name);
            dep->de_flags |= DEF_STOPPED;
        }
    }
}

```

```

        break;
    }
}
if ((dep->de_flags & (DEF_READING | DEF_STOPPED)) == (DEF_READING | DEF_STOPPED)) {
    /* The chip is stopped, and all arrived packets delivered */
    ns_reset(dep);
    dep->de_flags &= NOT(DEF_STOPPED);
}
return;
}

/*
** Name:      void ns_init(dpeth_t *dep)
** Function:  Initializes the NS 8390
*/
void ns_init(dpeth_t * dep)
{
    int dp_reg;
    int ix;

    /* NS8390 initialization (as recommended in National Semiconductor specs) */
    outb_reg0(dep, DP_CR, CR_PS_P0 | CR_STP | CR_NO_DMA); /* 0x21 */
#ifdef PIO16
    outb_reg0(dep, DP_DCR, (DCR_BYTEWIDE | DCR_LITTLE_ENDIAN | DCR_8BYTES | DCR_BMS));
#else
    outb_reg0(dep, DP_DCR, (((dep->de_16bit) ? DCR_WORDWIDE : DCR_BYTEWIDE) |
        DCR_LITTLE_ENDIAN | DCR_8BYTES | DCR_BMS));
#endif
    outb_reg0(dep, DP_RBCR0, 0);
    outb_reg0(dep, DP_RBCR1, 0);
    outb_reg0(dep, DP_RCR, RCR_MON); /* Sets Monitor mode */
    outb_reg0(dep, DP_TCR, TCR_INTERNAL); /* Sets Loopback mode 1 */
    outb_reg0(dep, DP_PSTART, dep->de_startpage);
    outb_reg0(dep, DP_PSTOP, dep->de_stoppage);
    outb_reg0(dep, DP_BNRY, dep->de_stoppage - 1);
    outb_reg0(dep, DP_ISR, 0xFF); /* Clears Interrupt Status Register */
    outb_reg0(dep, DP_IMR, 0); /* Clears Interrupt Mask Register */

    /* Copies station address in page 1 registers */
    outb_reg0(dep, DP_CR, CR_PS_P1 | CR_NO_DMA); /* Selects Page 1 */
    for (ix = 0; ix < SA_ADDR_LEN; ix += 1) /* Initializes address */
        outb_reg1(dep, DP_PAR0 + ix, dep->de_address.ea_addr[ix]);
    for (ix = DP_MAR0; ix <= DP_MAR7; ix += 1) /* Initializes address */
        outb_reg1(dep, ix, 0xFF);

    outb_reg1(dep, DP_CURR, dep->de_startpage);
    outb_reg1(dep, DP_CR, CR_PS_P0 | CR_NO_DMA); /* Selects Page 0 */

    inb_reg0(dep, DP_CNTR0); /* Resets counters by reading them */
    inb_reg0(dep, DP_CNTR1);
    inb_reg0(dep, DP_CNTR2);

    dp_reg = IMR_PRXE | IMR_PTXE | IMR_RXEE | IMR_TXEE | IMR_OVWE | IMR_CNTE;
    outb_reg0(dep, DP_ISR, 0xFF); /* Clears Interrupt Status Register */
    outb_reg0(dep, DP_IMR, dp_reg); /* Sets Interrupt Mask register */

    dp_reg = 0;
    if (dep->de_flags & DEF_PROMISC) dp_reg |= RCR_AB | RCR_PRO | RCR_AM;
    if (dep->de_flags & DEF_BROAD) dp_reg |= RCR_AB;
    if (dep->de_flags & DEF_MULTI) dp_reg |= RCR_AM;
    outb_reg0(dep, DP_RCR, dp_reg); /* Sets receive as requested */
    outb_reg0(dep, DP_TCR, TCR_NORMAL); /* Sets transmitter */

    outb_reg0(dep, DP_CR, CR_STA | CR_NO_DMA); /* Starts board */

    /* Initializes the send queue. */
    for (ix = 0; ix < dep->de_sendq_nr; ix += 1)
        dep->de_sendq[ix].sq_filled = 0;
    dep->de_sendq_head = dep->de_sendq_tail = 0;

    /* Device specific functions */
    if (!dep->de_prog_IO) {
        dep->de_user2nicf = mem_user2nic;
    }
}

```

```

        dep->de_nic2userf = mem_nic2user;
        dep->de_getblockf = mem_getblock;
    } else {
#if PIO16 == 0
        dep->de_user2nicf = pio_user2nic;
        dep->de_nic2userf = pio_nic2user;
        dep->de_getblockf = pio_getblock;
#else
#error Missing I/O functions for pio 16 bits
#endif
    }
    dep->de_recvf = ns_recv;
    dep->de_sendf = ns_send;
    dep->de_flagsf = ns_reinit;
    dep->de_resetf = ns_reset;
    dep->de_getstatsf = ns_stats;
    dep->de_dumpstatsf = ns_dodump;
    dep->de_interruptf = ns_interrupt;

    return;                                /* Done */
}

#if PIO16 == 1

/*
** Name:          void dp_piol6_user2nic(dpeth_t *dep, int pageno, int pktsize)
** Function:      Copies a packet from user area to board (Prog. I/O, 16bits).
*/
static void dp_piol6_user2nic(dpeth_t *dep, int pageno, int pktsize)
{
    u8_t two_bytes[2];
    phys_bytes phys_user, phys_2bytes = vir2phys(two_bytes);
    vir_bytes ecount = (pktsize + 1) & NOT(0x0001);
    int bytes, ix = 0, odd_byte = 0;
    iovec_dat_t *iovp = &dep->de_write_iovec;

    outb_reg0(dep, DP_ISR, ISR_RDC);
    dp_read_setup(dep, ecount, pageno * DP_PAGESIZE);

    do {
        bytes = iovp->iod_iovec[ix].iov_size;
        if (bytes > pktsize) bytes = pktsize;

        phys_user = numap(iovp->iod_proc_nr, iovp->iod_iovec[ix].iov_addr, bytes);
        if (!phys_user) panic(dep->de_name, UmapErrMsg, NO_NUM);

        if (odd_byte) {
            phys_copy(phys_user, phys_2bytes + 1, (phys_bytes) 1);
            out_word(dep->de_data_port, *(u16_t *)two_bytes);
            pktsize--;
            bytes--;
            phys_user++;
            odd_byte = 0;
            if (!bytes) continue;
        }
        ecount = bytes & NOT(0x0001);
        if (ecount != 0) {
            phys_outsw(dep->de_data_port, phys_user, ecount);
            pktsize -= ecount;
            bytes -= ecount;
            phys_user += ecount;
        }
        if (bytes) {
            phys_copy(phys_user, phys_2bytes, (phys_bytes) 1);
            pktsize--;
            bytes--;
            phys_user++;
            odd_byte = 1;
        }
        if (++ix >= IOVEC_NR) { /* Next buffer of I/O vector */
            dp_next_iovec(iovp);
            ix = 0;
        }
    }
}

```

```

} while (bytes > 0);

if (odd_byte) out_word(dep->de_data_port, *(u16_t *) two_bytes);
for (ix = 0; ix < 100; ix++) {
    if (inb_reg0(dep, DP_ISR) & ISR_RDC) break;
}
if (ix == 100) {
    panic(dep->de_name, RdmaErrMsg, NO_NUM);
}
return;
}

/*
** Name:      void dp_piol6_nic2user(dpeth_t *dep, int pageno, int pktsize)
** Function:  Copies a packet from board to user area (Prog. I/O, 16bits).
*/
static void dp_piol6_nic2user(dpeth_t * dep, int nic_addr, int count)
{
    phys_bytes phys_user;
    vir_bytes ecount;
    int bytes, i;
    u8_t two_bytes[2];
    phys_bytes phys_2bytes;
    int odd_byte;

    ecount = (count + 1) & ~1;
    phys_2bytes = vir2phys(two_bytes);
    odd_byte = 0;

    dp_read_setup(dep, ecount, nic_addr);

    i = 0;
    while (count > 0) {
        if (i >= IOVEC_NR) {
            dp_next_iovec(iovp);
            i = 0;
            continue;
        }
        bytes = iovp->ioc_iovec[i].ioc_size;
        if (bytes > count) bytes = count;

        phys_user = numap(iovp->ioc_proc_nr,
            iovp->ioc_iovec[i].ioc_addr, bytes);
        if (!phys_user) panic(dep->de_name, UmapErrMsg, NO_NUM);
        if (odd_byte) {
            phys_copy(phys_2bytes + 1, phys_user, (phys_bytes) 1);
            count--;
            bytes--;
            phys_user++;
            odd_byte = 0;
            if (!bytes) continue;
        }
        ecount = bytes & ~1;
        if (ecount != 0) {
            phys_insw(dep->de_data_port, phys_user, ecount);
            count -= ecount;
            bytes -= ecount;
            phys_user += ecount;
        }
        if (bytes) {
            *(u16_t *) two_bytes = in_word(dep->de_data_port);
            phys_copy(phys_2bytes, phys_user, (phys_bytes) 1);
            count--;
            bytes--;
            phys_user++;
            odd_byte = 1;
        }
    }
    return;
}

#endif /* PIO16 == 1 */

#endif /* ENABLE_DP8390 */

```

```
/** end 8390.c */
```

```

/*
** File:      8390.h          May 02, 2000
**
** Author:    Giovanni Falzoni <gfalzoni@inwind.it>
**
** National Semiconductor NS 8390 Network Interface Controller
**
** $Log: 8390.h,v $
** Revision 1.2  2005/08/22 15:17:40  beng
** Remove double-blank lines (Al)
**
** Revision 1.1  2005/06/29 10:16:46  beng
** Import of dpeth 3c501/3c509b/.. ethernet driver by
** Giovanni Falzoni <fgalzoni@inwind.it>.
**
** Revision 2.0  2005/06/26 16:16:46  lsodgf0
** Initial revision for Minix 3.0.6
**
** $Id: 8390.h,v 1.2 2005/08/22 15:17:40 beng Exp $
*/

#define DP_PAGESIZE      256      /* NS 8390 page size */
#define SENDQ_PAGES      6        /* SENDQ_PAGES * DP_PAGESIZE >= 1514 bytes */

/* Page 0, read/write ----- */
#define DP_CR            0x00      /* Command Register          RW */
#define DP_CLDA0         0x01      /* Current Local Dma Address 0 RO */
#define DP_PSTART        0x01      /* Page Start Register       WO */
#define DP_CLDA1         0x02      /* Current Local Dma Address 1 RO */
#define DP_PSTOP         0x02      /* Page Stop Register        WO */
#define DP_BNRY          0x03      /* Boundary Pointer          RW */
#define DP_TSR           0x04      /* Transmit Status Register   RO */
#define DP_TPSR          0x04      /* Transmit Page Start Register WO */
#define DP_NCR           0x05      /* No. of Collisions Register RO */
#define DP_TBCR0         0x05      /* Transmit Byte Count Reg. 0 WO */
#define DP_FIFO          0x06      /* Fifo                      RO */
#define DP_TBCR1         0x06      /* Transmit Byte Count Reg. 1 WO */
#define DP_ISR           0x07      /* Interrupt Status Register  RW */
#define DP_CRDA0         0x08      /* Current Remote Dma Addr.Low RO */
#define DP_RSAR0         0x08      /* Remote Start Address Low   WO */
#define DP_CRDA1         0x09      /* Current Remote Dma Addr.High RO */
#define DP_RSAR1         0x09      /* Remote Start Address High  WO */
#define DP_RBCR0         0x0A      /* Remote Byte Count Low      WO */
#define DP_RBCR1         0x0B      /* Remote Byte Count High     WO */
#define DP_RSR           0x0C      /* Receive Status Register    RO */
#define DP_RCR           0x0C      /* Receive Config. Register   WO */
#define DP_CNTR0         0x0D      /* Tally Counter 0            RO */
#define DP_TCR           0x0D      /* Transmit Config. Register  WO */
#define DP_CNTR1         0x0E      /* Tally Counter 1            RO */
#define DP_DCR           0x0E      /* Data Configuration Register WO */
#define DP_CNTR2         0x0F      /* Tally Counter 2            RO */
#define DP_IMR           0x0F      /* Interrupt Mask Register     WO */

/* Page 1, read/write ----- */
/* DP_CR 0x00 Command Register */
#define DP_PAR0          0x01      /* Physical Address Register 0 */
#define DP_PAR1          0x02      /* Physical Address Register 1 */
#define DP_PAR2          0x03      /* Physical Address Register 2 */
#define DP_PAR3          0x04      /* Physical Address Register 3 */
#define DP_PAR4          0x05      /* Physical Address Register 4 */
#define DP_PAR5          0x06      /* Physical Address Register 5 */
#define DP_CURR          0x07      /* Current Page Register */
#define DP_MAR0          0x08      /* Multicast Address Register 0 */
#define DP_MAR1          0x09      /* Multicast Address Register 1 */
#define DP_MAR2          0x0A      /* Multicast Address Register 2 */
#define DP_MAR3          0x0B      /* Multicast Address Register 3 */
#define DP_MAR4          0x0C      /* Multicast Address Register 4 */
#define DP_MAR5          0x0D      /* Multicast Address Register 5 */
#define DP_MAR6          0x0E      /* Multicast Address Register 6 */
#define DP_MAR7          0x0F      /* Multicast Address Register 7 */

/* Bits in dp_cr */
#define CR_STP           0x01      /* Stop: software reset */
#define CR_STA           0x02      /* Start: activate NIC */

```



```

#define CR_TXP          0x04    /* Transmit Packet */
#define CR_DMA          0x38    /* Mask for DMA control */
#define CR_DM_RR        0x08    /* DMA: Remote Read */
#define CR_DM_RW        0x10    /* DMA: Remote Write */
#define CR_DM_SP        0x18    /* DMA: Send Packet */
#define CR_NO_DMA       0x20    /* DMA: Stop Remote DMA Operation */
#define CR_PS           0xC0    /* Mask for Page Select */
#define CR_PS_P0        0x00    /* Register Page 0 */
#define CR_PS_P1        0x40    /* Register Page 1 */
#define CR_PS_P2        0x80    /* Register Page 2 */

/* Bits in dp_isr */
#define ISR_MASK        0x3F
#define ISR_PRX         0x01    /* Packet Received with no errors */
#define ISR_PTX         0x02    /* Packet Transmitted with no errors */
#define ISR_RXE         0x04    /* Receive Error */
#define ISR_TXE         0x08    /* Transmit Error */
#define ISR_OVW         0x10    /* Overwrite Warning */
#define ISR_CNT         0x20    /* Counter Overflow */
#define ISR_RDC         0x40    /* Remote DMA Complete */
#define ISR_RST         0x80    /* Reset Status */

/* Bits in dp_imr */
#define IMR_PRXE        0x01    /* Packet Received Enable */
#define IMR_PTXE        0x02    /* Packet Transmitted Enable */
#define IMR_RXEE        0x04    /* Receive Error Enable */
#define IMR_TXEE        0x08    /* Transmit Error Enable */
#define IMR_OVWE        0x10    /* Overwrite Warning Enable */
#define IMR_CNTE        0x20    /* Counter Overflow Enable */
#define IMR_RDCE        0x40    /* DMA Complete Enable */

/* Bits in dp_dcr */
#define DCR_WTS         0x01    /* Word Transfer Select */
#define DCR_BYTEWIDE    0x00    /* WTS: byte wide transfers */
#define DCR_WORDWIDE    0x01    /* WTS: word wide transfers */
#define DCR_BOS         0x02    /* Byte Order Select */
#define DCR_LITTLENDIAN 0x00    /* BOS: Little Endian */
#define DCR_BIGENDIAN   0x02    /* BOS: Big Endian */
#define DCR_LAS         0x04    /* Long Address Select */
#define DCR_BMS         0x08    /* Burst Mode Select */
#define DCR_AR          0x10    /* Autoinitialize Remote */
#define DCR_FTS         0x60    /* Fifo Threshold Select */
#define DCR_2BYTES      0x00    /* Fifo Threshold: 2 bytes */
#define DCR_4BYTES      0x20    /* Fifo Threshold: 4 bytes */
#define DCR_8BYTES      0x40    /* Fifo Threshold: 8 bytes */
#define DCR_12BYTES     0x60    /* Fifo Threshold: 12 bytes */

/* Bits in dp_tcr */
#define TCR_CRC          0x01    /* Inhibit CRC */
#define TCR_ELC         0x06    /* Encoded Loopback Control */
#define TCR_NORMAL      0x00    /* ELC: Normal Operation */
#define TCR_INTERNAL     0x02    /* ELC: Internal Loopback */
#define TCR_0EXTERNAL    0x04    /* ELC: External Loopback LPBK=0 */
#define TCR_1EXTERNAL    0x06    /* ELC: External Loopback LPBK=1 */
#define TCR_ATD         0x08    /* Auto Transmit */
#define TCR_OFST        0x10    /* Collision Offset Enable */

/* Bits in dp_tsr */
#define TSR_PTX         0x01    /* Packet Transmitted (without error) */
#define TSR_DFR         0x02    /* Transmit Deferred */
#define TSR_COL         0x04    /* Transmit Collided */
#define TSR_ABT         0x08    /* Transmit Aborted */
#define TSR_CRS         0x10    /* Carrier Sense Lost */
#define TSR_FU          0x20    /* FIFO Underrun */
#define TSR_CDH         0x40    /* CD Heartbeat */
#define TSR_OWC         0x80    /* Out of Window Collision */

/* Bits in dp_rcr */
#define RCR_SEP         0x01    /* Save Errored Packets */
#define RCR_AR          0x02    /* Accept Runt Packets */
#define RCR_AB          0x04    /* Accept Broadcast */
#define RCR_AM          0x08    /* Accept Multicast */
#define RCR_PRO         0x10    /* Physical Promiscuous */
#define RCR_MON         0x20    /* Monitor Mode */

```

```
/* Bits in dp_rsr */
#define RSR_PRX      0x01    /* Packet Received Intact */
#define RSR_CRC      0x02    /* CRC Error */
#define RSR_FAE      0x04    /* Frame Alignment Error */
#define RSR_FO       0x08    /* FIFO Overrun */
#define RSR_MPA      0x10    /* Missed Packet */
#define RSR_PHY      0x20    /* Multicast Address Match !! */
#define RSR_DIS      0x40    /* Receiver Disabled */

/* Some macros to simplify accessing the dp8390 */
#define inb_reg0(dep,reg) (inb(dep->de_dp8390_port+reg))
#define outb_reg0(dep,reg,data) (outb(dep->de_dp8390_port+reg,data))
#define inb_reg1(dep,reg) (inb(dep->de_dp8390_port+reg))
#define outb_reg1(dep,reg,data) (outb(dep->de_dp8390_port+reg,data))

typedef struct dp_rcvhdr {
    u8_t dr_status;          /* Copy of rsr */
    u8_t dr_next;           /* Pointer to next packet */
    u8_t dr_rbcl;           /* Receive Byte Count Low */
    u8_t dr_rbch;          /* Receive Byte Count High */
} dp_rcvhdr_t;

void ns_init(dpeth_t *);

/** 8390.h **/
```

```
##
## Makefile for ISA ethernet drivers   May  02, 2000
##
## $Log: Makefile,v $
## Revision 1.3  2005/07/19 13:21:48  jnherder
## Renamed src/lib/utlis to src/lib/sysutil --- because of new src/lib/util
##
## Revision 1.2  2005/07/19 12:12:47  jnherder
## Changed Makefiles: drivers are now installed in /usr/sbin.
## TTY now gets SYS_EVENT message with sigset (e.g., SIGKMESS, SIGKSTOP).
##
## Revision 1.1  2005/06/29 10:16:46  beng
## Import of dpeth 3c501/3c509b/.. ethernet driver by
## Giovanni Falzoni <fgalzoni@inwind.it>.
##
## Revision 2.0  2005/06/26 16:16:46  lsodgf0
## Initial revision for Minix 3.0.6
##
## $Id: Makefile,v 1.3 2005/07/19 13:21:48 jnherder Exp $

## Programs, flags, etc.
DRIVER = dpeth

debug    = 0

CC       = exec cc
LD       = $(CC)
CPPFLAGS= -I.. -I/usr/include -Ddebug=$(debug)
CFLAGS  = -ws $(CPPFLAGS)
LDFLAGS = -i -o $@

SRCS     = 3c501.c 3c509.c 3c503.c ne.c wd.c 8390.c devio.c netbuff.c dp.c
OBJS     = 3c501.o 3c509.o 3c503.o ne.o wd.o 8390.o devio.o netbuff.o dp.o
LIBS     = -lsysutil -lsys # -ltimers

## Build rules
all build:      $(DRIVER)

$(DRIVER):      $(OBJS)
                $(CC) $(OBJS) $(LIBS) $(LDFLAGS)
                install -S 4kw $(DRIVER)

## Install with other drivers
install:        /usr/sbin/$(DRIVER)
/usr/sbin/$(DRIVER):  $(DRIVER)
                install -o root -cs $? $@

## Generate dependencies

depend:
                /usr/bin/mkdep "$(CC) -E $(CPPFLAGS)" *.c > .depend

## Clean directory
clean:
                @rm -f $(DRIVER) *.o *.BAK

include .depend

## end
```

This is my implementation of a new network task for the Minix kernel. I did it initially to handle a 3c501 board (Etherlink), but those board are so unstable that it is not worth using them except for learning how to implement a driver. When I got a 3c509b board (Etherlink III) it was easier to write the code to handle them.

The Minix code in 'dp8390.c' is too specific for the National chip set, so what I did was to remove as much as I needed of the code dependant from the chip and produce a generic task that, I hope, will be able to handle many more cards.

\$Log: README,v \$  
Revision 1.1 2005/06/29 10:16:46 beng  
Import of dpeth 3c501/3c509b/.. ethernet driver by  
Giovanni Falzoni <fgalzoni@inwind.it>.

Revision 1.3 2004/04/14 12:49:07 lsodgfo  
Changes for porting to Minix 2.0.4 run on BOCHS

Revision 1.2 2002/03/25 14:16:09 lsodgfo  
The driver for the NEx000 has been rewritten to be operational with the ACCTON 18xx (an NE1000 clone)  
The I/O routines for 16 bit cards are still untested..

Revision 1.1 2002/02/09 09:35:09 lsodgfo  
Initial revision  
The package is not fully tested, i.e. I had only 3Com boards (3c501, 3c503, 3c503/16 and 3c509b) and WD8003. I got also a NE1000 clone but it was not fully operational and I could not appreciate the results. For this reason the changes done to the interface to I/O for 8 and 16 bits are not tested.

\$Id: README,v 1.1 2005/06/29 10:16:46 beng Exp \$

```
/*
** File:          devio.c          Jun. 11, 2005
**
** Author:       Giovanni Falzoni <gfalzoni@inwind.it>
**
** This file contains the routines for readind/writing
** from/to the device registers.
**
** $Id: devio.c,v 1.3 2005/08/05 19:08:43 beng Exp $
*/

#include "drivers.h"
#include <net/gen/ether.h>
#include <net/gen/eth_io.h>
#include "dp.h"

#if (USE_IOPL == 0)

static void warning(const char *type, int err)
{
    printf("Warning: eth#0 sys_%s failed (%d)\n", type, err);
    return;
}

/*
** Name:          unsigned int inb(unsigned short int port);
** Function:      Reads a byte from specified i/o port.
*/
PUBLIC unsigned int inb(unsigned short port)
{
    unsigned int value;
    int rc;

    if ((rc = sys_inb(port, &value)) != OK) warning("inb", rc);
    return value;
}

/*
** Name:          unsigned int inw(unsigned short int port);
** Function:      Reads a word from specified i/o port.
*/
PUBLIC unsigned int inw(unsigned short port)
{
    unsigned int value;
    int rc;

    if ((rc = sys_inw(port, &value)) != OK) warning("inw", rc);
    return value;
}

/*
** Name:          unsigned int insb(unsigned short int port, int proc_nr, void *buffer, int
count);
** Function:      Reads a sequence of bytes from specified i/o port to user space buffer.
*/
PUBLIC void insb(unsigned short int port, int proc_nr, void *buffer, int count)
{
    int rc;

    if ((rc = sys_insb(port, proc_nr, buffer, count)) != OK)
        warning("insb", rc);
    return;
}

/*
** Name:          unsigned int insw(unsigned short int port, int proc_nr, void *buffer, int
count);
** Function:      Reads a sequence of words from specified i/o port to user space buffer.
*/
PUBLIC void insw(unsigned short int port, int proc_nr, void *buffer, int count)
{
    int rc;
```

```
    if ((rc = sys_insw(port, proc_nr, buffer, count)) != OK)
        warning("insw", rc);
    return;
}

/*
** Name:          void outb(unsigned short int port, unsigned long value);
** Function:      Writes a byte to specified i/o port.
*/
PUBLIC void outb(unsigned short port, unsigned long value)
{
    int rc;

    if ((rc = sys_outb(port, value)) != OK) warning("outb", rc);
    return;
}

/*
** Name:          void outw(unsigned short int port, unsigned long value);
** Function:      Writes a word to specified i/o port.
*/
PUBLIC void outw(unsigned short port, unsigned long value)
{
    int rc;

    if ((rc = sys_outw(port, value)) != OK) warning("outw", rc);
    return;
}

/*
** Name:          void outsb(unsigned short int port, int proc_nr, void *buffer, int count)
** Function:      Writes a sequence of bytes from user space to specified i/o port.
*/
PUBLIC void outsb(unsigned short port, int proc_nr, void *buffer, int count)
{
    int rc;

    if ((rc = sys_outsb(port, proc_nr, buffer, count)) != OK)
        warning("outsb", rc);
    return;
}

/*
** Name:          void outsw(unsigned short int port, int proc_nr, void *buffer, int count)
** Function:      Writes a sequence of bytes from user space to specified i/o port.
*/
PUBLIC void outsw(unsigned short port, int proc_nr, void *buffer, int count)
{
    int rc;

    if ((rc = sys_outsw(port, proc_nr, buffer, count)) != OK)
        warning("outsw", rc);
    return;
}

#else
#error To be implemented
#endif
/* USE_IOPL */
/** devio.c */
```

```

/*
** File:      eth.c   Version 1.00,   Jan. 14, 1997
**
** Author:    Giovanni Falzoni <gfalzoni@inwind.it>
**
** This file contains the ethernet device driver main task.
** It has to be integrated with the board specific drivers.
** It is a rewriting of Minix 2.0.0 ethernet driver (dp8390.c)
** to remove bord specific code. It should operate (I hope)
** with any board driver.
**
** The valid messages and their parameters are:
**
**      m_type      DL_PORT   DL_PROC   DL_COUNT DL_MODE DL_ADDR
**      +-----+-----+-----+-----+-----+-----+
**      | HARD_INT   |         |         |         |         |         |
**      +-----+-----+-----+-----+-----+-----+
**      | SYN_ALARM  |         |         |         |         |         |
**      +-----+-----+-----+-----+-----+-----+
**      | HARD_STOP  |         |         |         |         |         |
**      +-----+-----+-----+-----+-----+-----+
**      | FKEY_PRESSED         |         |         |         |         | (99)
**      +-----+-----+-----+-----+-----+-----+
**      | DL_WRITE   | port nr | proc nr | count  | mode   | address | (3)
**      +-----+-----+-----+-----+-----+-----+
**      | DL_WRITEV  | port nr | proc nr | count  | mode   | address | (4)
**      +-----+-----+-----+-----+-----+-----+
**      | DL_READ    | port nr | proc nr | count  |         | address | (5)
**      +-----+-----+-----+-----+-----+-----+
**      | DL_READV   | port nr | proc nr | count  |         | address | (6)
**      +-----+-----+-----+-----+-----+-----+
**      | DL_INIT    | port nr | proc nr |         | mode   | address | (7)
**      +-----+-----+-----+-----+-----+-----+
**      | DL_STOP    | port_nr |         |         |         |         | (8)
**      +-----+-----+-----+-----+-----+-----+
**      | DL_GETSTAT | port nr | proc nr |         |         | address | (9)
**      +-----+-----+-----+-----+-----+-----+
**
** The messages sent are:
**
**      m-type      DL_PORT   DL_PROC   DL_COUNT   DL_STAT   DL_CLICK
**      +-----+-----+-----+-----+-----+-----+
**      |DL_TASK_REPL| port nr | proc nr |rd-count| err|stat| clock  | (21)
**      +-----+-----+-----+-----+-----+-----+
**
**      m_type      m3_i1      m3_i2      m3_ca1
**      +-----+-----+-----+-----+
**      |DL_INIT_REPL| port nr |last port| ethernet addr | (20)
**      +-----+-----+-----+-----+
**
** $Id: dp.c,v 1.13 2006/03/25 04:49:03 beng Exp $
*/

#include "drivers.h"
#include <minix/keymap.h>
#include <net/hton.h>
#include <net/gen/ether.h>
#include <net/gen/eth_io.h>

#include "dp.h"

/*
** Local data
**/
extern int errno;
static dpeth_t de_table[DE_PORT_NR];
static char *progname;

typedef struct dp_conf {          /* Configuration description structure */
    port_t dpc_port;
    int dpc_irq;
    phys_bytes dpc_mem;
    char *dpc_envvar;
} dp_conf_t;

```

```

/* Device default configuration */
static dp_conf_t dp_conf[DE_PORT_NR] = {
    /* I/O port, IRQ, Buff addr, Env. var, Buf. selector */
    { 0x300, 5, 0xC8000, "DPETH0", },
    { 0x280, 10, 0xCC000, "DPETH1", },
};

static const char CopyErrMsg[] = "unable to read/write user data";
static const char PortErrMsg[] = "illegal port";
static const char RecvErrMsg[] = "receive failed";
static const char SendErrMsg[] = "send failed";
static const char SizeErrMsg[] = "illegal packet size";
static const char TypeErrMsg[] = "illegal message type";
static const char DevName[] = "eth#?";

static void do_getname(message *mp);

/*
** Name:      void reply(dpeth_t *dep, int err)
** Function:  Fills a DL_TASK_REPLY reply message and sends it.
*/
static void reply(dpeth_t * dep, int err)
{
    message reply;
    int status = FALSE;

    if (dep->de_flags & DEF_ACK_SEND) status |= DL_PACK_SEND;
    if (dep->de_flags & DEF_ACK_RECV) status |= DL_PACK_RECV;

    reply.m_type = DL_TASK_REPLY;
    reply.DL_PORT = dep - de_table;
    reply.DL_PROC = dep->de_client;
    reply.DL_STAT = status /* / ((u32_t) err << 16) */;
    reply.DL_COUNT = dep->de_read_s;
    getuptime(&reply.DL_CLCK);

    DEBUG(sprintf("\treply %d (%ld)\n", reply.m_type, reply.DL_STAT));

    if ((status = send(dep->de_client, &reply)) == OK) {
        dep->de_read_s = 0;
        dep->de_flags &= NOT(DEF_ACK_SEND | DEF_ACK_RECV);
    } else if (status != ELOCKED || err == OK)
        panic(dep->de_name, SendErrMsg, status);

    return;
}

/*
** Name:      void dp_confaddr(dpeth_t *dep)
** Function:  Checks environment for a User defined ethernet address.
*/
static void dp_confaddr(dpeth_t * dep)
{
    static char ea_fmt[] = "x:x:x:x:x";
    char ea_key[16];
    int ix;
    long val;

    strcpy(ea_key, dp_conf[dep - de_table].dpc_envvar);
    strcat(ea_key, "_EA");

    for (ix = 0; ix < SA_ADDR_LEN; ix++) {
        val = dep->de_address.ea_addr[ix];
        if (env_parse(ea_key, ea_fmt, ix, &val, 0x00L, 0xFFL) != EP_SET)
            break;
        dep->de_address.ea_addr[ix] = val;
    }

    if (ix != 0 && ix != SA_ADDR_LEN)
        /* It's all or nothing, force a panic */
        env_parse(ea_key, "?", 0, &val, 0L, 0L);

    return;
}

```



```

}

/*
** Name:          void update_conf(dpeth_t *dep, dp_conf_t *dcp)
** Function:      Gets the default settings from 'dp_conf' table and
**               modifies them from the environment.
*/
static void update_conf(dpeth_t * dep, dp_conf_t * dcp)
{
    static char dpc_fmt[] = "x:d:x";
    long val;

    dep->de_mode = DEM_SINK;
    val = dcp->dpc_port;          /* Get I/O port address */
    switch (env_parse(dcp->dpc_envvar, dpc_fmt, 0, &val, 0x000L, 0x3FFL)) {
        case EP_OFF:             dep->de_mode = DEM_DISABLED;      break;
        case EP_ON:              dep->de_mode = DEM_ENABLED;       break;
        case EP_SET:             dep->de_mode = DEM_ENABLED;       break;
    }
    dep->de_base_port = val;

    val = dcp->dpc_irq | DEI_DEFAULT; /* Get Interrupt line (IRQ) */
    env_parse(dcp->dpc_envvar, dpc_fmt, 1, &val, 0L, (long) NR_IRQ_VECTORS - 1);
    dep->de_irq = val;

    val = dcp->dpc_mem;             /* Get shared memory address */
    env_parse(dcp->dpc_envvar, dpc_fmt, 2, &val, 0L, LONG_MAX);
    dep->de_linmem = val;

    return;
}

/*
** Name:          void do_dump(message *mp)
** Function:      Displays statistics on screen (SFx key from console)
*/
static void do_dump(message *mp)
{
    dpeth_t *dep;
    int port;

    printf("\n\n");
    for (port = 0, dep = de_table; port < DE_PORT_NR; port += 1, dep += 1) {

        if (dep->de_mode == DEM_DISABLED) continue;

        printf("%s statistics:\t\t", dep->de_name);

        /* Network interface status */
        printf("Status: 0x%04x (%d)\n\n", dep->de_flags, dep->de_int_pending);

        (*dep->de_dumpstatsf) (dep);

        /* Transmitted/received bytes */
        printf("Tx bytes: %10ld\t", dep->bytes_Tx);
        printf("Rx bytes: %10ld\n", dep->bytes_Rx);

        /* Transmitted/received packets */
        printf("Tx OK:   %8ld\t", dep->de_stat.ets_packetT);
        printf("Rx OK:   %8ld\n", dep->de_stat.ets_packetR);

        /* Transmit/receive errors */
        printf("Tx Err:  %8ld\t", dep->de_stat.ets_sendErr);
        printf("Rx Err:  %8ld\n", dep->de_stat.ets_recvErr);

        /* Transmit unnerruns/receive overrruns */
        printf("Tx Und:  %8ld\t", dep->de_stat.ets_fifoUnder);
        printf("Rx Ovr:  %8ld\n", dep->de_stat.ets_fifoOver);

        /* Transmit collisions/receive CRC errors */
        printf("Tx Coll: %8ld\t", dep->de_stat.ets_collision);
        printf("Rx CRC:  %8ld\n", dep->de_stat.ets_CRCerr);
    }
    return;
}

```

```

}

/*
** Name:          void get_userdata(int user_proc, vir_bytes user_addr, int count, void *loc
c_addr)
** Function:      Copies data from user area.
*/
static void get_userdata(int user_proc, vir_bytes user_addr, int count, void *loc_addr)
{
    int rc;
    vir_bytes len;

    len = (count > IOVEC_NR ? IOVEC_NR : count) * sizeof(iovec_t);
    if ((rc = sys_datacopy(user_proc, user_addr, SELF, (vir_bytes)loc_addr, len)) != OK)
        panic(DevName, CopyErrMsg, rc);
    return;
}

/*
** Name:          void do_first_init(dpeth_t *dep, dp_conf_t *dcp);
** Function:      Init action to setup task
*/
static void do_first_init(dpeth_t *dep, dp_conf_t *dcp)
{
    if (dep->de_linmem != 0) {
        dep->de_memsegm = BIOS_SEG;
        /* phys2seg(&dep->de_memsegm, &dep->de_memoffs, dep->de_linmem); */
    } else
        dep->de_linmem = 0xFFFF0000;

    /* Make sure statistics are cleared */
    memset((void *) &(dep->de_stat), 0, sizeof(eth_stat_t));

    /* Device specific initialization */
    (*dep->de_initf) (dep);

    /* Set the interrupt handler policy. Request interrupts not to be reenabled
     * automatically. Return the IRQ line number when an interrupt occurs.
     */
    dep->de_hook = dep->de_irq;
    sys_irqsetpolicy(dep->de_irq, 0 /*IRQ_REENABLE*/, &dep->de_hook);
    dep->de_int_pending = FALSE;
    sys_irqenable(&dep->de_hook);

    return;
}

/*
** Name:          void do_init(message *mp)
** Function:      Checks for hardware presence.
**               Provides initialization of hardware and data structures
*/
static void do_init(message * mp)
{
    int port;
    dpeth_t *dep;
    dp_conf_t *dcp;
    message reply_mess;

    port = mp->DL_PORT;
    if (port >= 0 && port < DE_PORT_NR) {

        dep = &de_table[port];
        dcp = &dp_conf[port];
        strcpy(dep->de_name, DevName);
        dep->de_name[4] = '0' + port;

        if (dep->de_mode == DEM_DISABLED) {

            update_conf(dep, dcp); /* First time thru */
            if (dep->de_mode == DEM_ENABLED &&
                !ell_probe(dep) && /* Probe for 3c501 */
                !wdeth_probe(dep) && /* Probe for WD80x3 */

```

```

        !ne_probe(dep) && /* Probe for NEx000 */
        !el2_probe(dep) && /* Probe for 3c503 */
        !el3_probe(dep)) { /* Probe for 3c509 */
            printf("%s: warning no ethernet card found at 0x%04X\n",
                dep->de_name, dep->de_base_port);
            dep->de_mode = DEM_DISABLED;
        }
    }

    /* 'de_mode' may change if probe routines fail, test again */
    switch (dep->de_mode) {

        case DEM_DISABLED:
            /* Device is configured OFF or hardware probe failed */
            port = ENXIO;
            break;

        case DEM_ENABLED:
            /* Device is present and probed */
            if (dep->de_flags == DEF_EMPTY) {
                /* These actions only the first time */
                do_first_init(dep, dcp);
                dep->de_flags |= DEF_ENABLED;
            }
            dep->de_flags &= NOT(DEF_PROMISC | DEF_MULTI | DEF_BROAD);
            if (mp->DL_MODE & DL_PROMISC_REQ)
                dep->de_flags |= DEF_PROMISC | DEF_MULTI | DEF_BROAD;
            if (mp->DL_MODE & DL_MULTI_REQ) dep->de_flags |= DEF_MULTI;
            if (mp->DL_MODE & DL_BROAD_REQ) dep->de_flags |= DEF_BROAD;
            (*dep->de_flagsf) (dep);
            dep->de_client = mp->m_source;
            break;

        case DEM_SINK:
            /* Device not present (sink mode) */
            memset(dep->de_address.ea_addr, 0, sizeof(ether_addr_t));
            dp_confaddr(dep); /* Station address from env. */
            break;

        default:      break;
    }
    *(ether_addr_t *) reply_mess.m3_cal = dep->de_address;

} else /* Port number is out of range */
    port = ENXIO;

reply_mess.m_type = DL_INIT_REPLY;
reply_mess.m3_i1 = port;
reply_mess.m3_i2 = DE_PORT_NR;
DEBUG(printf("\treply %d\n", reply_mess.m_type));
if (send(mp->m_source, &reply_mess) != OK) /* Can't send */
    panic(dep->de_name, SendErrMsg, mp->m_source);

return;
}

/*
** Name:      void dp_next_iovec(iovec_dat_t *iovp)
** Function:  Retrieves data from next iovec element.
**/
PUBLIC void dp_next_iovec(iovec_dat_t * iovp)
{
    iovp->iod_iovec_s -= IOVEC_NR;
    iovp->iod_iovec_addr += IOVEC_NR * sizeof(iovec_t);
    get_userdata(iovp->iod_proc_nr, iovp->iod_iovec_addr,
        iovp->iod_iovec_s, iovp->iod_iovec);
    return;
}

/*
** Name:      int calc_iovec_size(iovec_dat_t *iovp)
** Function:  Compute the size of a request.
**/

```

```

static int calc_iovec_size(iovec_dat_t * iovp)
{
    int size, ix;

    size = ix = 0;
    do {
        size += iovp->iod_iovec[ix].iov_size;
        if (++ix >= IOVEC_NR) {
            dp_next_iovec(iovp);
            ix = 0;
        }

        /* Till all vectors added */
    } while (ix < iovp->iod_iovec_s);
    return size;
}

/*
** Name:          void do_vwrite(message *mp, int vectored)
** Function:
*/
static void do_vwrite(message * mp, int vectored)
{
    int port, size;
    dpeth_t *dep;

    port = mp->DL_PORT;
    if (port < 0 || port >= DE_PORT_NR) /* Check for illegal port number */
        panic(dep->de_name, PortErrMsg, port);

    dep = &de_table[port];
    dep->de_client = mp->DL_PROC;

    if (dep->de_mode == DEM_ENABLED) {

        if (dep->de_flags & DEF_SENDING) /* Is sending in progress? */
            panic(dep->de_name, "send already in progress ", NO_NUM);

        dep->de_write_iovec.iod_proc_nr = mp->DL_PROC;
        if (vectored) {
            get_userdata(mp->DL_PROC, (vir_bytes) mp->DL_ADDR,
                mp->DL_COUNT, dep->de_write_iovec.iod_iovec);
            dep->de_write_iovec.iod_iovec_s = mp->DL_COUNT;
            dep->de_write_iovec.iod_iovec_addr = (vir_bytes) mp->DL_ADDR;
            size = calc_iovec_size(&dep->de_write_iovec);
        } else {
            dep->de_write_iovec.iod_iovec[0].iov_addr = (vir_bytes) mp->DL_ADDR;
            dep->de_write_iovec.iod_iovec[0].iov_size = size = mp->DL_COUNT;
            dep->de_write_iovec.iod_iovec_s = 1;
            dep->de_write_iovec.iod_iovec_addr = 0;
        }
        if (size < ETH_MIN_PACK_SIZE || size > ETH_MAX_PACK_SIZE)
            panic(dep->de_name, SizeErrMsg, size);

        dep->de_flags |= DEF_SENDING;
        (*dep->de_sendf) (dep, FALSE, size);

    } else if (dep->de_mode == DEM_SINK)
        dep->de_flags |= DEF_ACK_SEND;

    reply(dep, OK);
    return;
}

/*
** Name:          void do_vread(message *mp, int vectored)
** Function:
*/
static void do_vread(message * mp, int vectored)
{
    int port, size;
    dpeth_t *dep;

    port = mp->DL_PORT;

```

```

if (port < 0 || port >= DE_PORT_NR) /* Check for illegal port number */
    panic(dep->de_name, PortErrMsg, port);

dep = &de_table[port];
dep->de_client = mp->DL_PROC;

if (dep->de_mode == DEM_ENABLED) {

    if (dep->de_flags & DEF_READING) /* Reading in progress */
        panic(dep->de_name, "read already in progress", NO_NUM);

    dep->de_read_iovec.iod_proc_nr = mp->DL_PROC;
    if (vectored) {
        get_userdata(mp->DL_PROC, (vir_bytes) mp->DL_ADDR,
            mp->DL_COUNT, dep->de_read_iovec.iod_iovec);
        dep->de_read_iovec.iod_iovec_s = mp->DL_COUNT;
        dep->de_read_iovec.iod_iovec_addr = (vir_bytes) mp->DL_ADDR;
        size = calc_iovec_size(&dep->de_read_iovec);
    } else {
        dep->de_read_iovec.iod_iovec[0].iov_addr = (vir_bytes) mp->DL_ADDR;
        dep->de_read_iovec.iod_iovec[0].iov_size = size = mp->DL_COUNT;
        dep->de_read_iovec.iod_iovec_s = 1;
        dep->de_read_iovec.iod_iovec_addr = 0;
    }
    if (size < ETH_MAX_PACK_SIZE) panic(dep->de_name, SizeErrMsg, size);

    dep->de_flags |= DEF_READING;
    (*dep->de_rcvf) (dep, FALSE, size);
#if 0
    if ((dep->de_flags & (DEF_READING | DEF_STOPPED)) == (DEF_READING | DEF_STOPPED))
        /* The chip is stopped, and all arrived packets delivered */
        (*dep->de_resetf) (dep);
    dep->de_flags &= NOT(DEF_STOPPED);
#endif
    }
    reply(dep, OK);
    return;
}

/*
** Name:      void do_getstat(message *mp)
** Function:  Reports device statistics.
*/
static void do_getstat(message * mp)
{
    int port, rc;
    dpeth_t *dep;

    port = mp->DL_PORT;
    if (port < 0 || port >= DE_PORT_NR) /* Check for illegal port number */
        panic(dep->de_name, PortErrMsg, port);

    dep = &de_table[port];
    dep->de_client = mp->DL_PROC;

    if (dep->de_mode == DEM_ENABLED) (*dep->de_getstatsf) (dep);
    if ((rc = sys_datacopy(SELF, (vir_bytes)&dep->de_stat,
        mp->DL_PROC, (vir_bytes)mp->DL_ADDR,
        (vir_bytes) sizeof(dep->de_stat))) != OK)
        panic(DevName, CopyErrMsg, rc);
    reply(dep, OK);
    return;
}

static void do_getname(mp)
message *mp;
{
    int r;

    strncpy(mp->DL_NAME, progname, sizeof(mp->DL_NAME));
    mp->DL_NAME[sizeof(mp->DL_NAME)-1] = '\0';
    mp->m_type= DL_NAME_REPLY;
    r= send(mp->m_source, mp);
    if (r != OK)

```

```

        panic("dpeth", "do_getname: send failed: %d\n", r);
    }

/*
** Name:      void do_stop(message *mp)
** Function:  Stops network interface.
*/
static void do_stop(message * mp)
{
    int port;
    dpeth_t *dep;

    port = mp->DL_PORT;
    if (port < 0 || port >= DE_PORT_NR) /* Check for illegal port number */
        panic(dep->de_name, PortErrMsg, port);

    dep = &de_table[port];
    if (dep->de_mode == DEM_ENABLED && (dep->de_flags & DEF_ENABLED)) {

        /* Stop device */
        (dep->de_stopf) (dep);
        dep->de_flags = DEF_EMPTY;
        dep->de_mode = DEM_DISABLED;
    }
    return;
}

static void do_watchdog(void *message)
{
    DEBUG(printf("\t no reply"));
    return;
}

/*
** Name:      int dpeth_task(void)
** Function:  Main entry for dp task
*/
PUBLIC int main(int argc, char **argv)
{
    message m;
    dpeth_t *dep;
    int rc, fkeys, sfkeys, tasknr;

    (progname=strrchr(argv[0], '/')) ? progname++ : (progname=argv[0]);

    env_setargs(argc, argv);

    /* Request function key for debug dumps */
    fkeys = sfkeys = 0; bit_set(sfkeys, 8);
    if ((fkey_map(&fkeys, &sfkeys)) != OK)
        printf("%s: couldn't program Shift+F8 key (%d)\n", DevName, errno);

#ifdef ETH_IGN_PROTO
    {
        static ul6_t eth_ign_proto = 0;
        long val;
        val = 0xFFFF;
        env_parse("ETH_IGN_PROTO", "x", 0, &val, 0x0000L, 0xFFFFL);
        eth_ign_proto = htons((ul6_t) val);
    }
#endif

    /* Try to notify inet that we are present (again) */
    rc = _pm_findproc("inet", &tasknr);
    if (rc == OK)
        notify(tasknr);

    while (TRUE) {
        if ((rc = receive(ANY, &m)) != OK) panic(dep->de_name, RecvErrMsg, rc);

        DEBUG(printf("eth: got message %d, ", m.m_type));

        switch (m.m_type) {

```

```

    case DEV_PING:      /* Status request from RS */
        notify(m.m_source);
        continue;
    case DL_WRITE:      /* Write message to device */
        do_vwrite(&m, FALSE);
        break;
    case DL_WRITEV:     /* Write message to device */
        do_vwrite(&m, TRUE);
        break;
    case DL_READ:      /* Read message from device */
        do_vread(&m, FALSE);
        break;
    case DL_READV:     /* Read message from device */
        do_vread(&m, TRUE);
        break;
    case DL_INIT:      /* Initialize device */
        do_init(&m);
        break;
    case DL_GETSTAT:   /* Get device statistics */
        do_getstat(&m);
        break;
    case DL_GETNAME:   /* Get device name */
        do_getname(&m);
        break;
    case SYN_ALARM:    /* to be defined */
        do_watchdog(&m);
        break;
    case DL_STOP:      /* Stop device */
        do_stop(&m);
        break;
    case SYS_SIG: {
        sigset_t sigset = m.NOTIFY_ARG;
        if (sigismember(&sigset, SIGKSTOP)) { /* Shut down */
            for (rc = 0; rc < DE_PORT_NR; rc += 1) {
                if (de_table[rc].de_mode == DEM_ENABLED) {
                    m.m_type = DL_STOP;
                    m.DL_PORT = rc;
                    do_stop(&m);
                }
            }
        }
        break;
    }
    case HARD_INT:     /* Interrupt from device */
        for (dep = de_table; dep < &de_table[DE_PORT_NR]; dep += 1) {
            /* If device is enabled and interrupt pending */
            if (dep->de_mode == DEM_ENABLED) {
                dep->de_int_pending = TRUE;
                (*dep->de_interruptf) (dep);
                if (dep->de_flags & (DEF_ACK_SEND | DEF_ACK_RECV))
                    reply(dep, !OK);
                dep->de_int_pending = FALSE;
                sys_irqenable(&dep->de_hook);
            }
        }
        break;
    case FKEY_PRESSED: /* Function key pressed */
        do_dump(&m);
        break;
    case PROC_EVENT:
        break;
    default:            /* Invalid message type */
        panic(DevName, TypeErrMsg, m.m_type);
        break;
}
}
return OK;             /* Never reached, but keeps compiler happy */
}

/** dp.c **/

```

```

/*
** File:          eth.h   Version 1.00,   Jan. 14, 1997
**
** Author:       Giovanni Falzoni <gfalzoni@inwind.it>
**
** Interface description for ethernet device driver
**
** $Log: dp.h,v $
** Revision 1.4  2005/09/04 18:52:16  beng
** Giovanni's fixes to dpeth:
** Date: Sat, 03 Sep 2005 11:05:22 +0200
** Subject: Minix 3.0.8
**
** Revision 1.3  2005/08/03 11:53:34  jnherder
** Miscellaneous cleanups.
**
** Revision 1.2  2005/08/02 15:30:35  jnherder
** Various updates to support dynamically starting drivers.
** Output during initialization should be suppressed. Unless an error occurs.
** Note that main() can now be main(int argc, char **argv) and arguments can
** be passed when bringing up the driver.
**
** Revision 1.1  2005/06/29 10:16:46  beng
** Import of dpeth 3c501/3c509b/.. ethernet driver by
** Giovanni Falzoni <fgalzoni@inwind.it>.
**
** Revision 2.0  2005/06/26 16:16:46  lsodgf0
** Initial revision for Minix 3.0.6
**
** $Id: dp.h,v 1.4 2005/09/04 18:52:16 beng Exp $
*/

#undef  ENABLE_3C501
#undef  ENABLE_3C503
#undef  ENABLE_3C509
#undef  ENABLE_NE2000
#undef  ENABLE_WDETH
#undef  ENABLE_DP8390

#define ENABLE_3C501      1      /* enable 3Com Etherlink I board      */
#define ENABLE_3C503      1      /* enable 3Com Etherlink II board     */
#define ENABLE_3C509      1      /* enable 3Com Etherlink III board    */
#define ENABLE_NE2000     1      /* enable Novell N2000 board           */
#define ENABLE_WDETH      1      /* enable Western Digital WD80x3      */

#define ENABLE_DP8390     (ENABLE_3C503|ENABLE_WDETH|ENABLE_NE2000)
#define HAVE_BUFFERS      (ENABLE_3C501|ENABLE_3C509)

#undef  NULL
#define NULL ((void *)0)
#define NOT(x) (~(x))

#if debug == 1
#   define DEBUG(statm) statm
#else
#   define DEBUG(statm)
#endif

typedef struct _m_hdr_t {          /* Buffer handling header */
    struct _m_hdr_t *next;
    int size;
} m_hdr_t;

typedef struct _buff_t {          /* Receive/Transmit buffer header */
    struct _buff_t *next;
    int size;
    int client;
    char buffer[2];
} buff_t;

struct dpeth;
struct iovec_dat;
typedef void (*dp_eth_t) (struct dpeth *);
typedef void (*dp_send_rcv_t) (struct dpeth *, int, int);

```



```

#if ENABLE_DP8390 == 1
typedef void (*dp_user2nicf_t) (struct dpeth *, int, int);
typedef void (*dp_nic2userf_t) (struct dpeth *, int, int);
typedef void (*dp_getblock_t) (struct dpeth *, ul6_t, int, void *);
#endif

#define DE_PORT_NR      2      /* Number of devices supported */
#define SENDQ_NR      2      /* Size of the send queue */
#define IOVEC_NR      16      /* Number of IOVEC entries at a time */

typedef struct iovec_dat {
    iovec_t iod_iovec[IOVEC_NR];
    int iod_iovec_s;
    int iod_proc_nr;
    vir_bytes iod_iovec_addr;
} iovec_dat_t;

typedef struct dpeth {
    /* The de_base_port field is the starting point of the probe. The
     * conf routine also fills de_linmem and de_irq. If the probe routine
     * knows the irq and/or memory address because they are hardwired in
     * the board, the probe should modify these fields. Futhermore, the
     * probe routine should also fill in de_initf and de_stopf fields
     * with the appropriate function pointers and set de_prog_IO iff
     * programmed I/O is to be used.
     *
     * The initf function fills the following fields. Only cards that do
     * programmed I/O fill in the de_data_port field. In addition, the
     * init routine has to fill in the sendq data structures. */

    /* Board hardware interface */
    port_t de_base_port;
    port_t de_data_port;          /* For boards using Prog. I/O for xmit/recv */

    int de_irq;
    int de_int_pending;
    int de_hook;                  /* interrupt hook at kernel */

    char de_name[8];

#define DEI_DEFAULT      0x8000

    phys_bytes de_linmem;        /* For boards using shared memory */
    unsigned short de_memsegm;
    vir_bytes de_memoffs;
    int de_ramsize;              /* Size of on board memory */
    int de_offset_page;          /* Offset of shared memory page */

    /* Board specific functions */
    dp_eth_t de_initf;
    dp_eth_t de_stopf;
    dp_eth_t de_resetf;
    dp_eth_t de_flagsf;
    dp_eth_t de_getstatsf;
    dp_eth_t de_dumpstatsf;
    dp_eth_t de_interruptf;
    dp_send_rcv_t de_rcvf;
    dp_send_rcv_t de_sendf;

    ether_addr_t de_address;      /* Ethernet Address */
    eth_stat_t de_stat;           /* Ethernet Statistics */
    unsigned long bytes_Tx;        /* Total bytes sent/received */
    unsigned long bytes_Rx;

#define SA_ADDR_LEN      sizeof(ether_addr_t)

    int de_flags;                 /* Send/Receive mode (Configuration) */

#define DEF_EMPTY      0x0000
#define DEF_READING    0x0001
#define DEF_RECV_BUSY  0x0002
#define DEF_ACK_RECV   0x0004
#define DEF_SENDING    0x0010

```

```

#define DEF_XMIT_BUSY      0x0020
#define DEF_ACK_SEND      0x0040
#define DEF_PROMISC       0x0100
#define DEF_MULTI         0x0200
#define DEF_BROAD         0x0400
#define DEF_ENABLED       0x2000
#define DEF_STOPPED       0x4000

    int de_mode;                /* Status of the Interface */

#define DEM_DISABLED      0x0000
#define DEM_SINK         0x0001
#define DEM_ENABLED      0x0002

    /* Temporary storage for RECV/SEND requests */
    iovec_dat_t de_read_iovec;
    iovec_dat_t de_write_iovec;
    vir_bytes de_read_s;
    vir_bytes de_send_s;
    int de_client;
/*
    message de_sendmsg;
    iovec_dat_t de_tmp_iovec;
*/
#if ENABLE_DP8390 == 1
    /* For use by NS DP8390 driver */
    port_t de_dp8390_port;
    int de_prog_IO;
    int de_l6bit;
    int de_startpage;
    int de_stoppage;

    /* Do it yourself send queue */
    struct sendq {
        int sq_filled;           /* This buffer contains a packet */
        int sq_size;            /* with this size */
        int sq_sendpage;        /* starting page of the buffer */
    } de_sendq[SENDQ_NR];
    int de_sendq_nr;
    int de_sendq_head;          /* Enqueue at the head */
    int de_sendq_tail;          /* Dequeue at the tail */

    dp_user2nicf_t de_user2nicf;
    dp_nic2userf_t de_nic2userf;
    dp_getblock_t de_getblockf;
#endif

#if ENABLE_3C509 == 1
    /* For use by 3Com Etherlink III (3c509) driver */
    port_t de_id_port;
    port_t de_if_port;
#endif

#if ENABLE_3C501 == 1 || ENABLE_3C509 == 1
    /* For use by 3Com Etherlink (3c501 and 3c509) driver */
    buff_t *de_recvq_head;
    buff_t *de_recvq_tail;
    buff_t *de_xmitq_head;
    buff_t *de_xmitq_tail;
    ul6_t de_recv_mode;
    clock_t de_xmit_start;
#endif

} dpeth_t;

/*
 *      Function definitions
 */

/* dp.c */
void dp_next_iovec(iovec_dat_t * iovp);

/* devio.c */
#if defined USE_IOPL

```

```
#include <ibm/portio.h>
#else
unsigned int inb(unsigned short int);
unsigned int inw(unsigned short int);
void insb(unsigned short int, int, void *, int);
void insw(unsigned short int, int, void *, int);
void outb(unsigned short int, unsigned long);
void outw(unsigned short int, unsigned long);
void outsb(unsigned short int, int, void *, int);
void outsw(unsigned short int, int, void *, int);
#endif

/* netbuff.c */
void *alloc_buff(dpeth_t *, int);
void free_buff(dpeth_t *, void *);
void init_buff(dpeth_t *, buff_t **);
void mem2user(dpeth_t *, buff_t *);
void user2mem(dpeth_t *, buff_t *);

/* 3c501.c */
#if ENABLE_3C501 == 1
int el1_probe(dpeth_t *);
#else
#define el1_probe(x) (0)
#endif

/* 3c503.c */
#if ENABLE_3C503 == 1
int el2_probe(dpeth_t *);
#else
#define el2_probe(x) (0)
#endif

/* 3c509.c */
#if ENABLE_3C509 == 1
int el3_probe(dpeth_t *);
#else
#define el3_probe(x) (0)
#endif

/* ne.c */
#if ENABLE_NE2000 == 1
int ne_probe(dpeth_t * dep);
#else
#define ne_probe(x) (0)
#endif

/* wd.c */
#if ENABLE_WDETH == 1
int wdeth_probe(dpeth_t * dep);
#else
#define wdeth_probe(x) (0)
#endif

#define lock()    (++dep->de_int_pending, sys_irqdisable(&dep->de_hook))
#define unlock() do{int i=(--dep->de_int_pending)?0:sys_irqenable(&dep->de_hook);}while(0)
#define milli_delay(t) tickdelay(1)

/** dp.h **/
```

```
/*
** File:      ne.c      Jun. 08, 2000
**
** Driver for the NE*000 ethernet cards and derivates.
** This file contains only the ne specific code,
** the rest is in 8390.c   Code specific for ISA bus only
**
** Created:   March 15, 1994 by Philip Homburg <philip@cs.vu.nl>
** PchId:     ne2000.c,v 1.4 1996/01/19 23:30:34 philip Exp
**
** Modified:  Jun. 08, 2000 by Giovanni Falzoni <gfalzoni@inwind.it>
**           Adapted to interface new main network task.
**
** $Id: ne.c,v 1.3 2005/08/05 19:08:43 beng Exp $
*/

#include "drivers.h"
#include <net/gen/ether.h>
#include <net/gen/eth_io.h>
#include "dp.h"

#if (ENABLE_NE2000 == 1)

#include "8390.h"
#include "ne.h"

/*
** Name:      void ne_reset(dpeth_t * dep);
** Function:   Resets the board and checks if reset cycle completes
*/
static int ne_reset(dpeth_t * dep)
{
    int count = 0;

    /* Reset the ethernet card */
    outb_ne(dep, NE_RESET, inb_ne(dep, NE_RESET));
    do {
        if (++count > 10) return FALSE; /* 20 mSecs. timeout */
        milli_delay(2);
    } while ((inb_ne(dep, DP_ISR) & ISR_RST) == 0);
    return TRUE;
}

/*
** Name:      void ne_close(dpeth_t * dep);
** Function:   Stops the board by resetting it and masking interrupts.
*/
static void ne_close(dpeth_t * dep)
{
    (void)ne_reset(dep);
    outb_ne(dep, DP_ISR, 0xFF);
    sys_irqdisable(&dep->de_hook);
    return;
}

/*
** Name:      void ne_init(dpeth_t * dep);
** Function:   Initialize the board making it ready to work.
*/
static void ne_init(dpeth_t * dep)
{
    int ix;

    dep->de_data_port = dep->de_base_port + NE_DATA;
    if (dep->de_l6bit) {
        dep->de_ramsize = NE2000_SIZE;
        dep->de_offset_page = NE2000_START / DP_PAGESIZE;
    } else {
        dep->de_ramsize = NE1000_SIZE;
        dep->de_offset_page = NE1000_START / DP_PAGESIZE;
    }

    /* Allocates two send buffers from onboard RAM */
}
```

```

dep->de_sendq_nr = SENDQ_NR;
for (ix = 0; ix < SENDQ_NR; ix += 1) {
    dep->de_sendq[ix].sq_sendpage = dep->de_offset_page + ix * SENDQ_PAGES;
}

/* Remaining onboard RAM allocated for receiving */
dep->de_startpage = dep->de_offset_page + ix * SENDQ_PAGES;
dep->de_stoppage = dep->de_offset_page + dep->de_ramsize / DP_PAGESIZE;

/* Can't override the default IRQ. */
dep->de_irq &= NOT(DEI_DEFAULT);

ns_init(dep);          /* Initialize DP controller */

printf("%s: NE%d000 (%dkB RAM) at %X:%d - ",
        dep->de_name,
        dep->de_16bit ? 2 : 1,
        dep->de_ramsize / 1024,
        dep->de_base_port, dep->de_irq);
for (ix = 0; ix < SA_ADDR_LEN; ix += 1)
    printf("%02X%c", dep->de_address.ea_addr[ix], ix < SA_ADDR_LEN - 1 ? ':' : '\n');
return;
}

/*
** Name:      int ne_probe(dpeth_t * dep);
** Function:  Probe for the presence of a NE*000 card by testing
**            whether the board is reachable through the dp8390.
**            Note that the NE1000 is an 8bit card and has a memory
**            region distinct from the 16bit NE2000.
*/
PUBLIC int ne_probe(dpeth_t * dep)
{
    int ix, wd, loc1, loc2;
    char EPROM[32];
    static const struct {
        unsigned char offset;
        unsigned char value;
    } InitSeq[] =
    {
        {
            /* Selects page 0. */
            DP_CR, (CR_NO_DMA | CR_PS_P0 | CR_STP),
        },
        {
            /* Set byte-wide access and 8 bytes burst mode. */
            DP_DCR, (DCR_8BYTES | DCR_BMS),
        },
        {
            /* Clears the count registers. */
            DP_RBCR0, 0x00, }, { DP_RBCR1, 0x00,
        },
        {
            /* Mask completion irq. */
            DP_IMR, 0x00, }, { DP_ISR, 0xFF,
        },
        {
            /* Set receiver to monitor */
            DP_RCR, RCR_MON,
        },
        {
            /* and transmitter to loopback mode. */
            DP_TCR, TCR_INTERNAL,
        },
        {
            /* Transmit 32 bytes */
            DP_RBCR0, 32, }, { DP_RBCR1, 0,
        },
        {
            /* DMA starting at 0x0000. */
            DP_RSAR0, 0x00, }, { DP_RSAR1, 0x00,
        },
        {
            /* Start board (reads) */
            DP_CR, (CR_PS_P0 | CR_DM_RR | CR_STA),
        },
    };

    dep->de_dp8390_port = dep->de_base_port + NE_DP8390;

    if ((loc1 = inb_ne(dep, NE_DP8390)) == 0xFF) return FALSE;

    /* Check if the dp8390 is really there */

```

```
outb_ne(dep, DP_CR, CR_STP | CR_NO_DMA | CR_PS_P1);
loc2 = inb_ne(dep, DP_MAR5); /* Saves one byte of the address */
outb_ne(dep, DP_MAR5, 0xFF); /* Write 0xFF to it (same offset as DP_CNTR0) */
outb_ne(dep, DP_CR, CR_NO_DMA | CR_PS_P0); /* Back to page 0 */
inb_ne(dep, DP_CNTR0); /* Reading counter resets it */
if (inb_ne(dep, DP_CNTR0) != 0) {
    outb_ne(dep, NE_DP8390, loc1); /* Try to restore modified bytes */
    outb_ne(dep, DP_TCR, loc2);
    return FALSE;
}

/* Try to reset the board */
if (ne_reset(dep) == FALSE) return FALSE;

/* Checks whether the board is 8/16bits and a real NE*000 or clone */
for (ix = 0; ix < sizeof(InitSeq)/sizeof(InitSeq[0]); ix += 1) {
    outb_ne(dep, InitSeq[ix].offset, InitSeq[ix].value);
}
for (ix = 0, wd = 1; ix < 32; ix += 2) {
    EPROM[ix + 0] = inb_ne(dep, NE_DATA);
    EPROM[ix + 1] = inb_ne(dep, NE_DATA);
    /* NE2000s and clones read same value for even and odd addresses */
    if (EPROM[ix + 0] != EPROM[ix + 1]) wd = 0;
}
if (wd == 1) { /* Normalize EPROM contents for NE2000 */
    for (ix = 0; ix < 16; ix += 1) EPROM[ix] = EPROM[ix * 2];
}
/* Real NE*000 and good clones have '0x57' at locations 14 and 15 */
if (EPROM[14] != 0x57 || EPROM[15] != 0x57) return FALSE;

/* Setup the ethernet address. */
for (ix = 0; ix < SA_ADDR_LEN; ix += 1) {
    dep->de_address.ea_addr[ix] = EPROM[ix];
}
dep->de_16bit = wd;
dep->de_linmem = 0; /* Uses Programmed I/O only */
dep->de_prog_IO = 1;
dep->de_initf = ne_init;
dep->de_stopf = ne_close;
return TRUE;
}

#endif /* ENABLE_NE2000 */

/** ne.c **/
```

```
/*
** File:          ne.h
**
** Created:       March 15, 1994 by Philip Homburg <philip@cs.vu.nl>
** $PchId: ne2000.h,v 1.2 1995/12/22 08:42:31 philip Exp $
**
** $Log: ne.h,v $
** Revision 1.1  2005/06/29 10:16:46  beng
** Import of dpeth 3c501/3c509b/.. ethernet driver by
** Giovanni Falzoni <fgalzoni@inwind.it>.
**
** Revision 2.0  2005/06/26 16:16:46  lsodgfo
** Initial revision for Minix 3.0.6
**
** $Id: ne.h,v 1.1 2005/06/29 10:16:46 beng Exp $
*/

#ifndef NE2000_H
#define NE2000_H

#define NE_DP8390      0x00
#define NE_DATA        0x10
#define NE_RESET       0x1F

#define NE1000_START   0x2000
#define NE1000_SIZE     0x2000
#define NE2000_START   0x4000
#define NE2000_SIZE     0x4000

#define inb_ne(dep, reg) (inb(dep->de_base_port+reg))
#define outb_ne(dep, reg, data) (outb(dep->de_base_port+reg, data))
#define inw_ne(dep, reg) (inw(dep->de_base_port+reg))
#define outw_ne(dep, reg, data) (outw(dep->de_base_port+reg, data))

#endif /* NE2000_H */

/** ne.h **/
```

```
/*
** File:          netbuff.c          Jun. 10, 2000
**
** Author:       Giovanni Falzoni <gfalzoni@inwind.it>
**
** This file contains specific implementation of buffering
** for network packets.
**
** $Id: netbuff.c,v 1.3 2005/08/05 19:08:43 beng Exp $
*/

#include "drivers.h"
#include <net/gen/ether.h>
#include <net/gen/eth_io.h>
#include "dp.h"

#if (HAVE_BUFFERS == 1)

static m_hdr_t *allocptr = NULL;
static char tx_rx_buff[8192];

/*
** Name:          void *alloc_buff(dpeth_t *dep, int size)
** Function:      Allocates a buffer from the common pool.
*/
PUBLIC void *alloc_buff(dpeth_t *dep, int size)
{
    m_hdr_t *ptr, *wrk = allocptr;
    int units = ((size + sizeof(m_hdr_t) - 1) / sizeof(m_hdr_t)) + 1;

    lock();
    for (ptr = wrk->next;; wrk = ptr, ptr = ptr->next) {
        if (ptr->size >= units) {
            /* Memory is available, carve requested size from pool */
            if (ptr->size == units) {
                wrk->next = ptr->next;
            } else {
                /* Get memory from top address */
                ptr->size -= units;
                ptr += ptr->size;
                ptr->size = units;
            }
            allocptr = wrk;
            unlock();
            return ptr + 1;
        }
        if (ptr == allocptr) break;
    }
    unlock();
    return NULL; /* No memory available */
}

/*
** Name:          void free_buff(dpeth_t *dep, void *blk)
** Function:      Returns a buffer to the common pool.
*/
PUBLIC void free_buff(dpeth_t *dep, void *blk)
{
    m_hdr_t *wrk, *ptr = (m_hdr_t *) blk - 1;

    lock(); /* Scan linked list for the correct place */
    for (wrk = allocptr; !(ptr > wrk && ptr < wrk->next); wrk = wrk->next)
        if (wrk >= wrk->next && (ptr > wrk || ptr < wrk->next)) break;

    /* Check if adjacent block is free and join blocks */
    if (ptr + ptr->size == wrk->next) {
        ptr->size += wrk->next->size;
        ptr->next = wrk->next->next;
    } else
        ptr->next = wrk->next;
    if (wrk + wrk->size == ptr) {
        wrk->size += ptr->size;
        wrk->next = ptr->next;
    } else

```



```

        wrk->next = ptr;
        allocptr = wrk;                /* Point allocptr to block just released */
        unlock();
        return;
    }

/*
** Name:          void init_buff(dpeth_t *dep, buff_t **tx_buff)
** Function:      Initializes driver data structures.
*/
PUBLIC void init_buff(dpeth_t *dep, buff_t **tx_buff)
{
    /* Initializes buffer pool */
    if (allocptr == NULL) {
        m_hdr_t *rx = (m_hdr_t *) tx_rx_buff;
        rx->next = allocptr = rx;
        rx->size = 0;
        rx += 1;
        rx->next = NULL;
        rx->size = (sizeof(tx_rx_buff) / sizeof(m_hdr_t)) - 1;
        free_buff(dep, rx + 1);
        dep->de_rcvq_tail = dep->de_rcvq_head = NULL;
        if (tx_buff != NULL) {
            *tx_buff = alloc_buff(dep, ETH_MAX_PACK_SIZE + sizeof(buff_t));
            (*tx_buff)->size = 0;
        }
    }
    return;                            /* Done */
}

/*
** Name:          void mem2user(dpeth_t *dep, buff_t *rxbuff);
** Function:      Copies a packet from local buffer to user area.
*/
PUBLIC void mem2user(dpeth_t *dep, buff_t *rxbuff)
{
    phys_bytes phys_user;
    int bytes, ix = 0;
    iovec_dat_t *iovp = &dep->de_read_iovec;
    int pktsize = rxbuff->size;
    char *buffer = rxbuff->buffer;

    do {                                /* Reads chunks of packet into user buffers */

        bytes = iovp->iod_iovec[ix].iov_size;    /* Size of buffer */
        if (bytes > pktsize) bytes = pktsize;

        /* Reads from Rx buffer to user area */
        sys_datacopy(SELFS, (vir_bytes)buffer, iovp->iod_proc_nr,
                    iovp->iod_iovec[ix].iov_addr, bytes);
        buffer += bytes;

        if (++ix >= IOVEC_NR) { /* Next buffer of IO vector */
            dp_next_iovec(iovp);
            ix = 0;
        }
    } /* Till packet done */
    while ((pktsize -= bytes) > 0);
    return;
}

/*
** Name:          void user2mem(dpeth_t *dep, buff_t *txbuff)
** Function:      Copies a packet from user area to local buffer.
*/
PUBLIC void user2mem(dpeth_t *dep, buff_t *txbuff)
{
    phys_bytes phys_user;
    int bytes, ix = 0;
    iovec_dat_t *iovp = &dep->de_write_iovec;
    int pktsize = txbuff->size;
    char *buffer = txbuff->buffer;

```

```
do {                                     /* Reads chunks of packet from user buffers */

    bytes = iovp->iod_iovec[ix].iov_size; /* Size of buffer */
    if (bytes > pktsize) bytes = pktsize;
    sys_datacopy(iovp->iod_proc_nr, iovp->iod_iovec[ix].iov_addr,
                 SELF, (vir_bytes)buffer, bytes);
    buffer += bytes;

    if (++ix >= IOVEC_NR) { /* Next buffer of IO vector */
        dp_next_iovec(iovp);
        ix = 0;
    }
    /* Till packet done */
} while ((pktsize -= bytes) > 0);
return;
}

#endif                                     /* HAVE_BUFFERS */

/** netbuff.c **/
```

```

/*
** File:      wd.c
**
** Driver for the Ethercard (WD80x3) and derivatives
** This file contains only the wd80x3 specific code,
** the rest is in 8390.c
**
** Created:   March 14, 1994 by Philip Homburg
** PchId:
**
** Modified:  Jun. 08, 2000 by Giovanni Falzoni <gfalzoni@inwind.it>
** Adaptation to interfacing new main network task.
**
** $Id: wd.c,v 1.4 2005/08/22 15:17:40 beng Exp $
*/

#include "drivers.h"
#include <net/gen/ether.h>
#include <net/gen/eth_io.h>
#include "dp.h"

#if (ENABLE_WDETH == 1)

#include "8390.h"
#include "wd.h"

#define WET_ETHERNET      0x01    /* Ethernet transceiver */
#define WET_STARLAN      0x02    /* Starlan transceiver */
#define WET_INTERF_CHIP  0x04    /* has a WD83C583 interface chip */
#define WET_BRD_16BIT     0x08    /* 16 bit board */
#define WET_SLT_16BIT     0x10    /* 16 bit slot */
#define WET_790           0x20    /* '790 chip */

static const int we_int_table[8] = {9, 3, 5, 7, 10, 11, 15, 4};
static const int we_790int_table[8] = {0, 9, 3, 5, 7, 10, 11, 15};

_PROTOTYPE(static void we_init, (dpeth_t * dep));
_PROTOTYPE(static void we_stop, (dpeth_t * dep));
_PROTOTYPE(static int we_aliasing, (dpeth_t * dep));
_PROTOTYPE(static int we_interface_chip, (dpeth_t * dep));
_PROTOTYPE(static int we_16bitboard, (dpeth_t * dep));
_PROTOTYPE(static int we_16bitslot, (dpeth_t * dep));
_PROTOTYPE(static int we_ultra, (dpeth_t * dep));

/*=====
*                               wdeth_probe                               *
*=====*/
int wdeth_probe(dep)
dpeth_t *dep;
{
    int sum;

    if (dep->de_linmem == 0)
        return FALSE;    /* No shared memory, so no WD board */

    sum = inb_we(dep, EPL_EA0) + inb_we(dep, EPL_EA1) +
          inb_we(dep, EPL_EA2) + inb_we(dep, EPL_EA3) +
          inb_we(dep, EPL_EA4) + inb_we(dep, EPL_EA5) +
          inb_we(dep, EPL_TLB) + inb_we(dep, EPL_CHKSUM);
    if ((sum & 0xFF) != 0xFF)
        return FALSE;    /* No ethernet board at this address */

    dep->de_initf = we_init;
    dep->de_stopf = we_stop;
    dep->de_prog_IO = 0;
    return TRUE;
}

/*=====
*                               we_init                               *
*=====*/
static void we_init(dep)
dpeth_t *dep;
{

```

```

int i, int_idx, int_nr;
int tlb, rambit, revision;
int icr, irr, hwr, b, gcr;
int we_type;
int sendq_nr;

for (i = 0; i < 6; i += 1) {
    dep->de_address.ea_addr[i] = inb_we(dep, EPL_EA0 + i);
}

dep->de_dp8390_port = dep->de_base_port + EPL_DP8390;

dep->de_16bit = 0;

we_type = 0;
we_type |= WET_ETHERNET;      /* assume ethernet */
if (we_ultra(dep)) we_type |= WET_790;
if (!we_aliasing(dep)) {
    if (we_interface_chip(dep)) we_type |= WET_INTERF_CHIP;
    if (we_16bitboard(dep)) {
        we_type |= WET_BRD_16BIT;
        if (we_16bitslot(dep)) we_type |= WET_SLT_16BIT;
    }
}
if (we_type & WET_SLT_16BIT) dep->de_16bit = 1;

/* Look at the on board ram size. */
tlb = inb_we(dep, EPL_TLB);
revision = tlb & E_TLB_REV;
rambit = tlb & E_TLB_RAM;

if (dep->de_ramsize != 0) {
    /* Size set from boot environment. */
} else if (revision < 2) {
    dep->de_ramsize = 0x2000;      /* 8K */
    if (we_type & WET_BRD_16BIT)
        dep->de_ramsize = 0x4000;      /* 16K */
    else if ((we_type & WET_INTERF_CHIP) &&
        inb_we(dep, EPL_ICR) & E_ICR_MEMBIT) {
        dep->de_ramsize = 0x8000;      /* 32K */
    }
} else {
    if (we_type & WET_BRD_16BIT) {
        /* 32K or 16K */
        dep->de_ramsize = rambit ? 0x8000 : 0x4000;
    } else {
        /* 32K or 8K */
        dep->de_ramsize = rambit ? 0x8000 : 0x2000;
    }
}

if (we_type & WET_790) {
    outb_we(dep, EPL_MSR, E_MSR_RESET);
    if ((we_type & (WET_BRD_16BIT | WET_SLT_16BIT)) ==
        (WET_BRD_16BIT | WET_SLT_16BIT)) {
        outb_we(dep, EPL_LAAR, E_LAAR_LAN16E | E_LAAR_MEM16E);
    }
} else if (we_type & WET_BRD_16BIT) {
    if (we_type & WET_SLT_16BIT) {
        outb_we(dep, EPL_LAAR, E_LAAR_A19 | E_LAAR_SOFTINT |
            E_LAAR_LAN16E | E_LAAR_MEM16E);
    } else {
        outb_we(dep, EPL_LAAR, E_LAAR_A19 | E_LAAR_SOFTINT |
            E_LAAR_LAN16E);
    }
}

if (we_type & WET_790) {
    outb_we(dep, EPL_MSR, E_MSR_MENABLE);
    hwr = inb_we(dep, EPL_790_HWR);
    outb_we(dep, EPL_790_HWR, hwr | E_790_HWR_SWH);
    b = inb_we(dep, EPL_790_B);
    outb_we(dep, EPL_790_B, ((dep->de_linmem >> 13) & 0x0f) |
        ((dep->de_linmem >> 11) & 0x40) | (b & 0xb0));
    outb_we(dep, EPL_790_HWR, hwr & ~E_790_HWR_SWH);
}

```

```

} else {
    outb_we(dep, EPL_MSR, E_MSR_RESET);
    outb_we(dep, EPL_MSR, E_MSR_MENABLE |
        ((dep->de_linmem >> 13) & E_MSR_MEMADDR));
}

if ((we_type & WET_INTERF_CHIP) && !(we_type & WET_790)) {
    icr = inb_we(dep, EPL_ICR);
    irr = inb_we(dep, EPL_IRR);
    int_indx = (icr & E_ICR_IR2) | ((irr & (E_IRR_IR0 | E_IRR_IR1)) >> 5);
    int_nr = we_int_table[int_indx];
    DEBUG(sprintf("%s: encoded irq= %d\n", dep->de_name, int_nr));
    if (dep->de_irq & DEI_DEFAULT) dep->de_irq = int_nr;
    outb_we(dep, EPL_IRR, irr | E_IRR_IEN);
}

if (we_type & WET_790) {
    hwr = inb_we(dep, EPL_790_HWR);
    outb_we(dep, EPL_790_HWR, hwr | E_790_HWR_SWH);

    gcr = inb_we(dep, EPL_790_GCR);

    outb_we(dep, EPL_790_HWR, hwr & ~E_790_HWR_SWH);

    int_indx = ((gcr & E_790_GCR_IR2) >> 4) |
        ((gcr & (E_790_GCR_IR1 | E_790_GCR_IR0)) >> 2);
    int_nr = we_790int_table[int_indx];
    DEBUG(sprintf("%s: encoded irq= %d\n", dep->de_name, int_nr));
    if (dep->de_irq & DEI_DEFAULT) dep->de_irq = int_nr;
    icr = inb_we(dep, EPL_790_ICR);
    outb_we(dep, EPL_790_ICR, icr | E_790_ICR_EIL);
}

/* Strip the "default flag." */
dep->de_irq &= ~DEI_DEFAULT;

dep->de_offset_page = 0; /* Shared memory starts at 0 */
/* Allocate one send buffer (1.5KB) per 8KB of on board memory.
sendq_nr = dep->de_ramsize / 0x2000;
if (sendq_nr < 1)
    sendq_nr = 1;
else if (sendq_nr > SENDQ_NR) */
    sendq_nr = SENDQ_NR;
dep->de_sendq_nr = sendq_nr;
for (i = 0; i < sendq_nr; i++) {
    dep->de_sendq[i].sq_sendpage = i * SENDQ_PAGES;
}
dep->de_startpage = i * SENDQ_PAGES;
dep->de_stoppage = dep->de_ramsize / DP_PAGESIZE;

ns_init(dep); /* Initialize DP controller */

printf("%s: WD80%d3 (%dkB RAM) at %X:%d:%IX - ",
    dep->de_name,
    we_type & WET_BRD_16BIT ? 1 : 0,
    dep->de_ramsize / 1024,
    dep->de_base_port,
    dep->de_irq,
    dep->de_linmem);
for (i = 0; i < SA_ADDR_LEN; i += 1)
    printf("%02X%c", dep->de_address.ea_addr[i],
        i < SA_ADDR_LEN - 1 ? ':' : '\n');

return;
}

/*=====
*
*                               we_stop
*=====*/
static void we_stop(dep)
dpeth_t *dep;
{
    if (dep->de_16bit) outb_we(dep, EPL_LAAR, E_LAAR_A19 | E_LAAR_LAN16E);
    outb_we(dep, EPL_MSR, E_MSR_RESET);

```

```

    outb_we(dep, EPL_MSR, 0);
    sys_irqdisable(&dep->de_hook);
    return;
}

/*=====
 *                               we_aliasing                               *
 *=====*/
static int we_aliasing(dep)
dpeth_t *dep;
{
    /* Determine whether wd8003 hardware performs register aliasing. This implies
     * an old WD8003E board. */

    if (inb_we(dep, EPL_REG1) != inb_we(dep, EPL_EA1)) return 0;
    if (inb_we(dep, EPL_REG2) != inb_we(dep, EPL_EA2)) return 0;
    if (inb_we(dep, EPL_REG3) != inb_we(dep, EPL_EA3)) return 0;
    if (inb_we(dep, EPL_REG4) != inb_we(dep, EPL_EA4)) return 0;
    if (inb_we(dep, EPL_REG7) != inb_we(dep, EPL_CHKSUM)) return 0;
    return 1;
}

/*=====
 *                               we_interface_chip                               *
 *=====*/
static int we_interface_chip(dep)
dpeth_t *dep;
{
    /* Determine if the board has an interface chip. */

    outb_we(dep, EPL_GP2, 0x35);
    if (inb_we(dep, EPL_GP2) != 0x35) return 0;
    outb_we(dep, EPL_GP2, 0x3A);
    if (inb_we(dep, EPL_GP2) != 0x3A) return 0;
    return 1;
}

/*=====
 *                               we_16bitboard                               *
 *=====*/
static int we_16bitboard(dep)
dpeth_t *dep;
{
    /* Determine whether the board is capable of doing 16 bit memory moves.
     * If the 16 bit enable bit is unchangable by software we'll assume an
     * 8 bit board.
     */
    int icr;
    u8_t tlb;

    icr = inb_we(dep, EPL_ICR);

    outb_we(dep, EPL_ICR, icr ^ E_ICR_16BIT);
    if (inb_we(dep, EPL_ICR) == icr) {
        tlb = inb_we(dep, EPL_TLB);

        DEBUG(sprintf("%s: tlb= 0x%x\n", dep->de_name, tlb));

        return tlb == E_TLB_EB || tlb == E_TLB_E ||
            tlb == E_TLB_SMCE || tlb == E_TLB_SMC8216C;
    }
    outb_we(dep, EPL_ICR, icr);
    return 1;
}

/*=====
 *                               we_16bitslot                               *
 *=====*/
static int we_16bitslot(dep)
dpeth_t *dep;
{
    /* Determine if the 16 bit board is plugged into a 16 bit slot. */

    return !(inb_we(dep, EPL_ICR) & E_ICR_16BIT);
}

```

```
}

/*=====*
 *                               *
 *=====*/
static int we_ultra(dep)
dpeth_t *dep;
{
/* Determine if we has an '790 chip. */
u8_t tlb;

    tlb = inb_we(dep, EPL_TLB);
    return tlb == E_TLB_SMC8216C;
}

#endif                               /* ENABLE_WDETH */

/** wd.c **/
```

```

/*
** File: wd.h
**
** Created: before Dec 28, 1992 by Philip Homburg
** $PchId: wdeth.h,v 1.4 1995/12/22 08:36:57 philip Exp $
**
** $Log: wd.h,v $
** Revision 1.2 2005/08/22 15:17:40 beng
** Remove double-blank lines (Al)
**
** Revision 1.1 2005/06/29 10:16:46 beng
** Import of dpeth 3c501/3c509b/.. ethernet driver by
** Giovanni Falzoni <fgalzoni@inwind.it>.
**
** Revision 2.0 2005/06/26 16:16:46 lsodgfo
** Initial revision for Minix 3.0.6
**
** $Id: wd.h,v 1.2 2005/08/22 15:17:40 beng Exp $
*/

#ifndef WDETH_H
#define WDETH_H

/* Western Digital Ethercard Plus, or WD8003E card. */

#define EPL_REG0 0x0 /* Control(write) and status(read) */
#define EPL_REG1 0x1
#define EPL_REG2 0x2
#define EPL_REG3 0x3
#define EPL_REG4 0x4
#define EPL_REG5 0x5
#define EPL_REG6 0x6
#define EPL_REG7 0x7
#define EPL_EA0 0x8 /* Most significant eaddr byte */
#define EPL_EA1 0x9
#define EPL_EA2 0xA
#define EPL_EA3 0xB
#define EPL_EA4 0xC
#define EPL_EA5 0xD /* Least significant eaddr byte */
#define EPL_TLB 0xE
#define EPL_CHKSUM 0xF /* sum from epl_ea0 upto here is 0xFF */
#define EPL_DP8390 0x10 /* NatSemi chip */

#define EPL_MSR EPL_REG0 /* memory select register */
#define EPL_ICR EPL_REG1 /* interface configuration register */
#define EPL_IRR EPL_REG4 /* interrupt request register (IRR) */
#define EPL_790_HWR EPL_REG4 /* '790 hardware support register */
#define EPL_LAAR EPL_REG5 /* LA address register (write only) */
#define EPL_790_ICR EPL_REG6 /* '790 interrupt control register */
#define EPL_GP2 EPL_REG7 /* general purpose register 2 */
#define EPL_790_B EPL_EA3 /* '790 memory register */
#define EPL_790_GCR EPL_EA5 /* '790 General Control Register */

/* Bits in EPL_MSR */
#define E_MSR_MEMADDR 0x3F /* Bits SA18-SA13, SA19 implicit 1 */
#define E_MSR_MENABLE 0x40 /* Memory Enable */
#define E_MSR_RESET 0x80 /* Software Reset */

/* Bits in EPL_ICR */
#define E_ICR_16BIT 0x01 /* 16 bit bus */
#define E_ICR_IR2 0x04 /* bit 2 of encoded IRQ */
#define E_ICR_MEMBIT 0x08 /* 583 mem size mask */

/* Bits in EPL_IRR */
#define E_IRR_IR0 0x20 /* bit 0 of encoded IRQ */
#define E_IRR_IR1 0x40 /* bit 1 of encoded IRQ */
#define E_IRR_IEN 0x80 /* enable interrupts */

/* Bits in EPL_LAAR */
#define E_LAAR_A19 0x01 /* address lines for above 1M ram */
#define E_LAAR_A20 0x02 /* address lines for above 1M ram */
#define E_LAAR_A21 0x04 /* address lines for above 1M ram */
#define E_LAAR_A22 0x08 /* address lines for above 1M ram */
#define E_LAAR_A23 0x10 /* address lines for above 1M ram */

```



```
#define E_LAAR_SOFTINT 0x20 /* enable software interrupt */
#define E_LAAR_LAN16E 0x40 /* enables 16 bit RAM for LAN */
#define E_LAAR_MEM16E 0x80 /* enables 16 bit RAM for host */

/* Bits and values in EPL_TLB */
#define E_TLB_EB 0x05 /* WD8013EB */
#define E_TLB_E 0x27 /* WD8013 Elite */
#define E_TLB_SMCE 0x29 /* SMC Elite 16 */
#define E_TLB_SMC8216C 0x2B /* SMC 8216 C */

#define E_TLB_REV 0x1F /* revision mask */
#define E_TLB_SOFT 0x20 /* soft config */
#define E_TLB_RAM 0x40 /* extra ram bit */

/* Bits in EPL_790_HWR */
#define E_790_HWR_SWH 0x80 /* switch register set */

/* Bits in EPL_790_ICR */
#define E_790_ICR_EIL 0x01 /* enable interrupts */

/* Bits in EPL_790_GCR when E_790_HWR_SWH is set in EPL_790_HWR */
#define E_790_GCR_IR0 0x04 /* bit 0 of encoded IRQ */
#define E_790_GCR_IR1 0x08 /* bit 1 of encoded IRQ */
#define E_790_GCR_IR2 0x40 /* bit 2 of encoded IRQ */

#define inb_we(dep, reg) (inb(dep->de_base_port+reg))
#define outb_we(dep, reg, data) (outb(dep->de_base_port+reg, data))

#endif /* WDETH_H */

/** wd.h **/
```

```
# Makefile for the floppy disk driver (FLOPPY)
DRIVER = floppy

# directories
u = /usr
i = $u/include
s = $i/sys
m = $i/minix
b = $i/ibm
d = ..

# programs, flags, etc.
MAKE = exec make
CC = exec cc
CFLAGS = -I$i
LDFLAGS = -i
LIBS = -lsys -lsysutil -ltimers

OBJ = floppy.o
LIBDRIVER = $d/libdriver/driver.o $d/libdriver/drvlib.o

# build local binary
all build: $(DRIVER)
$(DRIVER): $(OBJ) $(LIBDRIVER)
    $(CC) -o $@ $(LDFLAGS) $(OBJ) $(LIBDRIVER) $(LIBS)
    install -S 4096 $(DRIVER)

$(LIBDRIVER):
    cd $d/libdriver && $(MAKE)

# install with other drivers
install: /sbin/$(DRIVER)
/sbin/$(DRIVER): $(DRIVER)
    install -o root -cs $? $@

# clean up local files
clean:
    rm -f $(DRIVER) *.o *.bak

depend:
    /usr/bin/mkdep "$(CC) -E $(CPPFLAGS)" *.c ../libdriver/*.c > .depend

# Include generated dependencies.
include .depend
```

```

/* This file contains the device dependent part of the driver for the Floppy
 * Disk Controller (FDC) using the NEC PD765 chip.
 *
 * The file contains two entry points:
 *
 * floppy_task:    main entry when system is brought up
 *
 * Changes:
 *   Sep 11, 2005    code cleanup (Andy Tanenbaum)
 *   Dec 01, 2004    floppy driver moved to user-space (Jorrit N. Herder)
 *   Sep 15, 2004    sync alarms/ local timer management (Jorrit N. Herder)
 *   Aug 12, 2003    null seek no interrupt fix (Mike Haertel)
 *   May 14, 2000    d-d/i rewrite (Kees J. Bot)
 *   Apr 04, 1992    device dependent/independent split (Kees J. Bot)
 *   Mar 27, 1992    last details on density checking (Kees J. Bot)
 *   Feb 14, 1992    check drive density on opens only (Andy Tanenbaum)
 *   1991            len[] / motors / reset / step rate / ... (Bruce Evans)
 *   May 13, 1991    renovated the errors loop (Don Chapman)
 *   1989            I/O vector to keep up with 1-1 interleave (Bruce Evans)
 *   Jan 06, 1988    allow 1.44 MB diskettes (Al Crew)
 *   Nov 28, 1986    better resetting for 386 (Peter Kay)
 *   Oct 27, 1986    fdc_results fixed for 8 MHz (Jakob Schripsema)
 */

#include "floppy.h"
#include <timers.h>
#include <ibm/diskparm.h>
#include <minix/sysutil.h>
#include <minix/syslib.h>

/* I/O Ports used by floppy disk task. */
#define DOR                0x3F2    /* motor drive control bits */
#define FDC_STATUS         0x3F4    /* floppy disk controller status register */
#define FDC_DATA           0x3F5    /* floppy disk controller data register */
#define FDC_RATE           0x3F7    /* transfer rate register */
#define DMA_ADDR           0x004    /* port for low 16 bits of DMA address */
#define DMA_TOP            0x081    /* port for top 8 bits of 24-bit DMA addr */
#define DMA_COUNT          0x005    /* port for DMA count (count = bytes - 1) */
#define DMA_FLIPFLOP       0x00C    /* DMA byte pointer flip-flop */
#define DMA_MODE           0x00B    /* DMA mode port */
#define DMA_INIT           0x00A    /* DMA init port */
#define DMA_RESET_VAL      0x006

#define DMA_ADDR_MASK      0xFFFFF  /* mask to verify DMA address is 24-bit */

/* Status registers returned as result of operation. */
#define ST0                0x00    /* status register 0 */
#define ST1                0x01    /* status register 1 */
#define ST2                0x02    /* status register 2 */
#define ST3                0x00    /* status register 3 (return by DRIVE_SENSE) */
#define ST_CYL             0x03    /* slot where controller reports cylinder */
#define ST_HEAD            0x04    /* slot where controller reports head */
#define ST_SEC             0x05    /* slot where controller reports sector */
#define ST_PCN            0x01    /* slot where controller reports present cyl */

/* Fields within the I/O ports. */
/* Main status register. */
#define CTL_BUSY           0x10    /* bit is set when read or write in progress */
#define DIRECTION         0x40    /* bit is set when reading data reg is valid */
#define MASTER            0x80    /* bit is set when data reg can be accessed */

/* Digital output port (DOR). */
#define MOTOR_SHIFT        4      /* high 4 bits control the motors in DOR */
#define ENABLE_INT        0x0C    /* used for setting DOR port */

/* ST0. */
#define ST0_BITS_TRANS     0xD8    /* check 4 bits of status */
#define TRANS_ST0         0x00    /* 4 bits of ST0 for READ/WRITE */
#define ST0_BITS_SEEK     0xF8    /* check top 5 bits of seek status */
#define SEEK_ST0          0x20    /* top 5 bits of ST0 for SEEK */

/* ST1. */
#define BAD_SECTOR         0x05    /* if these bits are set in ST1, recalibrate */
#define WRITE_PROTECT     0x02    /* bit is set if diskette is write protected */

```

```

/* ST2. */
#define BAD_CYL          0x1F      /* if any of these bits are set, recalibrate */

/* ST3 (not used). */
#define ST3_FAULT        0x80      /* if this bit is set, drive is sick */
#define ST3_WR_PROTECT   0x40      /* set when diskette is write protected */
#define ST3_READY        0x20      /* set when drive is ready */

/* Floppy disk controller command bytes. */
#define FDC_SEEK          0x0F      /* command the drive to seek */
#define FDC_READ          0xE6      /* command the drive to read */
#define FDC_WRITE         0xC5      /* command the drive to write */
#define FDC_SENSE         0x08      /* command the controller to tell its status */
#define FDC_RECALIBRATE   0x07      /* command the drive to go to cyl 0 */
#define FDC_SPECIFY       0x03      /* command the drive to accept params */
#define FDC_READ_ID       0x4A      /* command the drive to read sector identity */
#define FDC_FORMAT        0x4D      /* command the drive to format a track */

/* DMA channel commands. */
#define DMA_READ          0x46      /* DMA read opcode */
#define DMA_WRITE         0x4A      /* DMA write opcode */

/* Parameters for the disk drive. */
#define HC_SIZE           2880      /* # sectors on largest legal disk (1.44MB) */
#define NR_HEADS          0x02      /* two heads (i.e., two tracks/cylinder) */
#define MAX_SECTORS       18        /* largest # sectors per track */
#define DTL               0xFF      /* determines data length (sector size) */
#define SPEC2             0x02      /* second parameter to SPECIFY */
#define MOTOR_OFF         (3*HZ)    /* how long to wait before stopping motor */
#define WAKEUP            (2*HZ)    /* timeout on I/O, FDC won't quit. */

/* Error codes */
#define ERR_SEEK          (-1)      /* bad seek */
#define ERR_TRANSFER      (-2)      /* bad transfer */
#define ERR_STATUS        (-3)      /* something wrong when getting status */
#define ERR_READ_ID       (-4)      /* bad read id */
#define ERR_RECALIBRATE   (-5)      /* recalibrate didn't work properly */
#define ERR_DRIVE         (-6)      /* something wrong with a drive */
#define ERR_WR_PROTECT    (-7)      /* diskette is write protected */
#define ERR_TIMEOUT       (-8)      /* interrupt timeout */

/* No retries on some errors. */
#define err_no_retry(err)  ((err) <= ERR_WR_PROTECT)

/* Encoding of drive type in minor device number. */
#define DEV_TYPE_BITS     0x7C      /* drive type + 1, if nonzero */
#define DEV_TYPE_SHIFT    2         /* right shift to normalize type bits */
#define FORMAT_DEV_BIT    0x80      /* bit in minor to turn write into format */

/* Miscellaneous. */
#define MAX_ERRORS        6         /* how often to try rd/wt before quitting */
#define MAX_RESULTS       7         /* max number of bytes controller returns */
#define NR_DRIVES         2         /* maximum number of drives */
#define DIVISOR           128       /* used for sector size encoding */
#define SECTOR_SIZE_CODE  2         /* code to say "512" to the controller */
#define TIMEOUT_MICROS    500000L   /* microseconds waiting for FDC */
#define TIMEOUT_TICKS     30        /* ticks waiting for FDC */
#define NT                7         /* number of diskette/drive combinations */
#define UNCALIBRATED      0         /* drive needs to be calibrated at next use */
#define CALIBRATED        1         /* no calibration needed */
#define BASE_SECTOR       1         /* sectors are numbered starting at 1 */
#define NO_SECTOR         (-1)      /* current sector unknown */
#define NO_CYL            (-1)      /* current cylinder unknown, must seek */
#define NO_DENS           100       /* current media unknown */
#define BSY_IDLE          0         /* busy doing nothing */
#define BSY_IO            1         /* busy doing I/O */
#define BSY_WAKEN        2         /* got a wakeup call */

/* Seven combinations of diskette/drive are supported.
 *
 * # Diskette Drive Sectors Tracks Rotation Data-rate Comment
 * 0 360K 360K 9 40 300 RPM 250 kbps Standard PC DSDD
 * 1 1.2M 1.2M 15 80 360 RPM 500 kbps AT disk in AT drive

```

```

* 2 360K 720K 9 40 300 RPM 250 kbps Quad density PC
* 3 720K 720K 9 80 300 RPM 250 kbps Toshiba, et al.
* 4 360K 1.2M 9 40 360 RPM 300 kbps PC disk in AT drive
* 5 720K 1.2M 9 80 360 RPM 300 kbps Toshiba in AT drive
* 6 1.44M 1.44M 18 80 300 RPM 500 kbps PS/2, et al.
*
* In addition, 720K diskettes can be read in 1.44MB drives, but that does
* not need a different set of parameters. This combination uses
*
* 3 720K 1.44M 9 80 300 RPM 250 kbps PS/2, et al.
*/
PRIVATE struct density {
    u8_t  sectp; /* sectors per track */
    u8_t  cyls; /* tracks per side */
    u8_t  steps; /* steps per cylinder (2 = double step) */
    u8_t  test; /* sector to try for density test */
    u8_t  rate; /* data rate (2=250, 1=300, 0=500 kbps) */
    u8_t  start; /* motor start (clock ticks) */
    u8_t  gap; /* gap size */
    u8_t  spec1; /* first specify byte (SRT/HUT) */
} fdensity[NT] = {
    { 9, 40, 1, 4*9, 2, 4*HZ/8, 0x2A, 0xDF }, /* 360K / 360K */
    { 15, 80, 1, 14, 0, 4*HZ/8, 0x1B, 0xDF }, /* 1.2M / 1.2M */
    { 9, 40, 2, 2*9, 2, 4*HZ/8, 0x2A, 0xDF }, /* 360K / 720K */
    { 9, 80, 1, 4*9, 2, 6*HZ/8, 0x2A, 0xDF }, /* 720K / 720K */
    { 9, 40, 2, 2*9, 1, 4*HZ/8, 0x23, 0xDF }, /* 360K / 1.2M */
    { 9, 80, 1, 4*9, 1, 4*HZ/8, 0x23, 0xDF }, /* 720K / 1.2M */
    { 18, 80, 1, 17, 0, 6*HZ/8, 0x1B, 0xCF }, /* 1.44M / 1.44M */
};

/* The following table is used with the test_sector array to recognize a
* drive/floppy combination. The sector to test has been determined by
* looking at the differences in gap size, sectors/track, and double stepping.
* This means that types 0 and 3 can't be told apart, only the motor start
* time differs. If a read test succeeds then the drive is limited to the
* set of densities it can support to avoid unnecessary tests in the future.
*/

#define b(d) (1 << (d)) /* bit for density d. */

PRIVATE struct test_order {
    u8_t  t_density; /* floppy/drive type */
    u8_t  t_class; /* limit drive to this class of densities */
} test_order[NT-1] = {
    { 6, b(3) | b(6) }, /* 1.44M {720K, 1.44M} */
    { 1, b(1) | b(4) }, /* 1.2M {1.2M, 360K, 720K} */
    { 3, b(2) | b(3) | b(6) }, /* 720K {360K, 720K, 1.44M} */
    { 4, b(1) | b(4) | b(5) }, /* 360K {1.2M, 360K, 720K} */
    { 5, b(1) | b(4) | b(5) }, /* 720K {1.2M, 360K, 720K} */
    { 2, b(2) | b(3) }, /* 360K {360K, 720K} */
};

/* Note that type 0 is missing, type 3 can read/write it too, which is
* why the type 3 parameters have been pessimized to be like type 0.
*/

/* Variables. */
PRIVATE struct floppy {
    unsigned fl_cyl; /* main drive struct, one entry per drive */
    unsigned fl_hardcyl; /* current cylinder */
    unsigned fl_cylinder; /* hardware cylinder, as opposed to: */
    unsigned fl_sector; /* cylinder number addressed */
    unsigned fl_head; /* sector addressed */
    unsigned fl_head; /* head number addressed */
    char fl_calibration; /* CALIBRATED or UNCALIBRATED */
    u8_t fl_density; /* NO_DENS = ?, 0 = 360K; 1 = 360K/1.2M; etc. */
    u8_t fl_class; /* bitmap for possible densities */
    timer_t fl_tmr_stop; /* timer to stop motor */
    struct device fl_geom; /* Geometry of the drive */
    struct device fl_part[NR_PARTITIONS]; /* partition's base & size */
} floppy[NR_DRIVES];

PRIVATE int irq_hook_id; /* id of irq hook at the kernel */
PRIVATE int motor_status; /* bitmap of current motor status */
PRIVATE int need_reset; /* set to 1 when controller must be reset */
PRIVATE unsigned f_drive; /* selected drive */

```

```

PRIVATE unsigned f_device;      /* selected minor device */
PRIVATE struct floppy *f_fp;    /* current drive */
PRIVATE struct density *f_dp;   /* current density parameters */
PRIVATE struct density *prev_dp; /* previous density parameters */
PRIVATE unsigned f_sectors;     /* equal to f_dp->secpt (needed a lot) */
PRIVATE u16_t f_busy;           /* BSY_IDLE, BSY_IO, BSY_WAKEN */
PRIVATE struct device *f_dv;    /* device's base and size */
PRIVATE struct disk_parameter_s fmt_param; /* parameters for format */
PRIVATE u8_t f_results[MAX_RESULTS]; /* the controller can give lots of output */

/* The floppy uses various timers. These are managed by the floppy driver
 * itself, because only a single synchronous alarm is available per process.
 * Besides the 'f_tmr_timeout' timer below, the floppy structure for each
 * floppy disk drive contains a 'fl_tmr_stop' timer.
 */
PRIVATE timer_t f_tmr_timeout; /* timer for various timeouts */
PRIVATE timer_t *f_timers;    /* queue of floppy timers */
PRIVATE clock_t f_next_timeout; /* the next timeout time */
FORWARD _PROTOTYPE( void f_expire_tmrs, (struct driver *dp, message *m_ptr) );
FORWARD _PROTOTYPE( void f_set_timer, (timer_t *tp, clock_t delta,
                                     tmr_func_t watchdog) );

FORWARD _PROTOTYPE( void stop_motor, (timer_t *tp) );
FORWARD _PROTOTYPE( void f_timeout, (timer_t *tp) );

FORWARD _PROTOTYPE( struct device *f_prepare, (int device) );
FORWARD _PROTOTYPE( char *f_name, (void) );
FORWARD _PROTOTYPE( void f_cleanup, (void) );
FORWARD _PROTOTYPE( int f_transfer, (int proc_nr, int opcode, off_t position,
                                     iovec_t *iovec, unsigned nr_req) );
FORWARD _PROTOTYPE( int dma_setup, (int opcode) );
FORWARD _PROTOTYPE( void start_motor, (void) );
FORWARD _PROTOTYPE( int seek, (void) );
FORWARD _PROTOTYPE( int fdc_transfer, (int opcode) );
FORWARD _PROTOTYPE( int fdc_results, (void) );
FORWARD _PROTOTYPE( int fdc_command, (u8_t *cmd, int len) );
FORWARD _PROTOTYPE( void fdc_out, (int val) );
FORWARD _PROTOTYPE( int recalibrate, (void) );
FORWARD _PROTOTYPE( void f_reset, (void) );
FORWARD _PROTOTYPE( int f_intr_wait, (void) );
FORWARD _PROTOTYPE( int read_id, (void) );
FORWARD _PROTOTYPE( int f_do_open, (struct driver *dp, message *m_ptr) );
FORWARD _PROTOTYPE( void floppy_stop, (struct driver *dp, message *m_ptr) );
FORWARD _PROTOTYPE( int test_read, (int density) );
FORWARD _PROTOTYPE( void f_geometry, (struct partition *entry) );

/* Entry points to this driver. */
PRIVATE struct driver f_dtab = {
    f_name,      /* current device's name */
    f_do_open,   /* open or mount request, sense type of diskette */
    do_nop,      /* nothing on a close */
    do_diocntl,  /* get or set a partitions geometry */
    f_prepare,   /* prepare for I/O on a given minor device */
    f_transfer,  /* do the I/O */
    f_cleanup,   /* cleanup before sending reply to user process */
    f_geometry,  /* tell the geometry of the diskette */
    floppy_stop, /* floppy cleanup on shutdown */
    f_expire_tmrs, /* expire all alarm timers */
    nop_cancel,
    nop_select,
    NULL,
    NULL
};

/*=====
 *                      floppy_task
 *=====*/
PUBLIC void main()
{
    /* Initialize the floppy structure and the timers. */

    struct floppy *fp;
    int s;

    f_next_timeout = TMR_NEVER;

```

```

tmr_inittimer(&f_tmr_timeout);

for (fp = &floppy[0]; fp < &floppy[NR_DRIVES]; fp++) {
    fp->fl_curcyl = NO_CYL;
    fp->fl_density = NO_DENS;
    fp->fl_class = ~0;
    tmr_inittimer(&fp->fl_tmr_stop);
}

/* Set IRQ policy, only request notifications, do not automatically
 * reenale interrupts. ID return on interrupt is the IRQ line number.
 */
irq_hook_id = FLOPPY_IRQ;
if ((s=sys_irqsetpolicy(FLOPPY_IRQ, 0, &irq_hook_id)) != OK)
    panic("FLOPPY", "Couldn't set IRQ policy", s);
if ((s=sys_irgenable(&irq_hook_id)) != OK)
    panic("FLOPPY", "Couldn't enable IRQs", s);

/* Ignore signals */
signal(SIGHUP, SIG_IGN);

driver_task(&f_dtab);
}

/*=====
 *                               f_expire_tmrs                               *
 *=====*/
PRIVATE void f_expire_tmrs(struct driver *dp, message *m_ptr)
{
    /* A synchronous alarm message was received. Check if there are any expired
     * timers. Possibly reschedule the next alarm.
     */
    clock_t now;                               /* current time */
    timer_t *tp;
    int s;

    /* Get the current time to compare the timers against. */
    if ((s=getuptime(&now)) != OK)
        panic("FLOPPY", "Couldn't get uptime from clock.", s);

    /* Scan the timers queue for expired timers. Dispatch the watchdog function
     * for each expired timers. FLOPPY watchdog functions are f_tmr_timeout()
     * and stop_motor(). Possibly a new alarm call must be scheduled.
     */
    tmrs_exptimers(&f_timers, now, NULL);
    if (f_timers == NULL) {
        f_next_timeout = TMR_NEVER;
    } else {
        f_next_timeout = f_timers->tmr_exp_time;
        if ((s=sys_setalarm(f_next_timeout, 1)) != OK)
            panic("FLOPPY", "Couldn't set synchronous alarm.", s);
    }
}

/*=====
 *                               f_set_timer                               *
 *=====*/
PRIVATE void f_set_timer(tp, delta, watchdog)
timer_t *tp;                               /* timer to be set */
clock_t delta;                             /* in how many ticks */
tmr_func_t watchdog;                       /* watchdog function to be called */
{
    clock_t now;                               /* current time */
    int s;

    /* Get the current time. */
    if ((s=getuptime(&now)) != OK)
        panic("FLOPPY", "Couldn't get uptime from clock.", s);

    /* Add the timer to the local timer queue. */
    tmrs_settimer(&f_timers, tp, now + delta, watchdog, NULL);

    /* Possibly reschedule an alarm call. This happens when the front of the
     * timers queue was reinserted at another position, i.e., when a timer was

```

```

    * reset, or when a new timer was added in front.
    */
    if (f_timers->tmr_exp_time != f_next_timeout) {
        f_next_timeout = f_timers->tmr_exp_time;
        if ((s=sys_setalarm(f_next_timeout, 1)) != OK)
            panic("FLOPPY", "Couldn't set synchronous alarm.", s);
    }
}

/*=====
*                                     f_prepare                                     *
*=====*/
PRIVATE struct device *f_prepare(device)
int device;
{
    /* Prepare for I/O on a device. */

    f_device = device;
    f_drive = device & ~(DEV_TYPE_BITS | FORMAT_DEV_BIT);
    if (f_drive < 0 || f_drive >= NR_DRIVES) return(NIL_DEV);

    f_fp = &floppy[f_drive];
    f_dv = &f_fp->fl_geom;
    if (f_fp->fl_density < NT) {
        f_dp = &fdensity[f_fp->fl_density];
        f_sectors = f_dp->sectp;
        f_fp->fl_geom.dv_size = mul64u((long) (NR_HEADS * f_sectors
            * f_dp->cyls), SECTOR_SIZE);
    }

    /* A partition? */
    if ((device &= DEV_TYPE_BITS) >= MINOR_fd0p0)
        f_dv = &f_fp->fl_part[(device - MINOR_fd0p0) >> DEV_TYPE_SHIFT];

    return f_dv;
}

/*=====
*                                     f_name                                     *
*=====*/
PRIVATE char *f_name()
{
    /* Return a name for the current device. */
    static char name[] = "fd0";

    name[2] = '0' + f_drive;
    return name;
}

/*=====
*                                     f_cleanup                                    *
*=====*/
PRIVATE void f_cleanup()
{
    /* Start a timer to turn the motor off in a few seconds. */
    tmr_arg(&f_fp->fl_tmr_stop)->ta_int = f_drive;
    f_set_timer(&f_fp->fl_tmr_stop, MOTOR_OFF, stop_motor);

    /* Exiting the floppy driver, so forget where we are. */
    f_fp->fl_sector = NO_SECTOR;
}

/*=====
*                                     f_transfer                                    *
*=====*/
PRIVATE int f_transfer(proc_nr, opcode, position, iov, nr_req)
int proc_nr;          /* process doing the request */
int opcode;           /* DEV_GATHER or DEV_SCATTER */
off_t position;       /* offset on device to read or write */
iovec_t *iov;         /* pointer to read or write request vector */
unsigned nr_req;      /* length of request vector */
{
    struct floppy *fp = f_fp;
    iovec_t *iop, *iov_end = iov + nr_req;

```



```

int s, r, errors;
unsigned block;          /* Seen any 32M floppies lately? */
unsigned nbytes, count, chunk, sector;
unsigned long dv_size = cv64ul(f_dv->dv_size);
vir_bytes user_addr;
vir_bytes uaddrs[MAX_SECTORS], *up;
u8_t cmd[3];

/* Check disk address. */
if ((position & SECTOR_MASK) != 0) return(EINVAL);

#if 0 /* XXX hack to create a disk driver that crashes */
{ static int count= 0; if (++count > 10) {
    printf("floppy: time to die\n"); *(int *)-1= 42;
  }}
#endif

errors = 0;
while (nr_req > 0) {
    /* How many bytes to transfer? */
    nbytes = 0;
    for (iop = iov; iop < iov_end; iop++) nbytes += iop->iov_size;

    /* Which block on disk and how close to EOF? */
    if (position >= dv_size) return(OK);          /* At EOF */
    if (position + nbytes > dv_size) nbytes = dv_size - position;
    block = div64u(add64ul(f_dv->dv_base, position), SECTOR_SIZE);

    if ((nbytes & SECTOR_MASK) != 0) return(EINVAL);

    /* Using a formatting device? */
    if (f_device & FORMAT_DEV_BIT) {
        if (opcode != DEV_SCATTER) return(EIO);
        if (iop->iov_size < SECTOR_SIZE + sizeof(fmt_param))
            return(EINVAL);

        if ((s=sys_datacopy(proc_nr, iop->iov_addr + SECTOR_SIZE,
            SELF, (vir_bytes) &fmt_param,
            (phys_bytes) sizeof(fmt_param))) != OK)
            panic("FLOPPY", "Sys_vircopy failed", s);

        /* Check that the number of sectors in the data is reasonable,
         * to avoid division by 0. Leave checking of other data to
         * the FDC.
         */
        if (fmt_param.sectors_per_cylinder == 0) return(EIO);

        /* Only the first sector of the parameters now needed. */
        iop->iov_size = nbytes = SECTOR_SIZE;
    }

    /* Only try one sector if there were errors. */
    if (errors > 0) nbytes = SECTOR_SIZE;

    /* Compute cylinder and head of the track to access. */
    fp->fl_cylinder = block / (NR_HEADS * f_sectors);
    fp->fl_hardcyl = fp->fl_cylinder * f_dp->steps;
    fp->fl_head = (block % (NR_HEADS * f_sectors)) / f_sectors;

    /* For each sector on this track compute the user address it is to
     * go or to come from.
     */
    for (up = uaddrs; up < uaddrs + MAX_SECTORS; up++) *up = 0;
    count = 0;
    iop = iov;
    sector = block % f_sectors;
    for (;;) {
        user_addr = iop->iov_addr;
        chunk = iop->iov_size;
        if ((chunk & SECTOR_MASK) != 0) return(EINVAL);

        while (chunk > 0) {
            uaddrs[sector++] = user_addr;
            chunk -= SECTOR_SIZE;
        }
    }
}

```

```

        user_addr += SECTOR_SIZE;
        count += SECTOR_SIZE;
        if (sector == f_sectors || count == nbytes)
            goto track_set_up;
    }
    iop++;
}
track_set_up:

/* First check to see if a reset is needed. */
if (need_reset) f_reset();

/* See if motor is running; if not, turn it on and wait. */
start_motor();

/* Set the stepping rate and data rate */
if (f_dp != prev_dp) {
    cmd[0] = FDC_SPECIFY;
    cmd[1] = f_dp->spec1;
    cmd[2] = SPEC2;
    (void) fdc_command(cmd, 3);
    if ((s=sys_outb(FDC_RATE, f_dp->rate)) != OK)
        panic("FLOPPY", "Sys_outb failed", s);
    prev_dp = f_dp;
}

/* If we are going to a new cylinder, perform a seek. */
r = seek();

/* Avoid read_id() if we don't plan to read much. */
if (fp->fl_sector == NO_SECTOR && count < (6 * SECTOR_SIZE))
    fp->fl_sector = 0;

for (nbytes = 0; nbytes < count; nbytes += SECTOR_SIZE) {
    if (fp->fl_sector == NO_SECTOR) {
        /* Find out what the current sector is. This often
         * fails right after a seek, so try it twice.
         */
        if (r == OK && read_id() != OK) r = read_id();
    }

    /* Look for the next job in uaddrs[] */
    if (r == OK) {
        for (;;) {
            if (fp->fl_sector >= f_sectors)
                fp->fl_sector = 0;

            up = &uaddrs[fp->fl_sector];
            if (*up != 0) break;
            fp->fl_sector++;
        }
    }

    if (r == OK && opcode == DEV_SCATTER) {
        /* Copy the user bytes to the DMA buffer. */
        if ((s=sys_datacopy(proc_nr, *up, SELF,
            (vir_bytes) tmp_buf,
            (phys_bytes) SECTOR_SIZE)) != OK)
            panic("FLOPPY", "Sys_vircopy failed", s);
    }

    /* Set up the DMA chip and perform the transfer. */
    if (r == OK) {
        if (dma_setup(opcode) != OK) {
            /* This can only fail for addresses above 16MB
             * that cannot be handled by the controller,
             * because it uses 24-bit addressing.
             */
            return(EIO);
        }
        r = fdc_transfer(opcode);
    }

    if (r == OK && opcode == DEV_GATHER) {

```

```

        /* Copy the DMA buffer to user space. */
        if ((s=sys_datacopy(SELF, (vir_bytes) tmp_buf,
                             proc_nr, *up,
                             (phys_bytes) SECTOR_SIZE)) != OK)
            panic("FLOPPY", "Sys_vircopy failed", s);
    }

    if (r != OK) {
        /* Don't retry if write protected or too many errors. */
        if (err_no_retry(r) || ++errors == MAX_ERRORS) {
            return(EIO);
        }

        /* Recalibrate if halfway. */
        if (errors == MAX_ERRORS / 2)
            fp->fl_calibration = UNCALIBRATED;

        nbytes = 0;
        break;          /* retry */
    }
}

/* Book the bytes successfully transferred. */
position += nbytes;
for (;;) {
    if (nbytes < iov->iov_size) {
        /* Not done with this one yet. */
        iov->iov_addr += nbytes;
        iov->iov_size -= nbytes;
        break;
    }
    nbytes -= iov->iov_size;
    iov->iov_addr += iov->iov_size;
    iov->iov_size = 0;
    if (nbytes == 0) {
        /* The rest is optional, so we return to give FS a
         * chance to think it over.
         */
        return(OK);
    }
    iov++;
    nr_req--;
}
}
return(OK);
}

/*=====
 *                               dma_setup                               *
 *=====*/
PRIVATE int dma_setup(opcode)
int opcode;          /* DEV_GATHER or DEV_SCATTER */
{
    /* The IBM PC can perform DMA operations by using the DMA chip. To use it,
     * the DMA (Direct Memory Access) chip is loaded with the 20-bit memory address
     * to be read from or written to, the byte count minus 1, and a read or write
     * opcode. This routine sets up the DMA chip. Note that the chip is not
     * capable of doing a DMA across a 64K boundary (e.g., you can't read a
     * 512-byte block starting at physical address 65520).
     *
     * Warning! Also note that it's not possible to do DMA above 16 MB because
     * the ISA bus uses 24-bit addresses. Addresses above 16 MB therefore will
     * be interpreted modulo 16 MB, dangerously overwriting arbitrary memory.
     * A check here denies the I/O if the address is out of range.
     */
    pvb_pair_t byte_out[9];
    int s;

    /* First check the DMA memory address not to exceed maximum. */
    if (tmp_phys != (tmp_phys & DMA_ADDR_MASK)) {
        report("FLOPPY", "DMA denied because address out of range", NO_NUM);
        return(EIO);
    }
}

```

```

/* Set up the DMA registers. (The comment on the reset is a bit strong,
 * it probably only resets the floppy channel.)
 */
pv_set(byte_out[0], DMA_INIT, DMA_RESET_VAL); /* reset the dma controller */
pv_set(byte_out[1], DMA_FLIPFLOP, 0);          /* write anything to reset it */
pv_set(byte_out[2], DMA_MODE, opcode == DEV_SCATTER ? DMA_WRITE : DMA_READ);
pv_set(byte_out[3], DMA_ADDR, (unsigned) tmp_phys >> 0);
pv_set(byte_out[4], DMA_ADDR, (unsigned) tmp_phys >> 8);
pv_set(byte_out[5], DMA_TOP, (unsigned) (tmp_phys >> 16));
pv_set(byte_out[6], DMA_COUNT, (((SECTOR_SIZE - 1) >> 0) & 0xff));
pv_set(byte_out[7], DMA_COUNT, (SECTOR_SIZE - 1) >> 8);
pv_set(byte_out[8], DMA_INIT, 2);              /* some sort of enable */

if ((s=sys_vouth(byte_out, 9)) != OK)
    panic("FLOPPY", "Sys_vouth in dma_setup() failed", s);
return(OK);
}

/*=====
 *
 * start_motor
 *=====*/
PRIVATE void start_motor()
{
/* Control of the floppy disk motors is a big pain. If a motor is off, you
 * have to turn it on first, which takes 1/2 second. You can't leave it on
 * all the time, since that would wear out the diskette. However, if you turn
 * the motor off after each operation, the system performance will be awful.
 * The compromise used here is to leave it on for a few seconds after each
 * operation. If a new operation is started in that interval, it need not be
 * turned on again. If no new operation is started, a timer goes off and the
 * motor is turned off. I/O port DOR has bits to control each of 4 drives.
 */

int s, motor_bit, running;
message mess;

motor_bit = 1 << f_drive;          /* bit mask for this drive */
running = motor_status & motor_bit; /* nonzero if this motor is running */
motor_status |= motor_bit;          /* want this drive running too */

if ((s=sys_outb(DOR,
    (motor_status << MOTOR_SHIFT) | ENABLE_INT | f_drive)) != OK)
    panic("FLOPPY", "Sys_outb in start_motor() failed", s);

/* If the motor was already running, we don't have to wait for it. */
if (running) return;               /* motor was already running */

/* Set an alarm timer to force a timeout if the hardware does not interrupt
 * in time. Expect HARD_INT message, but check for SYN_ALARM timeout.
 */
f_set_timer(&f_tmr_timeout, f_dp->start, f_timeout);
f_busy = BSY_IO;
do {
    receive(ANY, &mess);
    if (mess.m_type == SYN_ALARM) {
        f_expire_tmrs(NULL, NULL);
    } else if (mess.m_type == DEV_PING) {
        notify(mess.m_source);
    } else {
        f_busy = BSY_IDLE;
    }
} while (f_busy == BSY_IO);
f_fp->fl_sector = NO_SECTOR;
}

/*=====
 *
 * stop_motor
 *=====*/
PRIVATE void stop_motor(tp)
timer_t *tp;
{
/* This routine is called from an alarm timer after several seconds have
 * elapsed with no floppy disk activity. It turns the drive motor off.
 */
}

```

```

int s;
motor_status &= ~(1 << tmr_arg(tp)->ta_int);
if ((s=sys_outb(DOR, (motor_status << MOTOR_SHIFT) | ENABLE_INT)) != OK)
    panic("FLOPPY", "Sys_outb in stop_motor() failed", s);
}

/*=====
*                               floppy_stop                               *
*=====*/
PRIVATE void floppy_stop(struct driver *dp, message *m_ptr)
{
    /* Stop all activity and cleanly exit with the system. */
    int s;
    sigset_t sigset = m_ptr->NOTIFY_ARG;
    if (sigismember(&sigset, SIGTERM) || sigismember(&sigset, SIGKSTOP)) {
        if ((s=sys_outb(DOR, ENABLE_INT)) != OK)
            panic("FLOPPY", "Sys_outb in floppy_stop() failed", s);
        exit(0);
    }
}

/*=====
*                               seek                               *
*=====*/
PRIVATE int seek()
{
    /* Issue a SEEK command on the indicated drive unless the arm is already
    * positioned on the correct cylinder.
    */

    struct floppy *fp = f_fp;
    int r;
    message mess;
    u8_t cmd[3];

    /* Are we already on the correct cylinder? */
    if (fp->fl_calibration == UNCALIBRATED)
        if (recalibrate() != OK) return(ERR_SEEK);
    if (fp->fl_curcyl == fp->fl_hardcyl) return(OK);

    /* No. Wrong cylinder. Issue a SEEK and wait for interrupt. */
    cmd[0] = FDC_SEEK;
    cmd[1] = (fp->fl_head << 2) | f_drive;
    cmd[2] = fp->fl_hardcyl;
    if (fdc_command(cmd, 3) != OK) return(ERR_SEEK);
    if (f_intr_wait() != OK) return(ERR_TIMEOUT);

    /* Interrupt has been received. Check drive status. */
    fdc_out(FDC_SENSE); /* probe FDC to make it return status */
    r = fdc_results(); /* get controller status bytes */
    if (r != OK || (f_results[ST0] & ST0_BITS_SEEK) != SEEK_ST0
        || f_results[ST1] != fp->fl_hardcyl) {
        /* seek failed, may need a recalibrate */
        return(ERR_SEEK);
    }

    /* Give head time to settle on a format, no retrying here! */
    if (f_device & FORMAT_DEV_BIT) {
        /* Set a synchronous alarm to force a timeout if the hardware does
        * not interrupt. Expect HARD_INT, but check for SYN_ALARM timeout.
        */
        f_set_timer(&f_tmr_timeout, HZ/30, f_timeout);
        f_busy = BSY_IO;
        do {
            receive(ANY, &mess);
            if (mess.m_type == SYN_ALARM) {
                f_expire_tmrs(NULL, NULL);
            } else if (mess.m_type == DEV_PING) {
                notify(mess.m_source);
            } else {
                f_busy = BSY_IDLE;
            }
        } while (f_busy == BSY_IO);
    }
    fp->fl_curcyl = fp->fl_hardcyl;
}

```

```

    fp->fl_sector = NO_SECTOR;
    return(OK);
}

/*=====
 *                               fd_c_transfer                               *
 *=====*/
PRIVATE int fd_c_transfer(opcode)
int opcode;                /* DEV_GATHER or DEV_SCATTER */
{
    /* The drive is now on the proper cylinder.  Read, write or format 1 block. */

    struct floppy *fp = f_fp;
    int r, s;
    u8_t cmd[9];

    /* Never attempt a transfer if the drive is uncalibrated or motor is off. */
    if (fp->fl_calibration == UNCALIBRATED) return(ERR_TRANSFER);
    if ((motor_status & (1 << f_drive)) == 0) return(ERR_TRANSFER);

    /* The command is issued by outputting several bytes to the controller chip.
    */
    if (f_device & FORMAT_DEV_BIT) {
        cmd[0] = FDC_FORMAT;
        cmd[1] = (fp->fl_head << 2) | f_drive;
        cmd[2] = fmt_param.sector_size_code;
        cmd[3] = fmt_param.sectors_per_cylinder;
        cmd[4] = fmt_param.gap_length_for_format;
        cmd[5] = fmt_param.fill_byte_for_format;
        if (fd_c_command(cmd, 6) != OK) return(ERR_TRANSFER);
    } else {
        cmd[0] = opcode == DEV_SCATTER ? FDC_WRITE : FDC_READ;
        cmd[1] = (fp->fl_head << 2) | f_drive;
        cmd[2] = fp->fl_cylinder;
        cmd[3] = fp->fl_head;
        cmd[4] = BASE_SECTOR + fp->fl_sector;
        cmd[5] = SECTOR_SIZE_CODE;
        cmd[6] = f_sectors;
        cmd[7] = f_dp->gap;        /* sector gap */
        cmd[8] = DTL;            /* data length */
        if (fd_c_command(cmd, 9) != OK) return(ERR_TRANSFER);
    }

    /* Block, waiting for disk interrupt. */
    if (f_intr_wait() != OK) {
        printf("%s: disk interrupt timed out.\n", f_name());
        return(ERR_TIMEOUT);
    }

    /* Get controller status and check for errors. */
    r = fd_c_results();
    if (r != OK) return(r);

    if (f_results[ST1] & WRITE_PROTECT) {
        printf("%s: diskette is write protected.\n", f_name());
        return(ERR_WR_PROTECT);
    }

    if ((f_results[ST0] & ST0_BITS_TRANS) != TRANS_ST0) return(ERR_TRANSFER);
    if (f_results[ST1] | f_results[ST2]) return(ERR_TRANSFER);

    if (f_device & FORMAT_DEV_BIT) return(OK);

    /* Compare actual numbers of sectors transferred with expected number. */
    s = (f_results[ST_CYL] - fp->fl_cylinder) * NR_HEADS * f_sectors;
    s += (f_results[ST_HEAD] - fp->fl_head) * f_sectors;
    s += (f_results[ST_SEC] - BASE_SECTOR - fp->fl_sector);
    if (s != 1) return(ERR_TRANSFER);

    /* This sector is next for I/O: */
    fp->fl_sector = f_results[ST_SEC] - BASE_SECTOR;
#ifdef 0
    if (processor < 386) fp->fl_sector++;        /* Old CPU can't keep up. */
#endif
}

```

```

    return(OK);
}

/*=====
 *                               fd_c_results                               *
 *=====*/
PRIVATE int fd_c_results()
{
    /* Extract results from the controller after an operation, then allow floppy
     * interrupts again.
     */

    int s, result_nr;
    unsigned long status;
    clock_t t0,t1;

    /* Extract bytes from FDC until it says it has no more. The loop is
     * really an outer loop on result_nr and an inner loop on status.
     * A timeout flag alarm is set.
     */
    result_nr = 0;
    getuptime(&t0);
    do {
        /* Reading one byte is almost a mirror of fd_c_out() - the DIRECTION
         * bit must be set instead of clear, but the CTL_BUSY bit destroys
         * the perfection of the mirror.
         */
        if ((s=sys_inb(FDC_STATUS, &status)) != OK)
            panic("FLOPPY", "Sys_inb in fd_c_results() failed", s);
        status &= (MASTER | DIRECTION | CTL_BUSY);
        if (status == (MASTER | DIRECTION | CTL_BUSY)) {
            unsigned long tmp_r;
            if (result_nr >= MAX_RESULTS) break; /* too many results */
            if ((s=sys_inb(FDC_DATA, &tmp_r)) != OK)
                panic("FLOPPY", "Sys_inb in fd_c_results() failed", s);
            f_results[result_nr] = tmp_r;
            result_nr ++;
            continue;
        }
        if (status == MASTER) {
            /* all read */
            if ((s=sys_irqenable(&irq_hook_id)) != OK)
                panic("FLOPPY", "Couldn't enable IRQs", s);

            return(OK); /* only good exit */
        }
    } while ( (s=getuptime(&t1))!=OK && (t1-t0) < TIMEOUT_TICKS );
    if (OK!=s) printf("FLOPPY: warning, getuptime failed: %d\n", s);
    need_reset = TRUE; /* controller chip must be reset */

    if ((s=sys_irqenable(&irq_hook_id)) != OK)
        panic("FLOPPY", "Couldn't enable IRQs", s);
    return(ERR_STATUS);
}

/*=====
 *                               fd_c_command                               *
 *=====*/
PRIVATE int fd_c_command(cmd, len)
u8_t *cmd; /* command bytes */
int len; /* command length */
{
    /* Output a command to the controller. */

    /* Set a synchronous alarm to force a timeout if the hardware does
     * not interrupt. Expect HARD_INT, but check for SYN_ALARM timeout.
     * Note that the actual check is done by the code that issued the
     * fd_c_command() call.
     */
    f_set_timer(&f_tmr_timeout, WAKEUP, f_timeout);

    f_busy = BSY_IO;
    while (len > 0) {
        fd_c_out(*cmd++);
        len--;
    }
}

```

```

}
return(need_reset ? ERR_DRIVE : OK);
}

/*=====
 *                               fd_c_out                               *
 *=====*/
PRIVATE void fd_c_out(val)
int val;          /* write this byte to floppy disk controller */
{
/* Output a byte to the controller. This is not entirely trivial, since you
 * can only write to it when it is listening, and it decides when to listen.
 * If the controller refuses to listen, the FDC chip is given a hard reset.
 */
clock_t t0, t1;
int s;
unsigned long status;

if (need_reset) return;      /* if controller is not listening, return */

/* It may take several tries to get the FDC to accept a command. */
getuptime(&t0);
do {
    if ( (s=getuptime(&t1))==OK && (t1-t0) > TIMEOUT_TICKS ) {
        if (OK!=s) printf("FLOPPY: warning, getuptime failed: %d\n", s);
        need_reset = TRUE;      /* hit it over the head */
        return;
    }
    if ((s=sys_inb(FDC_STATUS, &status)) != OK)
        panic("FLOPPY", "Sys_inb in fd_c_out() failed", s);
}
while ((status & (MASTER | DIRECTION)) != (MASTER | 0));

if ((s=sys_outb(FDC_DATA, val)) != OK)
    panic("FLOPPY", "Sys_outb in fd_c_out() failed", s);
}

/*=====
 *                               recalibrate                               *
 *=====*/
PRIVATE int recalibrate()
{
/* The floppy disk controller has no way of determining its absolute arm
 * position (cylinder). Instead, it steps the arm a cylinder at a time and
 * keeps track of where it thinks it is (in software). However, after a
 * SEEK, the hardware reads information from the diskette telling where the
 * arm actually is. If the arm is in the wrong place, a recalibration is done,
 * which forces the arm to cylinder 0. This way the controller can get back
 * into sync with reality.
 */

struct floppy *fp = f_fp;
int r;
u8_t cmd[2];

/* Issue the RECALIBRATE command and wait for the interrupt. */
cmd[0] = FDC_RECALIBRATE;      /* tell drive to recalibrate itself */
cmd[1] = f_drive;              /* specify drive */
if (fd_c_command(cmd, 2) != OK) return(ERR_SEEK);
if (f_intr_wait() != OK) return(ERR_TIMEOUT);

/* Determine if the recalibration succeeded. */
fd_c_out(FDC_SENSE);           /* issue SENSE command to request results */
r = fd_c_results();            /* get results of the FDC_RECALIBRATE command */
fp->fl_curcyl = NO_CYL;        /* force a SEEK next time */
fp->fl_sector = NO_SECTOR;
if (r != OK ||                /* controller would not respond */
    (f_results[ST0] & ST0_BITS_SEEK) != SEEK_ST0 || f_results[ST_PCN] != 0) {
    /* Recalibration failed. FDC must be reset. */
    need_reset = TRUE;
    return(ERR_RECALIBRATE);
} else {
    /* Recalibration succeeded. */
    fp->fl_calibration = CALIBRATED;
}
}

```



```

        fp->fl_curcyl = f_results[ST_PCN];
        return(OK);
    }
}

/*=====
 *                               f_reset                               *
 *=====*/
PRIVATE void f_reset()
{
    /* Issue a reset to the controller. This is done after any catastrophe,
     * like the controller refusing to respond.
     */
    pvb_pair_t byte_out[2];
    int s,i;
    message mess;

    /* Disable interrupts and strobe reset bit low. */
    need_reset = FALSE;

    /* It is not clear why the next lock is needed. Writing 0 to DOR causes
     * interrupt, while the PC documentation says turning bit 8 off disables
     * interrupts. Without the lock:
     * 1) the interrupt handler sets the floppy mask bit in the 8259.
     * 2) writing ENABLE_INT to DOR causes the FDC to assert the interrupt
     *    line again, but the mask stops the cpu being interrupted.
     * 3) the sense interrupt clears the interrupt (not clear which one).
     * and for some reason the reset does not work.
     */
    (void) fdc_command((u8_t *) 0, 0); /* need only the timer */
    motor_status = 0;
    pv_set(byte_out[0], DOR, 0);          /* strobe reset bit low */
    pv_set(byte_out[1], DOR, ENABLE_INT); /* strobe it high again */
    if ((s=sys_voutb(byte_out, 2)) != OK)
        panic("FLOPPY", "Sys_voutb in f_reset() failed", s);

    /* A synchronous alarm timer was set in fdc_command. Expect a HARD_INT
     * message to collect the reset interrupt, but be prepared to handle the
     * SYN_ALARM message on a timeout.
     */
    do {
        receive(ANY, &mess);
        if (mess.m_type == SYN_ALARM) {
            f_expire_tmrs(NULL, NULL);
        } else if (mess.m_type == DEV_PING) {
            notify(mess.m_source);
        } else {
            /* expect HARD_INT */
            f_busy = BSY_IDLE;
        }
    } while (f_busy == BSY_IO);

    /* The controller supports 4 drives and returns a result for each of them.
     * Collect all the results now. The old version only collected the first
     * result. This happens to work for 2 drives, but it doesn't work for 3
     * or more drives, at least with only drives 0 and 2 actually connected
     * (the controller generates an extra interrupt for the middle drive when
     * drive 2 is accessed and the driver panics).
     *
     * It would be better to keep collecting results until there are no more.
     * For this, fdc_results needs to return the number of results (instead
     * of OK) when it succeeds.
     */
    for (i = 0; i < 4; i++) {
        fdc_out(FDC_SENSE); /* probe FDC to make it return status */
        (void) fdc_results(); /* flush controller */
    }
    for (i = 0; i < NR_DRIVES; i++) /* clear each drive */
        floppy[i].fl_calibration = UNCALIBRATED;

    /* The current timing parameters must be specified again. */
    prev_dp = NULL;
}

/*=====

```

```

*                                     f_intr_wait                                     *
*=====*/
PRIVATE int f_intr_wait()
{
/* Wait for an interrupt, but not forever. The FDC may have all the time of
* the world, but we humans do not.
*/
    message mess;

/* We expect a HARD_INT message from the interrupt handler, but if there is
* a timeout, a SYN_ALARM notification is received instead. If a timeout
* occurs, report an error.
*/
    do {
        receive(ANY, &mess);
        if (mess.m_type == SYN_ALARM) {
            f_expire_tmrs(NULL, NULL);
        } else if (mess.m_type == DEV_PING) {
            notify(mess.m_source);
        } else {
            f_busy = BSY_IDLE;
        }
    } while (f_busy == BSY_IO);

    if (f_busy == BSY_WAKEN) {

        /* No interrupt from the FDC, this means that there is probably no
        * floppy in the drive. Get the FDC down to earth and return error.
        */
        need_reset = TRUE;
        return(ERR_TIMEOUT);
    }
    return(OK);
}

/*=====*/
*                                     f_timeout                                     *
*=====*/
PRIVATE void f_timeout(tp)
timer_t *tp;
{
/* This routine is called when a timer expires. Usually to tell that a
* motor has spun up, but also to forge an interrupt when it takes too long
* for the FDC to interrupt (no floppy in the drive). It sets a flag to tell
* what has happened.
*/
    if (f_busy == BSY_IO) {
        f_busy = BSY_WAKEN;
    }
}

/*=====*/
*                                     read_id                                     *
*=====*/
PRIVATE int read_id()
{
/* Determine current cylinder and sector. */

    struct floppy *fp = f_fp;
    int result;
    u8_t cmd[2];

/* Never attempt a read id if the drive is uncalibrated or motor is off. */
    if (fp->fl_calibration == UNCALIBRATED) return(ERR_READ_ID);
    if ((motor_status & (1 << f_drive)) == 0) return(ERR_READ_ID);

/* The command is issued by outputting 2 bytes to the controller chip. */
    cmd[0] = FDC_READ_ID; /* issue the read id command */
    cmd[1] = (fp->fl_head << 2) | f_drive;
    if (fdc_command(cmd, 2) != OK) return(ERR_READ_ID);
    if (f_intr_wait() != OK) return(ERR_TIMEOUT);

/* Get controller status and check for errors. */
    result = fdc_results();

```

```

if (result != OK) return(result);

if ((f_results[ST0] & ST0_BITS_TRANS) != TRANS_ST0) return(ERR_READ_ID);
if (f_results[ST1] | f_results[ST2]) return(ERR_READ_ID);

/* The next sector is next for I/O: */
fp->fl_sector = f_results[ST_SEC] - BASE_SECTOR + 1;
return(OK);
}

/*=====
*                                     f_do_open                                     *
*=====*/
PRIVATE int f_do_open(dp, m_ptr)
struct driver *dp;
message *m_ptr;                /* pointer to open message */
{
/* Handle an open on a floppy. Determine diskette type if need be. */

    int dtype;
    struct test_order *top;

    /* Decode the message parameters. */
    if (f_prepare(m_ptr->DEVICE) == NIL_DEV) return(ENXIO);

    dtype = f_device & DEV_TYPE_BITS;    /* get density from minor dev */
    if (dtype >= MINOR_fd0p0) dtype = 0;

    if (dtype != 0) {
        /* All types except 0 indicate a specific drive/medium combination.*/
        dtype = (dtype >> DEV_TYPE_SHIFT) - 1;
        if (dtype >= NT) return(ENXIO);
        f_fp->fl_density = dtype;
        (void) f_prepare(f_device);    /* Recompute parameters. */
        return(OK);
    }
    if (f_device & FORMAT_DEV_BIT) return(EIO);    /* Can't format /dev/fdN */

    /* The device opened is /dev/fdN. Experimentally determine drive/medium.
     * First check fl_density. If it is not NO_DENS, the drive has been used
     * before and the value of fl_density tells what was found last time. Try
     * that first. If the motor is still running then assume nothing changed.
     */
    if (f_fp->fl_density != NO_DENS) {
        if (motor_status & (1 << f_drive)) return(OK);
        if (test_read(f_fp->fl_density) == OK) return(OK);
    }

    /* Either drive type is unknown or a different diskette is now present.
     * Use test_order to try them one by one.
     */
    for (top = &test_order[0]; top < &test_order[NT-1]; top++) {
        dtype = top->t_density;

        /* Skip densities that have been proven to be impossible */
        if (!(f_fp->fl_class & (1 << dtype))) continue;

        if (test_read(dtype) == OK) {
            /* The test succeeded, use this knowledge to limit the
             * drive class to match the density just read.
             */
            f_fp->fl_class &= top->t_class;
            return(OK);
        }
        /* Test failed, wrong density or did it time out? */
        if (f_busy == BSY_WAKEN) break;
    }
    f_fp->fl_density = NO_DENS;
    return(EIO);    /* nothing worked */
}

/*=====
*                                     test_read                                     *
*=====*/

```

```
PRIVATE int test_read(density)
int density;
{
/* Try to read the highest numbered sector on cylinder 2. Not all floppy
 * types have as many sectors per track, and trying cylinder 2 finds the
 * ones that need double stepping.
 */
    int device;
    off_t position;
    iovec_t iovec1;
    int result;

    f_fp->fl_density = density;
    device = ((density + 1) << DEV_TYPE_SHIFT) + f_drive;

    (void) f_prepare(device);
    position = (off_t) f_dp->test << SECTOR_SHIFT;
    iovec1.iov_addr = (vir_bytes) tmp_buf;
    iovec1.iov_size = SECTOR_SIZE;
    result = f_transfer(SELF, DEV_GATHER, position, &iovec1, 1);

    if (iovec1.iov_size != 0) return(EIO);

    partition(&f_dtab, f_drive, P_FLOPPY, 0);
    return(OK);
}

/*=====
 *                               f_geometry                               *
 *=====*/
PRIVATE void f_geometry(entry)
struct partition *entry;
{
    entry->cylinders = f_dp->cyls;
    entry->heads = NR_HEADS;
    entry->sectors = f_sectors;
}
```

```
#include "../drivers.h"  
#include "../libdriver/driver.h"  
#include "../libdriver/drvlib.h"  
  
_PROTOTYPE(void main, (void));
```

```
# Makefile for Intel Pro/100 driver (FXP)
DRIVER = fxp

# directories
u = /usr
i = $u/include
s = $i/sys
m = $i/minix
b = $i/ibm
d = ..

# programs, flags, etc.
CC =      exec cc
CFLAGS = -I$i
LDFLAGS = -i
LIBS = -lsys -lsysutil -ltimers

OBJ = fxp.o mii.o

# build local binary
all build:      $(DRIVER)
$(DRIVER):      $(OBJ)
                $(CC) -o $@ $(LDFLAGS) $(OBJ) $(LIBS)
                install -S 4096 $(DRIVER)

# install with other drivers
install:      /usr/sbin/$(DRIVER)
/usr/sbin/$(DRIVER):  $(DRIVER)
                install -o root -cs $? $@

# clean up local files
clean:
                rm -f *.o *.bak $(DRIVER)

depend:
                /usr/bin/mkdep "$(CC) -E $(CPPFLAGS)" *.c > .depend

# Include generated dependencies.
include .depend
```

```

/*
 * fxp.c
 *
 * This file contains an ethernet device driver for Intel 82557, 82558,
 * 82559, 82550, and 82562 fast ethernet controllers.
 *
 * The valid messages and their parameters are:
 *
 *      m_type      DL_PORT      DL_PROC      DL_COUNT      DL_MODE      DL_ADDR
 *      /-----+-----+-----+-----+-----+-----/
 *      / HARDINT   /             /             /             /             /
 *      /-----+-----+-----+-----+-----+-----/
 *      / DL_WRITE  / port nr    / proc nr    / count      / mode       / address
 *      /-----+-----+-----+-----+-----+-----/
 *      / DL_WRITEV / port nr    / proc nr    / count      / mode       / address
 *      /-----+-----+-----+-----+-----+-----/
 *      / DL_READ   / port nr    / proc nr    / count      /             / address
 *      /-----+-----+-----+-----+-----+-----/
 *      / DL_READV  / port nr    / proc nr    / count      /             / address
 *      /-----+-----+-----+-----+-----+-----/
 *      / DL_INIT   / port nr    / proc nr    / mode       /             / address
 *      /-----+-----+-----+-----+-----+-----/
 *      / DL_GETSTAT / port nr    / proc nr    /             /             / address
 *      /-----+-----+-----+-----+-----+-----/
 *      / DL_STOP   / port_nr    /             /             /             /
 *      /-----+-----+-----+-----+-----+-----/
 *
 * The messages sent are:
 *
 *      m-type      DL_PORT      DL_PROC      DL_COUNT      DL_STAT      DL_CLOCK
 *      /-----+-----+-----+-----+-----+-----/
 *      / DL_TASK_REPLY / port nr  / proc nr  / rd-count  / err/stat / clock
 *      /-----+-----+-----+-----+-----+-----/
 *
 *      m_type      m3_i1      m3_i2      m3_ca1
 *      /-----+-----+-----+-----+-----/
 *      / DL_INIT_REPLY / port nr  / last port / ethernet addr /
 *      /-----+-----+-----+-----+-----/
 *
 * Created:      Nov 2004 by Philip Homburg <philip@f-mnx.phicoh.com>
 */

#include "../drivers.h"

#include <stdlib.h>
#include <net/hton.h>
#include <net/gen/ether.h>
#include <net/gen/eth_io.h>
#include <ibm/pci.h>

#include <timers.h>

#define tmra_ut          timer_t
#define tmra_inittimer(tp) tmr_inittimer(tp)
#define Proc_number(p)   proc_number(p)
#define debug            0
#define RAND_UPDATE      /**/
#define printW()          ((void)0)
#define vm_lphys2bus(p)   (p)

#include "assert.h"
#include "fxp.h"
#include "mii.h"

/* Number of receive buffers */
#define N_RX_BUF          40

/* Number of transmit buffers */
#define N_TX_BUF          4

/* I/O vectors are handled IOVEC_NR entries at a time. */
#define IOVEC_NR          16

/* Configuration */

```

```

#define FXP_ENVVAR      "FXPETH"

struct pcitab
{
    ul6_t vid;
    ul6_t did;
    int checkclass;
};

PRIVATE struct pcitab pcitab_fxp[]=
{
    { 0x8086, 0x1229, 0 },          /* Intel 82557, etc. */
    { 0x8086, 0x2449, 0 },          /* Intel 82801BA/BAM/CA/CAM */
    { 0x0000, 0x0000, 0 }
};

#define FXP_PORT_NR      1          /* Minix */

typedef int irq_hook_t;

/* Translate a pointer to a field in a structure to a pointer to the structure
 * itself. So it translates '&struct_ptr->field' back to 'struct_ptr'.
 */
#define structof(type, field, ptr) \
    ((type *) (((char *) (ptr)) - offsetof(type, field)))

#define MICROS_TO_TICKS(m)  (((m)*HZ/1000000)+1)

static timer_t *fxp_timers= NULL;
static clock_t fxp_next_timeout= 0;

static void micro_delay(unsigned long usecs);

/* ignore interrupt for the moment */
#define interrupt(x)      0

char buffer[70*1024];

typedef struct fxp
{
    port_t fxp_base_port;
    int fxp_mode;
    int fxp_got_int;
    int fxp_send_int;
    int fxp_flags;
    int fxp_client;
    int fxp_features;          /* Needed? */
    int fxp_irq;
    int fxp_type;              /* What kind of hardware */
    int fxp_ee_addrlen;        /* #EEPROM address bits */
    int fxp_tx_alive;
    int fxp_need_reset;

    /* Rx */
    vir_bytes fxp_read_s;
    int fxp_rx_nbuf;
    int fxp_rx_bufsize;
    struct rfd *fxp_rx_buf;
    phys_bytes fxp_rx_busaddr;
    int fxp_rx_head;
    int fxp_rx_need_restart;
    int fxp_need_conf;         /* Re-configure after draining send
                               * queue
                               */

    /* Tx */
    int fxp_tx_nbuf;
    int fxp_tx_bufsize;
    struct tx *fxp_tx_buf;
    phys_bytes fxp_tx_busaddr;
    int fxp_tx_idle;
    int fxp_tx_head;
    int fxp_tx_tail;

```



```

    int fxp_tx_threshold;

    /* Link status */
    int fxp_report_link;
    int fxp_link_up;
    int fxp_mii_busy;
    ul6_t fxp_mii_scr;

    /* PCI related */
    int fxp_seen; /* TRUE iff device available */
    u8_t fxp_pcibus;
    u8_t fxp_pcidev;
    u8_t fxp_pcifunc;

    /* 'large' items */
    irq_hook_t fxp_hook;
    ether_addr_t fxp_address;
    message fxp_rx_mess;
    message fxp_tx_mess;
    struct sc fxp_stat;
    u8_t fxp_conf_bytes[CC_BYTES_NR];
    char fxp_name[sizeof("fxp#n")];
    iovec_t fxp_iovec[IOVEC_NR];
}
fxp_t;

/* fxp_mode */
#define FM_DISABLED 0x0
#define FM_ENABLED 0x1

/* fxp_flags */
#define FF_EMPTY 0x000
#define FF_PACK_SENT 0x001
#define FF_PACK_RECV 0x002
#define FF_SEND_AVAIL 0x004
#define FF_READING 0x010
#define FF_PROMISC 0x040
#define FF_MULTI 0x080
#define FF_BROAD 0x100
#define FF_ENABLED 0x200

/* fxp_features */
#define FFE_NONE 0x0

/* fxp_type */
#define FT_UNKNOWN 0x0
#define FT_82557 0x1
#define FT_82558A 0x2
#define FT_82559 0x4

static fxp_t fxp_table[FXP_PORT_NR];

static int fxp_tasknr= ANY;
static ul6_t eth_ign_proto;
static tmra_ut fxp_watchdog;
static char *progrname;

extern int errno;

#define fxp_inb(port, offset) (do_inb((port) + (offset)))
#define fxp_inw(port, offset) (do_inw((port) + (offset)))
#define fxp_inl(port, offset) (do_inl((port) + (offset)))
#define fxp_outb(port, offset, value) (do_outb((port) + (offset), (value)))
#define fxp_outw(port, offset, value) (do_outw((port) + (offset), (value)))
#define fxp_outl(port, offset, value) (do_outl((port) + (offset), (value)))

_PROTOTYPE( static void fxp_init, (message *mp) );
_PROTOTYPE( static void fxp_pci_conf, (void) );
_PROTOTYPE( static int fxp_probe, (fxp_t *fp) );
_PROTOTYPE( static void fxp_conf_hw, (fxp_t *fp) );
_PROTOTYPE( static void fxp_init_hw, (fxp_t *fp) );
_PROTOTYPE( static void fxp_init_buf, (fxp_t *fp) );
_PROTOTYPE( static void fxp_reset_hw, (fxp_t *fp) );
_PROTOTYPE( static void fxp_confaddr, (fxp_t *fp) );

```

```

_PROTOTYPE( static void fxp_rec_mode, (fxp_t *fp) ) ;
_PROTOTYPE( static void fxp_writev, (message *mp, int from_int,
                                     int vectored) ) ;
_PROTOTYPE( static void fxp_readv, (message *mp, int from_int,
                                     int vectored) ) ;
_PROTOTYPE( static void fxp_do_conf, (fxp_t *fp) ) ;
_PROTOTYPE( static void fxp_cu_ptr_cmd, (fxp_t *fp, int cmd,
                                         phys_bytes bus_addr, int check_idle) ) ;
_PROTOTYPE( static void fxp_ru_ptr_cmd, (fxp_t *fp, int cmd,
                                         phys_bytes bus_addr, int check_idle) ) ;
_PROTOTYPE( static void fxp_restart_ru, (fxp_t *fp) ) ;
_PROTOTYPE( static void fxp_getstat, (message *mp) ) ;
_PROTOTYPE( static void fxp_getname, (message *mp) ) ;
_PROTOTYPE( static int fxp_handler, (fxp_t *fp) ) ;
_PROTOTYPE( static void fxp_check_ints, (fxp_t *fp) ) ;
_PROTOTYPE( static void fxp_watchdog_f, (timer_t *tp) ) ;
_PROTOTYPE( static int fxp_link_changed, (fxp_t *fp) ) ;
_PROTOTYPE( static void fxp_report_link, (fxp_t *fp) ) ;
_PROTOTYPE( static void fxp_stop, (void) ) ;
_PROTOTYPE( static void reply, (fxp_t *fp, int err, int may_block) ) ;
_PROTOTYPE( static void mess_reply, (message *req, message *reply) ) ;
_PROTOTYPE( static void put_userdata, (int user_proc,
                                       vir_bytes user_addr, vir_bytes count, void *loc_addr) ) ;
_PROTOTYPE( static ul6_t eeprom_read, (fxp_t *fp, int reg) ) ;
_PROTOTYPE( static void eeprom_addrsize, (fxp_t *fp) ) ;
_PROTOTYPE( static ul6_t mii_read, (fxp_t *fp, int reg) ) ;
_PROTOTYPE( static void fxp_set_timer, (timer_t *tp, clock_t delta,
                                       tmr_func_t watchdog) ) ;
_PROTOTYPE( static void fxp_expire_timers, (void) ) ;
_PROTOTYPE( static u8_t do_inb, (port_t port) ) ;
_PROTOTYPE( static u32_t do_inl, (port_t port) ) ;
_PROTOTYPE( static void do_outb, (port_t port, u8_t v) ) ;
_PROTOTYPE( static void do_outl, (port_t port, u32_t v) ) ;

/*=====
 *                               main                               *
 *=====*/
int main(int argc, char *argv[])
{
    message m;
    int i, r, tasknr;
    fxp_t *fp;
    long v;

    if ((fxp_tasknr= getprocnr())<0)
        panic("FXP", "couldn't get proc nr", errno);

    if (argc < 1)
        panic("FXP", "A head which at this time has no name", NO_NUM);
    (progname=strrchr(argv[0], '/')) ? progname++ : (progname=argv[0]);

    v= 0;
#if 0
    (void) env_parse("ETH_IGN_PROTO", "x", 0, &v, 0x0000L, 0xFFFFL);
#endif
    eth_ign_proto= htons((ul6_t) v);

#if 0
    /* What about memory allocation? */
    /* Claim buffer memory now under Minix, before MM takes it all. */
    for (fp= &fxp_table[0]; fp < fxp_table+FXP_PORT_NR; fp++)
        fxp_init_buf(fp);
#endif

    /* Try to notify inet that we are present (again) */
    r = _pm_findproc("inet", &tasknr);
    if (r == OK)
        notify(tasknr);

    while (TRUE)
    {
        if ((r= receive(ANY, &m)) != OK)
            panic("FXP", "receive failed", r);

        switch (m.m_type)

```

```

        {
        case DEV_PING:  notify(m.m_source);          continue;
        case DL_WRITEV: fxp_writev(&m, FALSE, TRUE); break;
        case DL_WRITE:  fxp_writev(&m, FALSE, FALSE); break;

    #if 0
        case DL_READ:   fxp_vread(&m, FALSE);        break;
    #endif

        case DL_READV:  fxp_readv(&m, FALSE, TRUE);  break;
        case DL_INIT:   fxp_init(&m);                break;
        case DL_GETSTAT: fxp_getstat(&m);            break;
        case DL_GETNAME: fxp_getname(&m);            break;
        case HARD_INT:
            for (i= 0, fp= &fxp_table[0]; i<FXP_PORT_NR; i++, fp++)
            {
                if (fp->fxp_mode != FM_ENABLED)
                    continue;
                fxp_handler(fp);

                r= sys_irgenable(&fp->fxp_hook);
                if (r != OK)
                    panic("FXP", "unable enable interrupts", r);

                if (!fp->fxp_got_int)
                    continue;
                fp->fxp_got_int= 0;
                assert(fp->fxp_flags & FF_ENABLED);
                fxp_check_ints(fp);
            }
            break;
        case SYS_SIG: {
            sigset_t sigset = m.NOTIFY_ARG;
            if (sigismember(&sigset, SIGKSTOP)) fxp_stop();
            break;
        }
        case PROC_EVENT: break;
        case SYN_ALARM: fxp_expire_timers();          break;
        default:
            panic("FXP", "illegal message", m.m_type);
        }
    }
}

/*=====
 *                               fxp_init                               *
 *=====*/
static void fxp_init(mp)
message *mp;
{
    static int first_time= 1;

    int port;
    fxp_t *fp;
    message reply_mess;

    if (first_time)
    {
        first_time= 0;
        fxp_pci_conf(); /* Configure PCI devices. */

        tmra_inittimer(&fxp_watchdog);
        tmr_arg(&fxp_watchdog)->ta_int= 0;
        fxp_set_timer(&fxp_watchdog, HZ, fxp_watchdog_f);
    }

    port = mp->DL_PORT;
    if (port < 0 || port >= FXP_PORT_NR)
    {
        reply_mess.m_type= DL_INIT_REPLY;
        reply_mess.m3_il= ENXIO;
        mess_reply(mp, &reply_mess);
        return;
    }
    fp= &fxp_table[port];
    if (fp->fxp_mode == FM_DISABLED)

```

```

{
    /* This is the default, try to (re)locate the device. */
    fxp_conf_hw(fp);
    if (fp->fxp_mode == FM_DISABLED)
    {
        /* Probe failed, or the device is configured off. */
        reply_mess.m_type= DL_INIT_REPLY;
        reply_mess.m3_i1= ENXIO;
        mess_reply(mp, &reply_mess);
        return;
    }
    if (fp->fxp_mode == FM_ENABLED)
        fxp_init_hw(fp);
    fxp_report_link(fp);
}

assert(fp->fxp_mode == FM_ENABLED);
assert(fp->fxp_flags & FF_ENABLED);

fp->fxp_flags &= ~(FF_PROMISC | FF_MULTI | FF_BROAD);

if (mp->DL_MODE & DL_PROMISC_REQ)
    fp->fxp_flags |= FF_PROMISC;
if (mp->DL_MODE & DL_MULTI_REQ)
    fp->fxp_flags |= FF_MULTI;
if (mp->DL_MODE & DL_BROAD_REQ)
    fp->fxp_flags |= FF_BROAD;

fp->fxp_client = mp->m_source;
fxp_rec_mode(fp);

reply_mess.m_type = DL_INIT_REPLY;
reply_mess.m3_i1 = mp->DL_PORT;
reply_mess.m3_i2 = FXP_PORT_NR;
*(ether_addr_t *) reply_mess.m3_cal = fp->fxp_address;

mess_reply(mp, &reply_mess);
}

/*=====
 *                               fxp_pci_conf                               *
 *=====*/
static void fxp_pci_conf()
{
    static char envvar[] = FXP_ENVVAR "#";
    static char envfmt[] = ":%d.d.d";

    int i, h;
    fxp_t *fp;
    long v;

    for (i= 0, fp= fxp_table; i<FXP_PORT_NR; i++, fp++)
    {
        strcpy(fp->fxp_name, "fxp#0");
        fp->fxp_name[4] += i;
        fp->fxp_seen= FALSE;
        fp->fxp_features= FFE_NONE;
        envvar[sizeof(FXP_ENVVAR)-1]= '0'+i;

        if (getenv(envvar) != NULL)
        {
            if (strcmp(getenv(envvar), "off") == 0)
            {
                fp->fxp_pcibus= 255;
                continue;
            }
            if (!env_prefix(envvar, "pci"))
                env_panic(envvar);
        }
    }

    v= 0;

    (void) env_parse(envvar, envfmt, 1, &v, 0, 255);
}

```

```

#endif
        fp->fxp_pcibus= v;
        v= 0;
#if 0
        (void) env_parse(envvar, envfmt, 2, &v, 0, 255);
#endif
        fp->fxp_pcidev= v;
        v= 0;
#if 0
        (void) env_parse(envvar, envfmt, 3, &v, 0, 255);
#endif
        fp->fxp_pcifunc= v;
    }

    pci_init();

    for (h= 1; h >= 0; h--) {
        for (i= 0, fp= fxp_table; i<FXP_PORT_NR; i++, fp++)
        {
            if (fp->fxp_pcibus == 255)
                continue;
            if (((fp->fxp_pcibus | fp->fxp_pcidev |
                fp->fxp_pcifunc) != 0) != h)
            {
                continue;
            }
            if (fxp_probe(fp))
                fp->fxp_seen= TRUE;
        }
    }
}

/*=====
 *                               fxp_probe                               *
 *=====*/
static int fxp_probe(fp)
fxp_t *fp;
{
    int i, r, devind, just_one;
    ul6_t vid, did;
    u32_t bar;
    u8_t ilr, rev;
    char *dname, *str;

    if ((fp->fxp_pcibus | fp->fxp_pcidev | fp->fxp_pcifunc) != 0)
    {
        /* Look for specific PCI device */
        r= pci_find_dev(fp->fxp_pcibus, fp->fxp_pcidev,
            fp->fxp_pcifunc, &devind);
        if (r == 0)
        {
            printf("%s: no PCI device found at %d.%d.%d\n",
                fp->fxp_name, fp->fxp_pcibus,
                fp->fxp_pcidev, fp->fxp_pcifunc);
            return FALSE;
        }
        pci_ids(devind, &vid, &did);
        just_one= TRUE;
    }
    else
    {
        r= pci_first_dev(&devind, &vid, &did);
        if (r == 0)
            return FALSE;
        just_one= FALSE;
    }

    for(;;)
    {
        for (i= 0; pcitab_fxp[i].vid != 0; i++)
        {
            if (pcitab_fxp[i].vid != vid)
                continue;
            if (pcitab_fxp[i].did != did)

```

```

        continue;
    if (pcitab_fxp[i].checkclass)
    {
        panic("FXP", "fxp_probe: class check not implemented",
            NO_NUM);
    }
    break;
}
if (pcitab_fxp[i].vid != 0)
    break;

if (just_one)
{
    printf(
"%s: wrong PCI device (%04x/%04x) found at %d.%d.%d\n",
        fp->fxp_name, vid, did,
        fp->fxp_pcibus,
        fp->fxp_pcidev, fp->fxp_pcifunc);
    return FALSE;
}

r= pci_next_dev(&devind, &vid, &did);
if (!r)
    return FALSE;
}

dname= pci_dev_name(vid, did);
#if VERBOSE
if (!dname)
    dname= "unknown device";
printf("%s: %s (%04x/%04x) at %s\n",
    fp->fxp_name, dname, vid, did, pci_slot_name(devind));
#endif
pci_reserve(devind);

bar= pci_attr_r32(devind, PCI_BAR_2) & 0xffffffff0;
if (bar < 0x400)
{
    panic("FXP", "fxp_probe: base address is not properly configured",
        NO_NUM);
}
fp->fxp_base_port= bar;

ilr= pci_attr_r8(devind, PCI_ILR);
fp->fxp_irq= ilr;
if (debug)
{
    printf("%s: using I/O address 0x%lx, IRQ %d\n",
        fp->fxp_name, (unsigned long)bar, ilr);
}

rev= pci_attr_r8(devind, PCI_REV);
str= NULL;
fp->fxp_type= FT_UNKNOWN;
switch(rev)
{
case FXP_REV_82557A:    str= "82557A";                /* 0x01 */
    fp->fxp_type= FT_82557;
    break;

case FXP_REV_82557B:    str= "82557B"; break;          /* 0x02 */
case FXP_REV_82557C:    str= "82557C"; break;          /* 0x03 */
case FXP_REV_82558A:    str= "82558A";                /* 0x04 */
    fp->fxp_type= FT_82558A;
    break;

case FXP_REV_82558B:    str= "82558B"; break;          /* 0x05 */
case FXP_REV_82559A:    str= "82559A"; break;          /* 0x06 */
case FXP_REV_82559B:    str= "82559B"; break;          /* 0x07 */
case FXP_REV_82559C:    str= "82559C";                /* 0x08 */
    fp->fxp_type= FT_82559;
    break;

case FXP_REV_82559ERA:  str= "82559ER-A";             /* 0x09 */
    fp->fxp_type= FT_82559;
    break;

case FXP_REV_82550_1:   str= "82550(1)";              /* 0x0C */

```

```

        fp->fxp_type= FT_82559;
        break;
    case FXP_REV_82550_2:    str= "82550(2)";          /* 0x0D */
        fp->fxp_type= FT_82559;
        break;
    case FXP_REV_82550_3:    str= "82550(3)";          /* 0x0E */
        fp->fxp_type= FT_82559;
        break;
    case FXP_REV_82551_1:    str= "82551(1)";          /* 0x0F */
        fp->fxp_type= FT_82559;
        break;
    case FXP_REV_82551_2:    str= "82551(2)";          /* 0x10 */
        fp->fxp_type= FT_82559;
        break;
    }

#ifdef VERBOSE
    if (str)
        printf("%s: device revision: %s\n", fp->fxp_name, str);
    else
        printf("%s: unknown revision: 0x%x\n", fp->fxp_name, rev);
#endif

    if (fp->fxp_type == FT_UNKNOWN)
    {
        printf("fxp_probe: device is not supported by this driver\n");
        return FALSE;
    }

    return TRUE;
}

/*=====
 *                               fxp_conf_hw                               *
 *=====*/
static void fxp_conf_hw(fp)
fxp_t *fp;
{
    int i;
    int mwi, ext_stat1, ext_stat2, lim_fifo, i82503, fc;

    fp->fxp_mode= FM_DISABLED;      /* Superfluous */

    if (!fp->fxp_seen)
        return;

    /* PCI device is present */
    fp->fxp_mode= FM_ENABLED;

    fp->fxp_flags= FF_EMPTY;
    fp->fxp_got_int= 0;
    fp->fxp_send_int= 0;
    fp->fxp_ee_addrlen= 0; /* Unknown */
    fp->fxp_need_reset= 0;
    fp->fxp_report_link= 0;
    fp->fxp_link_up= -1; /* Unknown */
    fp->fxp_mii_busy= 0;
    fp->fxp_read_s= 0;
    fp->fxp_rx_need_restart= 0;
    fp->fxp_need_conf= 0;
    fp->fxp_tx_head= 0;
    fp->fxp_tx_tail= 0;
    fp->fxp_tx_alive= 0;
    fp->fxp_tx_threshold= TXTT_MIN;

    /* Try to come up with a sensible configuration for the current
     * device. Unfortunately every device is different, defaults are
     * not always zero, and some fields are re-used with a completely
     * different interpretation. We start out with a sensible default
     * for all devices and then add device specific changes.
     */
    fp->fxp_conf_bytes[0]= CC_BYTES_NR;
    fp->fxp_conf_bytes[1]= CTL_DEFAULT | CRL_DEFAULT;
    fp->fxp_conf_bytes[2]= CAI_DEFAULT;

```

```

fp->fxp_conf_bytes[3]= 0;
fp->fxp_conf_bytes[4]= 0;
fp->fxp_conf_bytes[5]= 0;
fp->fxp_conf_bytes[6]= CCB6_ESC | CCB6_ETCB | CCB6_RES;
fp->fxp_conf_bytes[7]= CUR_1;
fp->fxp_conf_bytes[8]= CCB8_503_MII;
fp->fxp_conf_bytes[9]= 0;
fp->fxp_conf_bytes[10]= CLB_NORMAL | CPAL_DEFAULT | CCB10_NSAI |
                        CCB10_RES1;
fp->fxp_conf_bytes[11]= 0;
fp->fxp_conf_bytes[12]= CIS_DEFAULT;
fp->fxp_conf_bytes[13]= CCB13_DEFAULT;
fp->fxp_conf_bytes[14]= CCB14_DEFAULT;
fp->fxp_conf_bytes[15]= CCB15_RES1 | CCB15_RES2;
fp->fxp_conf_bytes[16]= CCB16_DEFAULT;
fp->fxp_conf_bytes[17]= CCB17_DEFAULT;
fp->fxp_conf_bytes[18]= CCB18_RES1 | CCB18_PFCT | CCB18_PE;
fp->fxp_conf_bytes[19]= CCB19_FDPE;
fp->fxp_conf_bytes[20]= CCB20_PFCL | CCB20_RES1;
fp->fxp_conf_bytes[21]= CCB21_RES21;

```

```

#if VERBOSE

```

```

    for (i= 0; i<CC_BYTES_NR; i++)
        printf("%d:%0x, ", i, fp->fxp_conf_bytes[i]);
    printf("\n");

```

```

#endif

```

```

mwi= 0;          /* Do we want "Memory Write and Invalidate"? */
ext_stat1= 0;    /* Do we want extended statistical counters? */
ext_stat2= 0;    /* Do we want even more statistical counters? */
lim_fifo= 0;     /* Limit number of frame in TX FIFO */
i82503= 0;       /* Older 10 Mbps interface on the 82557 */
fc= 0;          /* Flow control */

```

```

switch(fp->fxp_type)

```

```

{
case FT_82557:
    if (i82503)
    {
        fp->fxp_conf_bytes[8] &= ~CCB8_503_MII;
        fp->fxp_conf_bytes[15] |= CCB15_CRSCDT;
    }
    break;
case FT_82558A:
case FT_82559:
    if (mwi)
        fp->fxp_conf_bytes[3] |= CCB3_MWIE;
    if (ext_stat1)
        fp->fxp_conf_bytes[6] &= ~CCB6_ESC;
    if (ext_stat2)
        fp->fxp_conf_bytes[6] &= ~CCB6_TCOSC;
    if (lim_fifo)
        fp->fxp_conf_bytes[7] |= CCB7_2FFIFO;
    if (fc)
    {
        /* From FreeBSD driver */
        fp->fxp_conf_bytes[16]= 0x1f;
        fp->fxp_conf_bytes[17]= 0x01;

        fp->fxp_conf_bytes[19] |= CCB19_FDRSTAFC |
                                CCB19_FDRSTOFC;
    }

    fp->fxp_conf_bytes[18] |= CCB18_LROK;
    break;
default:
    panic("FXP", "fxp_conf_hw: bad device type", fp->fxp_type);
}

```

```

#if VERBOSE

```

```

    for (i= 0; i<CC_BYTES_NR; i++)
        printf("%d:%0x, ", i, fp->fxp_conf_bytes[i]);
    printf("\n");

```

```

#endif

```



```

}

/*=====
 *                               fxp_init_hw                               *
 *=====*/
static void fxp_init_hw(fp)
fxp_t *fp;
{
    int i, r, isr;
    port_t port;
    u32_t bus_addr;

    port= fp->fxp_base_port;

    fxp_init_buf(fp);

    fp->fxp_flags = FF_EMPTY;
    fp->fxp_flags |= FF_ENABLED;

    /* Set the interrupt handler and policy. Do not automatically
     * reenale interrupts. Return the IRQ line number on interrupts.
     */
    fp->fxp_hook = fp->fxp_irq;
    r= sys_irqsetpolicy(fp->fxp_irq, 0, &fp->fxp_hook);
    if (r != OK)
        panic("FXP", "sys_irqsetpolicy failed", r);

    fxp_reset_hw(fp);

    r= sys_irgenable(&fp->fxp_hook);
    if (r != OK)
        panic("FXP", "sys_irgenable failed", r);

    /* Reset PHY? */

    fxp_do_conf(fp);

    /* Set pointer to statistical counters */
    r= sys_umap(SELF, D, (vir_bytes)&fp->fxp_stat, sizeof(fp->fxp_stat),
        &bus_addr);
    if (r != OK)
        panic("FXP", "sys_umap failed", r);
    fxp_cu_ptr_cmd(fp, SC_CU_LOAD_DCA, bus_addr, TRUE /* check idle */);

    /* Ack previous interrupts */
    isr= fxp_inb(port, SCB_INT_STAT);
    fxp_outb(port, SCB_INT_STAT, isr);

    /* Enable interrupts */
    fxp_outb(port, SCB_INT_MASK, 0);

    fxp_ru_ptr_cmd(fp, SC_RU_START, fp->fxp_rx_busaddr,
        TRUE /* check idle */);

    fxp_confaddr(fp);
    if (debug)
    {
        printf("%s: Ethernet address ", fp->fxp_name);
        for (i= 0; i < 6; i++)
        {
            printf("%x%c", fp->fxp_address.ea_addr[i],
                i < 5 ? ':' : '\n');
        }
    }
}

/*=====
 *                               fxp_init_buf                               *
 *=====*/
static void fxp_init_buf(fp)
fxp_t *fp;
{
    size_t rx_totbufsize, tx_totbufsize, tot_bufsize;
    phys_bytes buf;

```

```

    int i, r;
    struct rfd *rfdp;
    struct tx *txp;

    fp->fxp_rx_nbuf= N_RX_BUF;
    rx_totbufsize= fp->fxp_rx_nbuf * sizeof(struct rfd);
    fp->fxp_rx_bufsize= rx_totbufsize;

    fp->fxp_tx_nbuf= N_TX_BUF;
    tx_totbufsize= fp->fxp_tx_nbuf * sizeof(struct tx);
    fp->fxp_tx_bufsize= tx_totbufsize;

    tot_bufsize= tx_totbufsize + rx_totbufsize;

    /* What about memory allocation? */
    {
        static int first_time= 1;

        assert(first_time);
        first_time= 0;
    }

#define BUFALIGN      4096
    assert(tot_bufsize <= sizeof(buffer)-BUFALIGN);
    buf= (phys_bytes)buffer;
    buf += BUFALIGN - (buf % BUFALIGN);
}

fp->fxp_rx_buf= (struct rfd *)buf;
r= sys_umap(SELF, D, (vir_bytes)buf, rx_totbufsize,
    &fp->fxp_rx_busaddr);
if (r != OK)
    panic("FXP", "sys_umap failed", r);
for (i= 0, rfdp= fp->fxp_rx_buf; i<fp->fxp_rx_nbuf; i++, rfdp++)
{
    rfdp->rfd_status= 0;
    rfdp->rfd_command= 0;
    if (i != fp->fxp_rx_nbuf-1)
    {
        r= sys_umap(SELF, D, (vir_bytes)&rfdp[1],
            sizeof(rfdp[1]), &rfdp->rfd_linkaddr);
        if (r != OK)
            panic("FXP", "sys_umap failed", r);
    }
    else
    {
        rfdp->rfd_linkaddr= fp->fxp_rx_busaddr;
        rfdp->rfd_command |= RFDC_EL;
    }
    rfdp->rfd_reserved= 0;
    rfdp->rfd_res= 0;
    rfdp->rfd_size= sizeof(rfdp->rfd_buf);
}
fp->fxp_rx_head= 0;

fp->fxp_tx_buf= (struct tx *) (buf+rx_totbufsize);
r= sys_umap(SELF, D, (vir_bytes)fp->fxp_tx_buf,
    (phys_bytes)tx_totbufsize, &fp->fxp_tx_busaddr);
if (r != OK)
    panic("FXP", "sys_umap failed", r);

for (i= 0, txp= fp->fxp_tx_buf; i<fp->fxp_tx_nbuf; i++, txp++)
{
    txp->tx_status= 0;
    txp->tx_command= TXC_EL | CBL_NOP;      /* Just in case */
    if (i != fp->fxp_tx_nbuf-1)
    {
        r= sys_umap(SELF, D, (vir_bytes)&txp[1],
            (phys_bytes)sizeof(txp[1]),
            &txp->tx_linkaddr);
        if (r != OK)
            panic("FXP", "sys_umap failed", r);
    }
    else

```

```

        {
            txp->tx_linkaddr= fp->fxp_tx_busaddr;
        }
        txp->tx_tbd= TX_TBDA_NIL;
        txp->tx_size= 0;
        txp->tx_tthresh= fp->fxp_tx_threshold;
        txp->tx_ntbd= 0;
    }
    fp->fxp_tx_idle= 1;
}

/*=====
 *                               fxp_reset_hw                               *
 *=====*/
static void fxp_reset_hw(fp)
fxp_t *fp;
{
    /* Inline the function in init? */
    port_t port;

    port= fp->fxp_base_port;

    /* Reset device */
    fxp_outl(port, CSR_PORT, CP_CMD_SOFT_RESET);
    tickdelay(MICROS_TO_TICKS(CSR_PORT_RESET_DELAY));

    /* Disable interrupts */
    fxp_outb(port, SCB_INT_MASK, SIM_M);

    /* Set CU base to zero */
    fxp_cu_ptr_cmd(fp, SC_CU_LOAD_BASE, 0, TRUE /* check idle */);

    /* Set RU base to zero */
    fxp_ru_ptr_cmd(fp, SC_RU_LOAD_BASE, 0, TRUE /* check idle */);
}

/*=====
 *                               fxp_confaddr                               *
 *=====*/
static void fxp_confaddr(fp)
fxp_t *fp;
{
    static char eakey[]= FXP_ENVVAR "#_EA";
    static char eafmt[]= "x:x:x:x:x";
    clock_t t0,t1;
    int i, r;
    port_t port;
    u32_t bus_addr;
    long v;
    struct ias ias;

    port= fp->fxp_base_port;

    /* User defined ethernet address? */
    eakey[sizeof(FXP_ENVVAR)-1]= '0' + (fp->fxp_table);

#if 0
    for (i= 0; i < 6; i++)
    {
        if (env_parse(eakey, eafmt, i, &v, 0x00L, 0xFFL) != EP_SET)
            break;
        fp->fxp_address.ea_addr[i]= v;
    }
#else
    i= 0;
#endif

#if 0
    if (i != 0 && i != 6) env_panic(eakey); /* It's all or nothing */
#endif

    if (i == 0)
    {
        /* Get ethernet address from EEPROM */

```

```

        for (i= 0; i<3; i++)
        {
            v= eeprom_read(fp, i);
            fp->fxp_address.ea_addr[i*2]= (v & 0xff);
            fp->fxp_address.ea_addr[i*2+1]= ((v >> 8) & 0xff);
        }
    }

    /* Tell NIC about ethernet address */
    ias.ias_status= 0;
    ias.ias_command= CBL_C_EL | CBL_AIS;
    ias.ias_linkaddr= 0;
    memcpy(ias.ias_ethaddr, fp->fxp_address.ea_addr,
           sizeof(ias.ias_ethaddr));
    r= sys_umap(SELFB, D, (vir_bytes)&ias, (phys_bytes)sizeof(ias),
               &bus_addr);
    if (r != OK)
        panic("FXP", "sys_umap failed", r);

    fxp_cu_ptr_cmd(fp, SC_CU_START, bus_addr, TRUE /* check idle */);

    getuptime(&t0);
    do {
        /* Wait for CU command to complete */
        if (ias.ias_status & CBL_F_C)
            break;
    } while (getuptime(&t1)==OK && (t1-t0) < MICROS_TO_TICKS(1000));

    if (!(ias.ias_status & CBL_F_C))
        panic("FXP", "fxp_confaddr: CU command failed to complete", NO_NUM);
    if (!(ias.ias_status & CBL_F_OK))
        panic("FXP", "fxp_confaddr: CU command failed", NO_NUM);

#ifdef VERBOSE
    printf("%s: hardware ethernet address: ", fp->fxp_name);
    for (i= 0; i<6; i++)
    {
        printf("%02x%s", fp->fxp_address.ea_addr[i],
               i < 5 ? ":" : "");
    }
    printf("\n");
#endif
}

/*=====
 *                               fxp_rec_mode                               *
 *=====*/
static void fxp_rec_mode(fp)
fxp_t *fp;
{
    fp->fxp_conf_bytes[0]= CC_BYTES_NR;      /* Just to be sure */
    fp->fxp_conf_bytes[15] &= ~(CCB15_BD|CCB15_PM);
    fp->fxp_conf_bytes[21] &= ~CCB21_MA;

    if (fp->fxp_flags & FF_PROMISC)
        fp->fxp_conf_bytes[15] |= CCB15_PM;
    if (fp->fxp_flags & FF_MULTII)
        fp->fxp_conf_bytes[21] |= CCB21_MA;

    if (!(fp->fxp_flags & (FF_BROAD|FF_MULTII|FF_PROMISC)))
        fp->fxp_conf_bytes[15] |= CCB15_BD;

    /* Queue request if not idle */
    if (fp->fxp_tx_idle)
    {
        fxp_do_conf(fp);
    }
    else
    {
        printf("fxp_rec_mode: setting fxp_need_conf\n");
        fp->fxp_need_conf= TRUE;
    }
}

```

```

/*=====
 *                               fxp_writev                               *
 *=====*/
static void fxp_writev(mp, from_int, vectored)
message *mp;
int from_int;
int vectored;
{
    vir_bytes iov_src;
    int i, j, n, o, r, s, dl_port, count, size, prev_head;
    int fxp_client, fxp_tx_nbuf, fxp_tx_head;
    ul6_t tx_command;
    fxp_t *fp;
    iovec_t *iovp;
    struct tx *txp, *prev_txp;

    dl_port = mp->DL_PORT;
    count = mp->DL_COUNT;
    if (dl_port < 0 || dl_port >= FXP_PORT_NR)
        panic("FXP", "fxp_writev: illegal port", dl_port);
    fp = &fxp_table[dl_port];
    fxp_client = mp->DL_PROC;
    fp->fxp_client = fxp_client;

    assert(fp->fxp_mode == FM_ENABLED);
    assert(fp->fxp_flags & FF_ENABLED);

    if (from_int)
    {
        assert(fp->fxp_flags & FF_SEND_AVAIL);
        fp->fxp_flags &= ~FF_SEND_AVAIL;
        fp->fxp_tx_alive = TRUE;
    }

    if (fp->fxp_tx_idle)
    {
        txp = fp->fxp_tx_buf;
        fxp_tx_head = 0; /* lint */
        prev_txp = NULL; /* lint */
    }
    else
    {
        fxp_tx_nbuf = fp->fxp_tx_nbuf;
        prev_head = fp->fxp_tx_head;
        fxp_tx_head = prev_head + 1;
        if (fxp_tx_head == fxp_tx_nbuf)
            fxp_tx_head = 0;
        assert(fxp_tx_head < fxp_tx_nbuf);

        if (fxp_tx_head == fp->fxp_tx_tail)
        {
            /* Send queue is full */
            assert(!(fp->fxp_flags & FF_SEND_AVAIL));
            fp->fxp_flags |= FF_SEND_AVAIL;
            goto suspend;
        }

        prev_txp = &fp->fxp_tx_buf[prev_head];
        txp = &fp->fxp_tx_buf[fxp_tx_head];
    }

    assert(!(fp->fxp_flags & FF_SEND_AVAIL));
    assert(!(fp->fxp_flags & FF_PACK_SENT));

    if (vectored)
    {
        iov_src = (vir_bytes)mp->DL_ADDR;

        size = 0;
        o = 0;
        for (i = 0; i < count; i += IOVEC_NR,
            iov_src += IOVEC_NR * sizeof(fp->fxp_iovec[0]))
    {

```

```

        n= IOVEC_NR;
        if (i+n > count)
            n= count-i;
        r= sys_vircopy(fxp_client, D, iov_src,
            SELF, D, (vir_bytes)fp->fxp_iovec,
            n * sizeof(fp->fxp_iovec[0]));
        if (r != OK)
            panic("FXP", "fxp_writev: sys_vircopy failed", r);

        for (j= 0, iovp= fp->fxp_iovec; j<n; j++, iovp++)
        {
            s= iovp->iov_size;
            if (size + s > ETH_MAX_PACK_SIZE_TAGGED)
            {
                panic("FXP", "fxp_writev: invalid packet size",
                    NO_NUM);
            }

            r= sys_vircopy(fxp_client, D, iovp->iov_addr,
                SELF, D, (vir_bytes)(txp->tx_buf+o),
                s);
            if (r != OK)
            {
                panic("FXP", "fxp_writev: sys_vircopy failed",
                    r);
            }
            size += s;
            o += s;
        }
    }
    if (size < ETH_MIN_PACK_SIZE)
        panic("FXP", "fxp_writev: invalid packet size", size);
}
else
{
    size= mp->DL_COUNT;
    if (size < ETH_MIN_PACK_SIZE || size > ETH_MAX_PACK_SIZE_TAGGED)
        panic("FXP", "fxp_writev: invalid packet size", size);

    r= sys_vircopy(fxp_client, D, (vir_bytes)mp->DL_ADDR,
        SELF, D, (vir_bytes)txp->tx_buf, size);
    if (r != OK)
        panic("FXP", "fxp_writev: sys_vircopy failed", r);
}

txp->tx_status= 0;
txp->tx_command= TXC_EL | CBL_XMIT;
txp->tx_tbda= TX_TBDA_NIL;
txp->tx_size= TXSZ_EOF | size;
txp->tx_tthresh= fp->fxp_tx_threshold;
txp->tx_ntbd= 0;
if (fp->fxp_tx_idle)
{
    fp->fxp_tx_idle= 0;
    fp->fxp_tx_head= fp->fxp_tx_tail= 0;

    fxp_cu_ptr_cmd(fp, SC_CU_START, fp->fxp_tx_busaddr,
        TRUE /* check idle */);
}
else
{
    /* Link new request in transmit list */
    tx_command= prev_txp->tx_command;
    assert(tx_command == (TXC_EL | CBL_XMIT));
    prev_txp->tx_command= CBL_XMIT;
    fp->fxp_tx_head= fxp_tx_head;
}

fp->fxp_flags |= FF_PACK_SENT;

/* If the interrupt handler called, don't send a reply. The reply
 * will be sent after all interrupts are handled.
 */
if (from_int)

```

```

        return;
    reply(fp, OK, FALSE);
    return;
suspend:
    if (from_int)
        panic("FXP", "fxp: should not be sending\n", NO_NUM);

    fp->fxp_tx_mess= *mp;
    reply(fp, OK, FALSE);
}

/*=====
 *                               fxp_readv                               *
 *=====*/
static void fxp_readv(mp, from_int, vectored)
message *mp;
int from_int;
int vectored;
{
    int i, j, n, o, r, s, dl_port, fxp_client, count, size,
        fxp_rx_head, fxp_rx_nbuf;
    port_t port;
    unsigned packlen;
    vir_bytes iov_src;
    ul6_t rfd_status;
    ul6_t rfd_res;
    u8_t scb_status;
    fxp_t *fp;
    iovec_t *iovp;
    struct rfd *rfdp, *prev_rfdp;

    dl_port = mp->DL_PORT;
    count = mp->DL_COUNT;
    if (dl_port < 0 || dl_port >= FXP_PORT_NR)
        panic("FXP", "fxp_readv: illegal port", dl_port);
    fp= &fxp_table[dl_port];
    fxp_client= mp->DL_PROC;
    fp->fxp_client= fxp_client;

    assert(fp->fxp_mode == FM_ENABLED);
    assert(fp->fxp_flags & FF_ENABLED);

    port= fp->fxp_base_port;

    fxp_rx_head= fp->fxp_rx_head;
    rfdp= &fp->fxp_rx_buf[fxp_rx_head];

    rfd_status= rfdp->rfd_status;
    if (!(rfd_status & RFDS_C))
    {
        /* Receive buffer is empty, suspend */
        goto suspend;
    }

    if (!(rfd_status & RFDS_OK))
    {
        /* Not OK? What happened? */
        assert(0);
    }
    else
    {
        assert(!(rfd_status & (RFDS_CRCERR | RFDS_ALIGNERR |
            RFDS_OUTOFBUF | RFDS_DMAOVR | RFDS_TOOSHORT |
            RFDS_RXERR)));
    }
    rfd_res= rfdp->rfd_res;
    assert(rfd_res & RFDR_EOF);
    assert(rfd_res & RFDR_F);

    packlen= rfd_res & RFDSZ_SIZE;

    if (vectored)
    {

```

```

    iov_src = (vir_bytes)mp->DL_ADDR;

    size= 0;
    o= 0;
    for (i= 0; i<count; i += IOVEC_NR,
        iov_src += IOVEC_NR * sizeof(fp->fxp_iovec[0]))
    {
        n= IOVEC_NR;
        if (i+n > count)
            n= count-i;
        r= sys_vircopy(fxp_client, D, iov_src,
            SELF, D, (vir_bytes)fp->fxp_iovec,
            n * sizeof(fp->fxp_iovec[0]));
        if (r != OK)
            panic("FXP", "fxp_readv: sys_vircopy failed", r);

        for (j= 0, iovp= fp->fxp_iovec; j<n; j++, iovp++)
        {
            s= iovp->iov_size;
            if (size + s > packlen)
            {
                assert(packlen > size);
                s= packlen-size;
            }

            r= sys_vircopy(SELF, D,
                (vir_bytes)(rfdp->rfd_buf+o),
                fxp_client, D, iovp->iov_addr, s);
            if (r != OK)
            {
                panic("FXP", "fxp_readv: sys_vircopy failed",
                    r);
            }

            size += s;
            if (size == packlen)
                break;
            o += s;
        }
        if (size == packlen)
            break;
    }
    if (size < packlen)
    {
        assert(0);
    }
}
else
{
    assert(0);
}

fp->fxp_read_s= packlen;
fp->fxp_flags= (fp->fxp_flags & ~FF_READING) | FF_PACK_RECV;

/* Re-init the current buffer */
rfdp->rfd_status= 0;
rfdp->rfd_command= RFDC_EL;
rfdp->rfd_reserved= 0;
rfdp->rfd_res= 0;
rfdp->rfd_size= sizeof(rfdp->rfd_buf);

fxp_rx_nbuf= fp->fxp_rx_nbuf;
if (fxp_rx_head == 0)
{
    prev_rfdp= &fp->fxp_rx_buf[fxp_rx_nbuf-1];
}
else
    prev_rfdp= &rfdp[-1];

assert(prev_rfdp->rfd_command & RFDC_EL);
prev_rfdp->rfd_command &= ~RFDC_EL;

fxp_rx_head++;

```



```

    if (fxp_rx_head == fxp_rx_nbuf)
        fxp_rx_head = 0;
    assert(fxp_rx_head < fxp_rx_nbuf);
    fp->fxp_rx_head = fxp_rx_head;

    if (!from_int)
        reply(fp, OK, FALSE);

    return;
suspend:
    if (fp->fxp_rx_need_restart)
    {
        fp->fxp_rx_need_restart = 0;

        /* Check the status of the RU */
        scb_status = fxp_inb(port, SCB_STATUS);
        if ((scb_status & SS_RUS_MASK) != SS_RU_NORES)
        {
            /* Race condition? */
            printf("fxp_readv: restart race: 0x%x\n",
                scb_status);
            assert((scb_status & SS_RUS_MASK) == SS_RU_READY);
        }
        else
        {
            fxp_restart_ru(fp);
        }
    }
    if (from_int)
    {
        assert(fp->fxp_flags & FF_READING);

        /* No need to store any state */
        return;
    }

    fp->fxp_rx_mess = *mp;
    assert(!(fp->fxp_flags & FF_READING));
    fp->fxp_flags |= FF_READING;

    reply(fp, OK, FALSE);
}

/*=====
 *                               fxp_do_conf                               *
 *=====*/
static void fxp_do_conf(fp)
fxp_t *fp;
{
    int r;
    u32_t bus_addr;
    struct cbl_conf cc;
    clock_t t0, t1;

    /* Configure device */
    cc.cc_status = 0;
    cc.cc_command = CBL_C_EL | CBL_CONF;
    cc.cc_linkaddr = 0;
    memcpy(cc.cc_bytes, fp->fxp_conf_bytes, sizeof(cc.cc_bytes));

    r = sys_umap(SELF, D, (vir_bytes)&cc, (phys_bytes)sizeof(cc),
        &bus_addr);
    if (r != OK)
        panic("FXP", "sys_umap failed", r);

    fxp_cu_ptr_cmd(fp, SC_CU_START, bus_addr, TRUE /* check idle */);

    getuptime(&t0);
    do {
        /* Wait for CU command to complete */
        if (cc.cc_status & CBL_F_C)
            break;
    } while (getuptime(&t1) == OK && (t1 - t0) < MICROS_TO_TICKS(100000));

```

```

    if (!(cc.cc_status & CBL_F_C))
        panic("FXP", "fxp_do_conf: CU command failed to complete", NO_NUM);
    if (!(cc.cc_status & CBL_F_OK))
        panic("FXP", "fxp_do_conf: CU command failed", NO_NUM);
}

/*=====
 *                               fxp_cu_ptr_cmd                               *
 *=====*/
static void fxp_cu_ptr_cmd(fp, cmd, bus_addr, check_idle)
fxp_t *fp;
int cmd;
phys_bytes bus_addr;
int check_idle;
{
    clock_t t0,t1;
    port_t port;
    u8_t scb_cmd;

    port= fp->fxp_base_port;

    if (check_idle)
    {
        /* Consistency check. Make sure that CU is idle */
        if ((fxp_inb(port, SCB_STATUS) & SS_CUS_MASK) != SS_CU_IDLE)
            panic("FXP", "fxp_cu_ptr_cmd: CU is not idle", NO_NUM);
    }

    fxp_outl(port, SCB_POINTER, bus_addr);
    fxp_outb(port, SCB_CMD, cmd);

    /* What is a reasonable time-out? There is nothing in the
     * documentation. 1 ms should be enough.
     */
    getuptime(&t0);
    do {
        /* Wait for CU command to be accepted */
        scb_cmd= fxp_inb(port, SCB_CMD);
        if ((scb_cmd & SC_CUC_MASK) == SC_CU_NOP)
            break;
    } while (getuptime(&t1)==OK && (t1-t0) < MICROS_TO_TICKS(100000));

    if ((scb_cmd & SC_CUC_MASK) != SC_CU_NOP)
        panic("FXP", "fxp_cu_ptr_cmd: CU does not accept command", NO_NUM);
}

/*=====
 *                               fxp_ru_ptr_cmd                               *
 *=====*/
static void fxp_ru_ptr_cmd(fp, cmd, bus_addr, check_idle)
fxp_t *fp;
int cmd;
phys_bytes bus_addr;
int check_idle;
{
    clock_t t0,t1;
    port_t port;
    u8_t scb_cmd;

    port= fp->fxp_base_port;

    if (check_idle)
    {
        /* Consistency check, make sure that RU is idle */
        if ((fxp_inb(port, SCB_STATUS) & SS_RUS_MASK) != SS_RU_IDLE)
            panic("FXP", "fxp_ru_ptr_cmd: RU is not idle", NO_NUM);
    }

    fxp_outl(port, SCB_POINTER, bus_addr);
    fxp_outb(port, SCB_CMD, cmd);

    getuptime(&t0);

```

```

    do {
        /* Wait for RU command to be accepted */
        scb_cmd= fxp_inb(port, SCB_CMD);
        if ((scb_cmd & SC_RUC_MASK) == SC_RU_NOP)
            break;
    } while (getuptime(&t1)==OK && (t1-t0) < MICROS_TO_TICKS(1000));

    if ((scb_cmd & SC_RUC_MASK) != SC_RU_NOP)
        panic("FXP", "fxp_ru_ptr_cmd: RU does not accept command", NO_NUM);
}

/*=====
 *                               fxp_restart_ru                               *
 *=====*/
static void fxp_restart_ru(fp)
fxp_t *fp;
{
    int i, fxp_rx_nbuf;
    port_t port;
    struct rfd *rfdp;

    port= fp->fxp_base_port;

    fxp_rx_nbuf= fp->fxp_rx_nbuf;
    for (i= 0, rfdp= fp->fxp_rx_buf; i<fxp_rx_nbuf; i++, rfdp++)
    {
        rfdp->rfd_status= 0;
        rfdp->rfd_command= 0;
        if (i == fp->fxp_rx_nbuf-1)
            rfdp->rfd_command= RFDC_EL;
        rfdp->rfd_reserved= 0;
        rfdp->rfd_res= 0;
        rfdp->rfd_size= sizeof(rfdp->rfd_buf);
    }
    fp->fxp_rx_head= 0;

    /* Make sure that RU is in the 'No resources' state */
    if ((fxp_inb(port, SCB_STATUS) & SS_RUS_MASK) != SS_RU_NORES)
        panic("FXP", "fxp_restart_ru: RU is in an unexpected state", NO_NUM);

    fxp_ru_ptr_cmd(fp, SC_RU_START, fp->fxp_rx_busaddr,
        FALSE /* do not check idle */);
}

/*=====
 *                               fxp_getstat                               *
 *=====*/
static void fxp_getstat(mp)
message *mp;
{
    clock_t t0,t1;
    int dl_port;
    port_t port;
    fxp_t *fp;
    u32_t *p;
    eth_stat_t stats;

    dl_port = mp->DL_PORT;
    if (dl_port < 0 || dl_port >= FXP_PORT_NR)
        panic("FXP", "fxp_getstat: illegal port", dl_port);
    fp= &fxp_table[dl_port];
    fp->fxp_client= mp->DL_PROC;

    assert(fp->fxp_mode == FM_ENABLED);
    assert(fp->fxp_flags & FF_ENABLED);

    port= fp->fxp_base_port;

    p= &fp->fxp_stat.sc_tx_fcp;
    *p= 0;

    /* The dump command doesn't take a pointer. Setting a pointer
     * doesn't hard though.
     */
}

```

```

fxp_cu_ptr_cmd(fp, SC_CU_DUMP_SC, 0, FALSE /* do not check idle */);

getuptime(&t0);
do {
    /* Wait for CU command to complete */
    if (*p != 0)
        break;
} while (getuptime(&t1)==OK && (t1-t0) < MICROS_TO_TICKS(1000));

if (*p == 0)
    panic("FXP", "fxp_getstat: CU command failed to complete", NO_NUM);
if (*p != SCM_DSC)
    panic("FXP", "fxp_getstat: bad magic", NO_NUM);

stats.ets_recvErr=
    fp->fxp_stat.sc_rx_crc +
    fp->fxp_stat.sc_rx_align +
    fp->fxp_stat.sc_rx_resource +
    fp->fxp_stat.sc_rx_overrun +
    fp->fxp_stat.sc_rx_cd +
    fp->fxp_stat.sc_rx_short;
stats.ets_sendErr=
    fp->fxp_stat.sc_tx_maxcol +
    fp->fxp_stat.sc_tx_latecol +
    fp->fxp_stat.sc_tx_crs;
stats.ets_OVW= fp->fxp_stat.sc_rx_overrun;
stats.ets_CRCerr= fp->fxp_stat.sc_rx_crc;
stats.ets_frameAll= fp->fxp_stat.sc_rx_align;
stats.ets_missedP= fp->fxp_stat.sc_rx_resource;
stats.ets_packetR= fp->fxp_stat.sc_rx_good;
stats.ets_packetT= fp->fxp_stat.sc_tx_good;
stats.ets_transDef= fp->fxp_stat.sc_tx_defered;
stats.ets_collision= fp->fxp_stat.sc_tx_totcol;
stats.ets_transAb= fp->fxp_stat.sc_tx_maxcol;
stats.ets_carrSense= fp->fxp_stat.sc_tx_crs;
stats.ets_fifoUnder= fp->fxp_stat.sc_tx_underrun;
stats.ets_fifoOver= fp->fxp_stat.sc_rx_overrun;
stats.ets_CDheartbeat= 0;
stats.ets_OWC= fp->fxp_stat.sc_tx_latecol;

put_userdata(mp->DL_PROC, (vir_bytes) mp->DL_ADDR,
    (vir_bytes) sizeof(stats), &stats);
reply(fp, OK, FALSE);
}

/*=====
 *                               fxp_getname                               *
 *=====*/
static void fxp_getname(mp)
message *mp;
{
    int r;

    strncpy(mp->DL_NAME, progname, sizeof(mp->DL_NAME));
    mp->DL_NAME[sizeof(mp->DL_NAME)-1] = '\0';
    mp->m_type= DL_NAME_REPLY;
    r= send(mp->m_source, mp);
    if (r != OK)
        panic("FXP", "fxp_getname: send failed", r);
}

/*=====
 *                               fxp_handler                               *
 *=====*/
static int fxp_handler(fp)
fxp_t *fp;
{
    int port;
    ul6_t isr;

    RAND_UPDATE

    port= fp->fxp_base_port;

```

```

/* Ack interrupt */
isr= fxp_inb(port, SCB_INT_STAT);
fxp_outb(port, SCB_INT_STAT, isr);

if (isr & SIS_FR)
{
    isr &= ~SIS_FR;

    if (!fp->fxp_got_int && (fp->fxp_flags & FF_READING))
    {
        fp->fxp_got_int= TRUE;
        interrupt(fxp_tasknr);
    }
}
if (isr & SIS_CNA)
{
    isr &= ~SIS_CNA;
    if (!fp->fxp_tx_idle)
    {
        fp->fxp_send_int= TRUE;
        if (!fp->fxp_got_int)
        {
            fp->fxp_got_int= TRUE;
            interrupt(fxp_tasknr);
        }
    }
}
if (isr & SIS_RNR)
{
    isr &= ~SIS_RNR;

    /* Assume that receive buffer is full of packets. fxp_readv
     * will restart the RU.
     */
    fp->fxp_rx_need_restart= 1;
}
if (isr)
{
    printf("fxp_handler: unhandled interrupt: isr = 0x%02x\n",
           isr);
}

return 1;
}

/*=====
 *                               fxp_check_ints                               *
 *=====*/
static void fxp_check_ints(fp)
fxp_t *fp;
{
    int n, fxp_flags, prev_tail;
    int fxp_tx_tail, fxp_tx_nbuf, fxp_tx_threshold;
    port_t port;
    u32_t busaddr;
    u16_t tx_status;
    u8_t scb_status;
    struct tx *txp;

    fxp_flags= fp->fxp_flags;

    if (fxp_flags & FF_READING)
    {
        if (!(fp->fxp_rx_buf[fp->fxp_rx_head].rfd_status & RFDS_C))
            ; /* Nothing */
        else if (fp->fxp_rx_mess.m_type == DL_READV)
        {
            fxp_readv(&fp->fxp_rx_mess, TRUE /* from int */,
                      TRUE /* vectored */);
        }
        else
        {
            assert(fp->fxp_rx_mess.m_type == DL_READ);
        }
    }
}

```

```

        fxp_readv(&fp->fxp_rx_mess, TRUE /* from int */,
                  FALSE /* !vectored */);
    }
}
if (fp->fxp_tx_idle)
    ; /* Nothing to do */
else if (fp->fxp_send_int)
{
    fp->fxp_send_int= FALSE;
    fxp_tx_tail= fp->fxp_tx_tail;
    fxp_tx_nbuf= fp->fxp_tx_nbuf;
    n= 0;
    for (;;)
    {
        txp= &fp->fxp_tx_buf[fxp_tx_tail];
        tx_status= txp->tx_status;
        if (!(tx_status & TXS_C))
            break;

        n++;

        assert(tx_status & TXS_OK);
        if (tx_status & TXS_U)
        {
            fxp_tx_threshold= fp->fxp_tx_threshold;
            if (fxp_tx_threshold < TXTT_MAX)
            {
                fxp_tx_threshold++;
                fp->fxp_tx_threshold= fxp_tx_threshold;
            }
            printf(
                "fxp_check_ints: fxp_tx_threshold = 0x%x\n",
                fxp_tx_threshold);
        }

        if (txp->tx_command & TXC_EL)
        {
            fp->fxp_tx_idle= 1;
            break;
        }

        fxp_tx_tail++;
        if (fxp_tx_tail == fxp_tx_nbuf)
            fxp_tx_tail= 0;
        assert(fxp_tx_tail < fxp_tx_nbuf);
    }

    if (fp->fxp_need_conf)
    {
        /* Check the status of the CU */
        port= fp->fxp_base_port;
        scb_status= fxp_inb(port, SCB_STATUS);
        if ((scb_status & SS_CUS_MASK) != SS_CU_IDLE)
        {
            /* Nothing to do */
            printf("scb_status = 0x%x\n", scb_status);
        }
        else
        {
            printf("fxp_check_ints: fxp_need_conf\n");
            fp->fxp_need_conf= FALSE;
            fxp_do_conf(fp);
        }
    }

    if (n)
    {
        if (!fp->fxp_tx_idle)
        {
            fp->fxp_tx_tail= fxp_tx_tail;

            /* Check the status of the CU */
            port= fp->fxp_base_port;
            scb_status= fxp_inb(port, SCB_STATUS);

```

```

        if ((scb_status & SS_CUS_MASK) != SS_CU_IDLE)
        {
            /* Nothing to do */
            printf("scb_status=0x%x\n",
                scb_status);
        }
    }
    else
    {
        if (fxp_tx_tail == 0)
            prev_tail= fxp_tx_nbuf-1;
        else
            prev_tail= fxp_tx_tail-1;
        busaddr= fp->fxp_tx_buf[prev_tail].
            tx_linkaddr;

        fxp_cu_ptr_cmd(fp, SC_CU_START,
            busaddr, 1 /* check idle */);
    }
}

if (fp->fxp_flags & FF_SEND_AVAIL)
{
    if (fp->fxp_tx_mess.m_type == DL_WRITEV)
    {
        fxp_writev(&fp->fxp_tx_mess,
            TRUE /* from int */,
            TRUE /* vectored */);
    }
    else
    {
        assert(fp->fxp_tx_mess.m_type ==
            DL_WRITE);
        fxp_writev(&fp->fxp_tx_mess,
            TRUE /* from int */,
            FALSE /* !vectored */);
    }
}

}

if (fp->fxp_report_link)
    fxp_report_link(fp);

if (fp->fxp_flags & (FF_PACK_SENT | FF_PACK_RECV))
    reply(fp, OK, TRUE);
}

/*=====
 *                               fxp_watchdog_f                               *
 *=====*/
static void fxp_watchdog_f(tp)
timer_t *tp;
{
    int i;
    fxp_t *fp;

    tmr_arg(&fxp_watchdog)->ta_int= 0;
    fxp_set_timer(&fxp_watchdog, HZ, fxp_watchdog_f);

    for (i= 0, fp = &fxp_table[0]; i<FXP_PORT_NR; i++, fp++)
    {
        if (fp->fxp_mode != FM_ENABLED)
            continue;

        /* Handle race condition, MII interface mgith be busy */
        if(!fp->fxp_mii_busy)
        {
            /* Check the link status. */
            if (fxp_link_changed(fp))
            {
                #if VERBOSE
                printf("fxp_watchdog_f: link changed\n");
                #endif
            }
        }
    }
}

```

```

        fp->fxp_report_link= TRUE;
        fp->fxp_got_int= TRUE;
        interrupt(fxp_tasknr);
    }
}

if (!(fp->fxp_flags & FF_SEND_AVAIL))
{
    /* Assume that an idle system is alive */
    fp->fxp_tx_alive= TRUE;
    continue;
}
if (fp->fxp_tx_alive)
{
    fp->fxp_tx_alive= FALSE;
    continue;
}

fp->fxp_need_reset= TRUE;
fp->fxp_got_int= TRUE;
interrupt(fxp_tasknr);
}
}

/*=====
 *                               fxp_link_changed                               *
 *=====*/
static int fxp_link_changed(fp)
fxp_t *fp;
{
    ul6_t scr;

    scr= mii_read(fp, MII_SCR);
    scr &= ~(MII_SCR_RES|MII_SCR_RES_1);

    return (fp->fxp_mii_scr != scr);
}

/*=====
 *                               fxp_report_link                               *
 *=====*/
static void fxp_report_link(fp)
fxp_t *fp;
{
    port_t port;
    ul6_t mii_ctrl, mii_status, mii_id1, mii_id2,
          mii_ana, mii_anlpa, mii_ane, mii_extstat,
          mii_ms_ctrl, mii_ms_status, scr;
    u32_t oui;
    int model, rev;
    int f, link_up, ms_regs;

    /* Assume an 82555 (compatible) PHY. The should be changed for
     * 82557 NICs with different PHYs
     */
    ms_regs= 0;      /* No master/slave registers. */

    fp->fxp_report_link= FALSE;
    port= fp->fxp_base_port;

    scr= mii_read(fp, MII_SCR);
    scr &= ~(MII_SCR_RES|MII_SCR_RES_1);
    fp->fxp_mii_scr= scr;

    mii_ctrl= mii_read(fp, MII_CTRL);
    mii_read(fp, MII_STATUS); /* Read the status register twice, why? */
    mii_status= mii_read(fp, MII_STATUS);
    mii_id1= mii_read(fp, MII_PHYID_H);
    mii_id2= mii_read(fp, MII_PHYID_L);
    mii_ana= mii_read(fp, MII_ANA);
    mii_anlpa= mii_read(fp, MII_ANLPA);
    mii_ane= mii_read(fp, MII_ANE);
    if (mii_status & MII_STATUS_EXT_STAT)
        mii_extstat= mii_read(fp, MII_EXT_STATUS);

```



```

    else
        mii_extstat= 0;
    if (ms_regs)
    {
        mii_ms_ctrl= mii_read(fp, MII_MS_CTRL);
        mii_ms_status= mii_read(fp, MII_MS_STATUS);
    }
    else
    {
        mii_ms_ctrl= 0;
        mii_ms_status= 0;
    }

    /* How do we know about the link status? */
    link_up= !(mii_status & MII_STATUS_LS);

    fp->fxp_link_up= link_up;
    if (!link_up)
    {
#if VERBOSE
        printf("%s: link down\n", fp->fxp_name);
#endif
        return;
    }

    oui= (mii_id1 << MII_PH_OUI_H_C_SHIFT) |
        ((mii_id2 & MII_PL_OUI_L_MASK) >> MII_PL_OUI_L_SHIFT);
    model= ((mii_id2 & MII_PL_MODEL_MASK) >> MII_PL_MODEL_SHIFT);
    rev= (mii_id2 & MII_PL_REV_MASK);

#if VERBOSE
    printf("OUI 0x%06lx, Model 0x%02x, Revision 0x%x\n", oui, model, rev);
#endif

    if (mii_ctrl & (MII_CTRL_LB|MII_CTRL_PD|MII_CTRL_ISO))
    {
        printf("%s: PHY: ", fp->fxp_name);
        f= 1;
        if (mii_ctrl & MII_CTRL_LB)
        {
            printf("loopback mode");
            f= 0;
        }
        if (mii_ctrl & MII_CTRL_PD)
        {
            if (!f) printf(",");
            f= 0;
            printf("powered down");
        }
        if (mii_ctrl & MII_CTRL_ISO)
        {
            if (!f) printf(",");
            f= 0;
            printf("isolated");
        }
        printf("\n");
        return;
    }
    if (!(mii_ctrl & MII_CTRL_ANE))
    {
        printf("%s: manual config: ", fp->fxp_name);
        switch(mii_ctrl & (MII_CTRL_SP_LSB|MII_CTRL_SP_MSB))
        {
            case MII_CTRL_SP_10:    printf("10 Mbps"); break;
            case MII_CTRL_SP_100:   printf("100 Mbps"); break;
            case MII_CTRL_SP_1000:  printf("1000 Mbps"); break;
            case MII_CTRL_SP_RES:    printf("reserved speed"); break;
        }
        if (mii_ctrl & MII_CTRL_DM)
            printf(", full duplex");
        else
            printf(", half duplex");
        printf("\n");
        return;
    }

```

```
}

if (!debug) goto resspeed;

printf("%s: ", fp->fxp_name);
mii_print_stat_speed(mii_status, mii_extstat);
printf("\n");

if (!(mii_status & MII_STATUS_ANC))
    printf("%s: auto-negotiation not complete\n", fp->fxp_name);
if (mii_status & MII_STATUS_RF)
    printf("%s: remote fault detected\n", fp->fxp_name);
if (!(mii_status & MII_STATUS_ANA))
{
    printf("%s: local PHY has no auto-negotiation ability\n",
        fp->fxp_name);
}
if (!(mii_status & MII_STATUS_LS))
    printf("%s: link down\n", fp->fxp_name);
if (mii_status & MII_STATUS_JD)
    printf("%s: jabber condition detected\n", fp->fxp_name);
if (!(mii_status & MII_STATUS_EC))
{
    printf("%s: no extended register set\n", fp->fxp_name);
    goto resspeed;
}
if (!(mii_status & MII_STATUS_ANC))
    goto resspeed;

printf("%s: local cap.: ", fp->fxp_name);
if (mii_ms_ctrl & (MII_MSC_1000T_FD | MII_MSC_1000T_HD))
{
    printf("1000 Mbps: T-");
    switch(mii_ms_ctrl & (MII_MSC_1000T_FD | MII_MSC_1000T_HD))
    {
        case MII_MSC_1000T_FD: printf("FD"); break;
        case MII_MSC_1000T_HD: printf("HD"); break;
        default: printf("FD/HD"); break;
    }
    if (mii_ana)
        printf(",");
}
mii_print_techab(mii_ana);
printf("\n");

if (mii_ane & MII_ANE_PDF)
    printf("%s: parallel detection fault\n", fp->fxp_name);
if (!(mii_ane & MII_ANE_LPANA))
{
    printf("%s: link-partner does not support auto-negotiation\n",
        fp->fxp_name);
    goto resspeed;
}

printf("%s: remote cap.: ", fp->fxp_name);
if (mii_ms_ctrl & (MII_MSC_1000T_FD | MII_MSC_1000T_HD))
if (mii_ms_status & (MII_MSS_LP1000T_FD | MII_MSS_LP1000T_HD))
{
    printf("1000 Mbps: T-");
    switch(mii_ms_status &
        (MII_MSS_LP1000T_FD | MII_MSS_LP1000T_HD))
    {
        case MII_MSS_LP1000T_FD: printf("FD"); break;
        case MII_MSS_LP1000T_HD: printf("HD"); break;
        default: printf("FD/HD"); break;
    }
    if (mii_anlpa)
        printf(",");
}
mii_print_techab(mii_anlpa);
printf("\n");

if (ms_regs)
{
```

```

printf("%s: ", fp->fxp_name);
if (mii_ms_ctrl & MII_MSC_MS_MANUAL)
{
    printf("manual %s",
           (mii_ms_ctrl & MII_MSC_MS_VAL) ?
           "MASTER" : "SLAVE");
}
else
{
    printf("%s device",
           (mii_ms_ctrl & MII_MSC_MULTIPORT) ?
           "multiport" : "single-port");
}
if (mii_ms_ctrl & MII_MSC_RES)
    printf(" reserved<0x%x>", mii_ms_ctrl & MII_MSC_RES);
printf(":");
if (mii_ms_status & MII_MSS_FAULT)
    printf("M/S config fault");
else if (mii_ms_status & MII_MSS_MASTER)
    printf("MASTER");
else
    printf("SLAVE");
printf("\n");
}

if (mii_ms_status & (MII_MSS_LP1000T_FD|MII_MSS_LP1000T_HD))
{
    if (!(mii_ms_status & MII_MSS_LOCREC))
    {
        printf("%s: local receiver not OK\n",
               fp->fxp_name);
    }
    if (!(mii_ms_status & MII_MSS_REMREC))
    {
        printf("%s: remote receiver not OK\n",
               fp->fxp_name);
    }
}
if (mii_ms_status & (MII_MSS_RES|MII_MSS_IDLE_ERR))
{
    printf("%s", fp->fxp_name);
    if (mii_ms_status & MII_MSS_RES)
        printf(" reserved<0x%x>", mii_ms_status & MII_MSS_RES);
    if (mii_ms_status & MII_MSS_IDLE_ERR)
    {
        printf(" idle error %d",
               mii_ms_status & MII_MSS_IDLE_ERR);
    }
    printf("\n");
}

```

resspeed:

**#if** VERBOSE

```

    printf("%s: link up, %d Mbps, %s duplex\n",
           fp->fxp_name, (scr & MII_SCR_100) ? 100 : 10,
           (scr & MII_SCR_FD) ? "full" : "half");

```

**#endif**

;

}

```

/*=====
 *                               fxp_stop                               *
 *=====*/

```

**static void** fxp\_stop()

{

```

    int i;
    port_t port;
    fxp_t *fp;

```

```

    for (i= 0, fp= &fxp_table[0]; i<FXP_PORT_NR; i++, fp++)
    {

```

```

        if (fp->fxp_mode != FM_ENABLED)

```

```

            continue;

```

```

        if (!(fp->fxp_flags & FF_ENABLED))

```

```

        continue;
    port= fp->fxp_base_port;

    /* Reset device */
    if (debug)
        printf("%s: resetting device\n", fp->fxp_name);
    fxp_outl(port, CSR_PORT, CP_CMD_SOFT_RESET);
}
sys_exit(0);
}

/*=====
 *                               reply                               *
 *=====*/
static void reply(fp, err, may_block)
fxp_t *fp;
int err;
int may_block;
{
    message reply;
    int status;
    int r;

    status = 0;
    if (fp->fxp_flags & FF_PACK_SENT)
        status |= DL_PACK_SEND;
    if (fp->fxp_flags & FF_PACK_RECV)
        status |= DL_PACK_RECV;

    reply.m_type = DL_TASK_REPLY;
    reply.DL_PORT = fp - fxp_table;
    reply.DL_PROC = fp->fxp_client;
    reply.DL_STAT = status | ((u32_t) err << 16);
    reply.DL_COUNT = fp->fxp_read_s;

#if 0
    reply.DL_CLCK = get_uptime();
#else
    reply.DL_CLCK = 0;
#endif

    r= send(fp->fxp_client, &reply);

    if (r == ELOCKED && may_block)
    {
#if 0
        printW(); printf("send locked\n");
#endif
        return;
    }

    if (r < 0)
        panic("FXP", "fxp: send failed:", r);

    fp->fxp_read_s = 0;
    fp->fxp_flags &= ~(FF_PACK_SENT | FF_PACK_RECV);
}

/*=====
 *                               mess_reply                          *
 *=====*/
static void mess_reply(req, reply_mess)
message *req;
message *reply_mess;
{
    if (send(req->m_source, reply_mess) != OK)
        panic("FXP", "fxp: unable to mess_reply", NO_NUM);
}

/*=====
 *                               put_userdata                        *
 *=====*/
static void put_userdata(user_proc, user_addr, count, loc_addr)
int user_proc;
vir_bytes user_addr;

```

```

vir_bytes count;
void *loc_addr;
{
    int r;

    r= sys_vircopy(SELF, D, (vir_bytes)loc_addr,
        user_proc, D, user_addr, count);
    if (r != OK)
        panic("FXP", "put_userdata: sys_vircopy failed", r);
}

/*=====
 *                               eeprom_read                               *
 *=====*/
PRIVATE ul6_t eeprom_read(fp, reg)
fxp_t *fp;
int reg;
{
    port_t port;
    ul6_t v;
    int b, i, alen;

    alen= fp->fxp_ee_addrlen;
    if (!alen)
    {
        eeprom_addrsize(fp);
        alen= fp->fxp_ee_addrlen;
        assert(alen == 6 || alen == 8);
    }

    port= fp->fxp_base_port;

    fxp_outb(port, CSR_EEPROM, CE_EECS);    /* Enable EEPROM */
    v= EEPROM_READ_PREFIX;
    for (i= EEPROM_PREFIX_LEN-1; i >= 0; i--)
    {
        b= ((v & (1 << i)) ? CE_EEDI : 0);
        fxp_outb(port, CSR_EEPROM, CE_EECS | b);    /* bit */
        fxp_outb(port, CSR_EEPROM, CE_EECS | b | CE_EESK); /* Clock */
        micro_delay(EESK_PERIOD/2+1);
        fxp_outb(port, CSR_EEPROM, CE_EECS | b);
        micro_delay(EESK_PERIOD/2+1);
    }

    v= reg;
    for (i= alen-1; i >= 0; i--)
    {
        b= ((v & (1 << i)) ? CE_EEDI : 0);
        fxp_outb(port, CSR_EEPROM, CE_EECS | b);    /* bit */
        fxp_outb(port, CSR_EEPROM, CE_EECS | b | CE_EESK); /* Clock */
        micro_delay(EESK_PERIOD/2+1);
        fxp_outb(port, CSR_EEPROM, CE_EECS | b);
        micro_delay(EESK_PERIOD/2+1);
    }

    v= 0;
    for (i= 0; i<16; i++)
    {
        fxp_outb(port, CSR_EEPROM, CE_EECS | CE_EESK); /* Clock */
        micro_delay(EESK_PERIOD/2+1);
        b= !(fxp_inb(port, CSR_EEPROM) & CE_EEDO);
        v= (v << 1) | b;
        fxp_outb(port, CSR_EEPROM, CE_EECS );
        micro_delay(EESK_PERIOD/2+1);
    }
    fxp_outb(port, CSR_EEPROM, 0); /* Disable EEPROM */
    micro_delay(EECS_DELAY);

    return v;
}

/*=====
 *                               eeprom_addrsize                               *
 *=====*/

```

```

PRIVATE void eeprom_addrsize(fp)
fxp_t *fp;
{
    port_t port;
    ul6_t v;
    int b, i;

    port= fp->fxp_base_port;

    /* Try to find out the size of the EEPROM */
    fxp_outb(port, CSR_EEPROM, CE_EECS); /* Enable EEPROM */
    v= EEPROM_READ_PREFIX;
    for (i= EEPROM_PREFIX_LEN-1; i >= 0; i--)
    {
        b= ((v & (1 << i)) ? CE_EEDI : 0);
        fxp_outb(port, CSR_EEPROM, CE_EECS | b); /* bit */
        fxp_outb(port, CSR_EEPROM, CE_EECS | b | CE_EESK); /* Clock */
        micro_delay(EESK_PERIOD/2+1);
        fxp_outb(port, CSR_EEPROM, CE_EECS | b);
        micro_delay(EESK_PERIOD/2+1);
    }

    for (i= 0; i<32; i++)
    {
        b= 0;
        fxp_outb(port, CSR_EEPROM, CE_EECS | b); /* bit */
        fxp_outb(port, CSR_EEPROM, CE_EECS | b | CE_EESK); /* Clock */
        micro_delay(EESK_PERIOD/2+1);
        fxp_outb(port, CSR_EEPROM, CE_EECS | b);
        micro_delay(EESK_PERIOD/2+1);
        v= fxp_inb(port, CSR_EEPROM);
        if (!(v & CE_EEDO))
            break;
    }
    if (i >= 32)
        panic("FXP", "eeprom_addrsize: failed", NO_NUM);
    fp->fxp_ee_addrlen= i+1;

    /* Discard 16 data bits */
    for (i= 0; i<16; i++)
    {
        fxp_outb(port, CSR_EEPROM, CE_EECS | CE_EESK); /* Clock */
        micro_delay(EESK_PERIOD/2+1);
        fxp_outb(port, CSR_EEPROM, CE_EECS );
        micro_delay(EESK_PERIOD/2+1);
    }
    fxp_outb(port, CSR_EEPROM, 0); /* Disable EEPROM */
    micro_delay(EECS_DELAY);

#ifdef VERBOSE
    printf("%s EEPROM address length: %d\n",
           fp->fxp_name, fp->fxp_ee_addrlen);
#endif
}

/*=====
 *                               mii_read                               *
 *=====*/
PRIVATE ul6_t mii_read(fp, reg)
fxp_t *fp;
int reg;
{
    clock_t t0,t1;
    port_t port;
    u32_t v;

    port= fp->fxp_base_port;

    assert(!fp->fxp_mii_busy);
    fp->fxp_mii_busy++;

    if (!(fxp_inl(port, CSR_MDI_CTL) & CM_READY))
        panic("FXP", "mii_read: MDI not ready", NO_NUM);
    fxp_outl(port, CSR_MDI_CTL, CM_READ | (1 << CM_PHYADDR_SHIFT) |

```

```

        (reg << CM_REG_SHIFT));

    getuptime(&t0);
    do {
        v= fxp_inl(port, CSR_MDI_CTL);
        if (v & CM_READY)
            break;
    } while (getuptime(&t1)==OK && (t1-t0) < MICROS_TO_TICKS(100000));

    if (!(v & CM_READY))
        panic("FXP", "mii_read: MDI not ready after command", NO_NUM);

    fp->fxp_mii_busy--;
    assert(!fp->fxp_mii_busy);

    return v & CM_DATA_MASK;
}

/*=====
 *                               fxp_set_timer                               *
 *=====*/
PRIVATE void fxp_set_timer(tp, delta, watchdog)
timer_t *tp;                /* timer to be set */
clock_t delta;              /* in how many ticks */
tmr_func_t watchdog;        /* watchdog function to be called */
{
    clock_t now;             /* current time */
    int r;

    /* Get the current time. */
    r= getuptime(&now);
    if (r != OK)
        panic("FXP", "unable to get uptime from clock", r);

    /* Add the timer to the local timer queue. */
    tmrs_settimer(&fxp_timers, tp, now + delta, watchdog, NULL);

    /* Possibly reschedule an alarm call. This happens when a new timer
     * is added in front.
     */
    if (fxp_next_timeout == 0 ||
        fxp_timers->tmr_exp_time < fxp_next_timeout)
    {
        fxp_next_timeout= fxp_timers->tmr_exp_time;
#ifdef VERBOSE
        printf("fxp_set_timer: calling sys_setalarm for %d (now+%d)\n",
            fxp_next_timeout, fxp_next_timeout-now);
#endif
        r= sys_setalarm(fxp_next_timeout, 1);
        if (r != OK)
            panic("FXP", "unable to set synchronous alarm", r);
    }
}

/*=====
 *                               fxp_expire_tmrs                               *
 *=====*/
PRIVATE void fxp_expire_timers()
{
    /* A synchronous alarm message was received. Check if there are any expired
     * timers. Possibly reschedule the next alarm.
     */
    clock_t now;             /* current time */
    timer_t *tp;
    int r;

    /* Get the current time to compare the timers against. */
    r= getuptime(&now);
    if (r != OK)
        panic("FXP", "Unable to get uptime from clock.", r);

    /* Scan the timers queue for expired timers. Dispatch the watchdog function
     * for each expired timers. Possibly a new alarm call must be scheduled.
     */
}

```

```
tmr_exptimers(&fxp_timers, now, NULL);
if (fxp_timers == NULL)
    fxp_next_timeout = TMR_NEVER;
else
{
    /* set new alarm */
    fxp_next_timeout = fxp_timers->tmr_exp_time;
    r = sys_setalarm(fxp_next_timeout, 1);
    if (r != OK)
        panic("FXP", "Unable to set synchronous alarm.", r);
}
}

static void micro_delay(unsigned long usecs)
{
    tickdelay(MICROS_TO_TICKS(usecs));
}

static u8_t do_inb(port_t port)
{
    int r;
    u32_t value;

    r = sys_inb(port, &value);
    if (r != OK)
        panic("FXP", "sys_inb failed", r);
    return value;
}

static u32_t do_inl(port_t port)
{
    int r;
    u32_t value;

    r = sys_inl(port, &value);
    if (r != OK)
        panic("FXP", "sys_inl failed", r);
    return value;
}

static void do_outb(port_t port, u8_t value)
{
    int r;

    r = sys_outb(port, value);
    if (r != OK)
        panic("FXP", "sys_outb failed", r);
}

static void do_outl(port_t port, u32_t value)
{
    int r;

    r = sys_outl(port, value);
    if (r != OK)
        panic("FXP", "sys_outl failed", r);
}

/*
 * $PchId: fxp.c,v 1.4 2005/01/31 22:10:37 philip Exp $
 */
```



```
/*  
ibm/fxp.h
```

```
Registers and datastructures of the Intel 82557, 82558, 82559, 82550,  
and 82562 fast ethernet controllers.
```

```
Created:      Nov 2004 by Philip Homburg <philip@f-mnx.phicoh.com>  
*/
```

```
#define VERBOSE      0      /* display output during intialization */  
  
/* Revisions in PCI_REV */  
#define FXP_REV_82557A      0x01  
#define FXP_REV_82557B      0x02  
#define FXP_REV_82557C      0x03  
#define FXP_REV_82558A      0x04  
#define FXP_REV_82558B      0x05  
#define FXP_REV_82559A      0x06  
#define FXP_REV_82559B      0x07  
#define FXP_REV_82559C      0x08  
#define FXP_REV_82559ERA     0x09  
#define FXP_REV_82550_1     0x0C  
#define FXP_REV_82550_2     0x0D  
#define FXP_REV_82550_3     0x0E  
#define FXP_REV_82551_1     0x0F  
#define FXP_REV_82551_2     0x10  
  
/* Control/Status Registers (CSR). The first 8 bytes are called  
 * System Control Block (SCB)  
 */  
#define SCB_STATUS      0x00      /* Lower half of the SCB status word. CU and  
 * RU status.  
 */  
#define      SS_CUS_MASK      0xC0      /* CU Status */  
#define      SS_CU_IDLE      0x00      /* Idle */  
#define      SS_CU_SUSP      0x40      /* Suspended */  
#define      SS_CU_LPQA      0x80      /* LPQ Active */  
#define      SS_CU_HQPA      0xC0      /* HQP Active */  
#define      SS_RUS_MASK      0x3C      /* RU Status */  
#define      SS_RU_IDLE      0x00      /* Idle */  
#define      SS_RU_SUSP      0x04      /* Suspended */  
#define      SS_RU_NORES      0x08      /* No Resources */  
#define      SS_RU_READY      0x10      /* Ready */  
/* Other values are reserved */  
#define      SS_RESERVED      0x03      /* Reserved */  
#define SCB_INT_STAT      0x01      /* Upper half of the SCB status word.  
 * Interrupt status. Also used to acknowledge  
 * interrupts.  
 */  
#define      SIS_CX      0x80      /* CU command with interrupt bit set. On  
 * 82557 also TNO Interrupt.  
 */  
#define      SIS_FR      0x40      /* Frame Received */  
#define      SIS_CNA      0x20      /* CU Not Active */  
#define      SIS_RNR      0x10      /* RU Not Ready */  
#define      SIS_MDI      0x08      /* MDI read/write cycle completed */  
#define      SIS_SWI      0x04      /* Software Interrupt */  
#define      SIS_RES      0x02      /* Reserved */  
#define      SIS_FCP      0x01      /* Flow Control Pause Interrupt (82558 and  
 * later, reserved on 82557)  
 */  
#define SCB_CMD      0x02      /* Lower half of the SCB command word. CU and  
 * RU commands.  
 */  
#define      SC_CUC_MASK      0xF0  
#define      SC_CU_NOP      0x00      /* NOP */  
#define      SC_CU_START      0x10      /* Start CU */  
#define      SC_CU_RESUME      0x20      /* Resume CU */  
#define      SC_CU_LOAD_DCA      0x40      /* Load Dump Counters Address */  
#define      SC_CU_DUMP_SC      0x50      /* Dump Statistical Counters */  
#define      SC_CU_LOAD_BASE      0x60      /* Load CU Base */  
#define      SC_CU_DUMP_RSET_SC      0x70      /* Dump and Reset Counters */  
#define      SC_CU_STATIC_RESUME      0xA0      /* Static Resume, 82558 and  
 * above
```

```

*/
#define SC_RESERVED 0x08 /* Reserved */
#define SC_RUC_MASK 0x07 /* RU Command Mask */
#define SC_RU_NOP 0x00 /* NOP */
#define SC_RU_START 0x01 /* Start RU */
#define SC_RU_RESUME 0x02 /* Resume RU */
#define SC_RU_DMA_REDIR 0x03 /* DMA Redirect */
#define SC_RU_ABORT 0x04 /* Abort RU */
#define SC_RU_LOAD_HDR 0x05 /* Load Header Data Size */
#define SC_RU_LOAD_BASE 0x06 /* Load RU Base */
#define SCB_INT_MASK 0x03 /* Upper half of the SCB command word.
 * Interrupt mask. Can also be used to
 * generate a 'software' interrupt.
 */
/* The following 6 mask bits are not valid on
 * the 82557.
 */
#define SIM_CX 0x80 /* Mask CX */
#define SIM_FR 0x40 /* Mask FR */
#define SIM_CNA 0x20 /* Mask CNA */
#define SIM_RNR 0x10 /* Mask RNR */
#define SIM_ER 0x08 /* Mask ER */
#define SIM_FCP 0x04 /* Mask FCP */
#define SIM_SI 0x02 /* Generate Software Interrupt */
#define SIM_M 0x01 /* Mask all interrupts */
#define SCB_POINTER 0x04 /* A 32-bit (pointer) argument for CU and RU
 * commands.
 */
#define CSR_PORT 0x08 /* Control functions that bypass the SCB */
#define CP_PTR_MASK 0FFFFFFF0 /* Argument pointer */
#define CP_CMD_MASK 0x0000000F /* Commands bits */
#define CP_CMD_SOFT_RESET 0x00000000 /* Software reset */
#define CSR_PORT_RESET_DELAY 10 /* Wait for reset to
 * complete. In micro
 * seconds.
 */
#define CP_CMD_SELF_TEST 0x00000001 /* Self test */
#define CP_CMD_SEL_RESET 0x00000002 /* Selective reset */
#define CP_CMD_DUMP 0x00000003 /* Dump */
#define CP_CMD_DUMP_WAKEUP 0x00000007 /* Dump and wake-up,
 * 82559 and later.
 */
#define CSR_RESERVED 0x0C /* reserved, 16-bits */
#define CSR_EEPROM 0x0E /* EEPROM Control Register */
#define CE_RESERVED 0xF0 /* Reserved */
#define CE_EEDO 0x08 /* Serial Data Out (of the EEPROM) */
#define CE_EEDI 0x04 /* Serial Data In (to the EEPROM) */
#define CE_EECS 0x02 /* Chip Select */
#define CE_EESK 0x01 /* Serial Clock */
#define CSR_RESERVED1 0x0F /* Reserved */
#define CSR_MDI_CTL 0x10 /* MDI Control Register, 32-bits */
#define CM_RESERVED 0xC0000000 /* Reserved */
#define CM_IE 0x20000000 /* Enable Interrupt */
#define CM_READY 0x10000000 /* Command completed */
#define CM_OPCODE_MASK 0x0C000000 /* Opcode */
#define CM_WRITE 0x04000000 /* Write */
#define CM_READ 0x08000000 /* Read */
#define CM_PHYADDR_MASK 0x03E00000 /* Which PHY */
#define CM_PHYADDR_SHIFT 21
#define CM_REG_MASK 0x001F0000 /* Which register in the PHY */
#define CM_REG_SHIFT 16
#define CM_DATA_MASK 0x0000FFFF /* Data to be read or written */

/* Control Block List (CBL) commands */
#define CBL_NOP 0 /* No-operation */
#define CBL_AIS 1 /* Individual Address Setup */
#define CBL_CONF 2 /* Configure NIC */
#define CBL_MAS 3 /* Multicast Address Setup */
#define CBL_XMIT 4 /* Transmit */
#define CBL_LM 5 /* Load Microcode */
#define CBL_DUMP 6 /* Dump Internal Registers */
#define CBL_DIAG 7 /* Diagnose Command */

/* Common command fields */

```

```

#define CBL_C_CMD_MASK    0x0007 /* Command bits */
#define CBL_C_EL          0x8000 /* End of CBL */
#define CBL_C_S           0x4000 /* Suspend after the completion of the CB */
#define CBL_C_I           0x2000 /* Request CX Interrupt */
#define CBL_C_RES         0x1FF8 /* Reserved */

/* Command flags */
#define CBL_F_C           0x8000 /* Command has completed */
#define CBL_F_RES1        0x4000 /* Reserved */
#define CBL_F_OK          0x2000 /* Command was executed without errors */
#define CBL_F_RES0        0x1FFF /* Reserved */

/* Individual Address Setup (1) */
struct ias
{
    u16_t ias_status;
    u16_t ias_command;
    u32_t ias_linkaddr;
    u8_t ias_ethaddr[6];
    u8_t ias_reserved[2];
};

/* Configure (2) */
#define CC_BYTES_NR        22      /* Number of configuration bytes */
struct cbl_conf
{
    u16_t cc_status;
    u16_t cc_command;
    u32_t cc_linkaddr;
    u8_t cc_bytes[CC_BYTES_NR];
};

/* Byte 0 */
#define CCB0_RES           0xC0    /* Reserved (0) */
#define CCB0_BYTECOUNT   0x3F    /* Byte Count (typically either 8 or 22) */

/* Byte 1 */
#define CCB1_RES           0x80    /* Reserved (0) */
#define CCB1_TXFIFO_LIM    0x70    /* Transmit FIFO Limit, in DWORDS */
#define CCB1_CTL_DEFAULT   0x00    /* 0 bytes */
#define CCB1_RXFIFO_LIM    0x0F    /* Receive FIFO Limit */
#define CCB1_CRL_DEFAULT   0x08    /* 32 bytes on 82557, 64 bytes on
                                     * 82558/82559.
                                     */

/* Byte 2 */
#define CCB2_AIFS           0xFF    /* Adaptive IFS */
#define CCB2_CAI_DEFAULT   0

/* Byte 3 */
#define CCB3_RES           0xF0    /* Reserved (must be 0) on 82557 */
#define CCB3_TWCL          0x08    /* Reserved (0) */
#define CCB3_TWCL          0x08    /* Terminate Write on Cache Line */
#define CCB3_RAE           0x04    /* Read Alignment Enable */
#define CCB3_TE            0x02    /* Type Enable??? */
#define CCB3_MWIE          0x01    /* Memory Write and Invalidate (MWI) Enable
                                     * Additionally the MWI bit in the PCI
                                     * command register has to be set.
                                     * Recommended by Intel.
                                     */

/* Byte 4 */
#define CCB4_RES           0x80    /* Reserved (0) */
#define CCB4_RXDMA_MAX      0x7F    /* Receive DMA Maximum Byte Count */

/* Byte 5 */
#define CCB5_DMBCE          0x80    /* DMA Maximum Byte Count Enable */
#define CCB5_TXDMA_MAX      0x7F    /* Transmit DMA Maximum Byte Count */

/* Byte 6 */
#define CCB6_SBF           0x80    /* Save Bad Frames */
#define CCB6_DORF          0x40    /* (Do not) Discard Overrun Receive Frame,
                                     * Set this bit to keep them.
                                     */

```

```

#define CCB6_ESC          0x20      /* Extended Statistical Counter. Reserved
                                     * on 82557, must be set to 1.
                                     * Clear this bit to get more counters.
                                     */
#define CCB6_ETCB          0x10      /* Extended Transmit CB. Reserved on 82557,
                                     * must be set to 1.
                                     * Clear this bit to use Extended TxCBs.
                                     */
#define CCB6_CI_INT        0x08      /* CPU Idle (CI) Interrupt. Generate a
                                     * CI Int (bit set) or a CNA Int (bit clear)
                                     * when the CU goes to the idle state (or
                                     * to suspended for CNA).
                                     */
#define CCB6_TNO_INT       0x04      /* Enable TNO Interrupt (82557 only) */
#define CCB6_TCOSC         0x04      /* TCO Statistical Counter (82559 only) */
#define CCB6_RES           0x02      /* Reserved, must be set to 1. Called "disable
                                     * direct rcv dma mode" by the FreeBSD
                                     * driver.
                                     */
#define CCB6_LSCB          0x01      /* Late SCB Update. Only on 82557. */

/* Byte 7 */
#define CCB7_DTBD          0x80      /* Dynamic TBD. Reserved on 82557, should be
                                     * be set to 0.
                                     */
#define CCB7_2FFIFO        0x40      /* (At Most) Two Frames in FIFO. Reserved on
                                     * 82557, should be set to 0.
                                     */
#define CCB7_RES           0x38      /* Reserved (0) */
#define CCB7_UR            0x06      /* Underrun Retry */
#define CUR_0              0x00      /* No re-transmission */
#define CUR_1              0x02      /* One re-transmission */
#define CUR_2              0x04      /* Two re-transmissions, 1st retry with
                                     * 512 bytes.
                                     */
#define CUR_3              0x06      /* Tree re-transmissions, 1st retry
                                     * with 512 bytes, 2nd retry with 1024.
                                     */
#define CCB7_DSRF          0x01      /* Discard Short Receive Frames. */

/* Byte 8 */
#define CCB8_CSMAD         0x80      /* CSMA Disable. Reserved on 82557, should be
                                     * set to zero.
                                     */
#define CCB8_RES           0x7E      /* Reserved (0) */
#define CCB8_503_MII       0x01      /* 503 mode or MII mode. Reserved on 82558
                                     * and 82559, should be set to 1.
                                     */

/* Byte 9 */
#define CCB9_MMWE          0x80      /* Multicast Match Wake Enable. 82558 B-step
                                     * only, should be set to zero on other
                                     * devices.
                                     */
#define CCB9_AWE           0x40      /* ARP Wake-up Enable. 82558 B-step only,
                                     * should be set to zero on other devices.
                                     */
#define CCB9_LSCWE         0x20      /* Link Status Change Wake Enable. Available
                                     * on 82558 B-step and 82559. Should be
                                     * set to zero on 82557 and 82558 A-step
                                     */
#define CCB9_VARP          0x10      /* VLAN ARP (82558 B-step) or VLAN TCO (82559).
                                     * Should be zero on 82557 and 82558 A-step
                                     */
#define CCB9_RES           0x0E      /* Reserved (0) */
#define CCB9_TUC           0x01      /* TCP/UDP Checksum. 82559 only, should be
                                     * zero on other devices.
                                     */

/* Byte 10 */
#define CCB10_LOOPBACK     0xC0      /* Loopback mode */
#define CLB_NORMAL         0x00      /* Normal operation */
#define CLB_INTERNAL       0x40      /* Internal loopback */
#define CLB_RESERVED       0x80      /* Reserved */

```

```

#define CLB_EXTERNAL 0xC0 /* External loopback */
#define CCB10_PAL 0x30 /* Pre-amble length */
#define CPAL_1 0x00 /* 1 byte */
#define CPAL_3 0x10 /* 3 bytes */
#define CPAL_7 0x20 /* 7 bytes */
#define CPAL_15 0x30 /* 15 bytes */
#define CPAL_DEFAULT CPAL_7
#define CCB10_NSAI 0x08 /* No Source Address Insertion */
#define CCB10_RES1 0x06 /* Reserved, should be set to 1 */
#define CCB10_RES0 0x01 /* Reserved (0) */

/* Byte 11 */
#define CCB11_RES 0xF8 /* Reserved (0) */
#define CCB11_LINPRIO 0x07 /* Linear Priority. 82557 only,
    * should be zero on other devices.
    */

/* Byte 12 */
#define CCB12_IS 0xF0 /* Interframe spacing in multiples of
    * 16 bit times.
    */
#define CIS_DEFAULT 0x60 /* 96 (6 in register) */
#define CCB12_RES 0x0E /* Reserved (0) */
#define CCB12_LPM 0x01 /* Linear Priority Mode. 82557 only,
    * should be zero on other devices.
    */

/* Byte 13, 4th byte of IP address for ARP frame filtering. Only valid on
    * 82558 B-step. Should be 0 on other devices.
    */
#define CCB13_DEFAULT 0x00
/* Byte 14, 3rd byte of IP address for ARP frame filtering. Only valid on
    * 82558 B-step. Should be 0xF2 on other devices.
    */
#define CCB14_DEFAULT 0xF2

/* Byte 15 */
#define CCB15_CRSCDT 0x80 /* CRS or CDT. */
#define CCB15_RES1 0x40 /* Reserved, should be set to one. */
#define CCB15_CRC16 0x20 /* 16-bit CRC. Only on 82559,
    * should be zero on other devices
    */
#define CCB15_IUL 0x10 /* Ignore U/L. Reserved on 82557 and
    * should be set to zero.
    */
#define CCB15_RES2 0x08 /* Reserved, should be set to one. */
#define CCB15_WAW 0x04 /* Wait After Win. Reserved on 82557,
    * should be set to zero.
    */
#define CCB15_BD 0x02 /* Broadcast disable */
#define CCB15_PM 0x01 /* Promiscuous mode */

/* Byte 16. FC Delay Least Significant Byte. Reserved on the 82557 and
    * should be set to zero.
    */
#define CCB16_DEFAULT 0x00

/* Byte 17. FC Delay Most Significant Byte. This byte is reserved on the
    * 82557 and should be set to 0x40.
    */
#define CCB17_DEFAULT 0x40

/* Byte 18 */
#define CCB18_RES1 0x80 /* Reserved, should be set to 1 */
#define CCB18_PFCT 0x70 /* Priority Flow Control Threshold.
    * Reserved on the 82557 and should
    * be set to 1. All bits 1 (disabled)
    * is the recommended default.
    */
#define CCB18_LROK 0x08 /* Long Receive OK. Reserved on the
    * 82557 and should be set to zero.
    * Required for VLANs.
    */
#define CCB18_RCRCT 0x04 /* Receive CRC Transfer */

```

```

#define CCB18_PE          0x02    /* Padding Enable */
#define CCB18_SE          0x01    /* Stripping Enable */

/* Byte 19 */
#define CCB19_FDPE        0x80    /* Full Duplex Pin Enable */
#define CCB19_FFD         0x40    /* Force Full Duplex */
#define CCB19_RFC         0x20    /* Reject FC. Reserved on the 82557
    * and should be set to zero.
    */
#define CCB19_FDRSTAFCD   0x10    /* Full Duplex Restart Flow Control.
    * Reserved on the 82557 and should be
    * set to zero.
    */
#define CCB19_FDRSTOFC    0x08    /* Full Duplex Restop Flow Control.
    * Reserved on the 82557 and should be
    * set to zero.
    */
#define CCB19_FDTFCD      0x04    /* Full Duplex Transmit Flow Control
    * Disable. Reserved on the 82557 and
    * should be set to zero.
    */
#define CCB19_MPWD        0x02    /* Magic Packet Wake-up Disable.
    * Reserved on the 82557 and 82559ER
    * and should be set to zero.
    */
#define CCB19_AW          0x01    /* Address Wake-up (82558 A-step) and
    * IA Match Wake Enable (82558 B-step)
    * Reserved on the 82557 and 82559 and
    * should be set to zero.
    */

/* Byte 20 */
#define CCB20_RES         0x80    /* Reserved (0) */
#define CCB20_MIA         0x40    /* Multiple IA */
#define CCB20_PFCL        0x20    /* Priority FC Location. Reserved on
    * the 82557 and should be set to 1.
    */
#define CCB20_RES1        0x1F    /* Reserved, should be set to 1 */

/* Byte 21 */
#define CCB21_RES         0xF0    /* Reserved (0) */
#define CCB21_MA          0x08    /* Multicast All */
#define CCB21_RES1_MASK   0x07    /* Reserved, should be set to 5 */
#define CCB21_RES21       0x05

/* Transmit (4) */
struct tx
{
    u16_t tx_status;
    u16_t tx_command;
    u32_t tx_linkaddr;
    u32_t tx_tbd;
    u16_t tx_size;
    u8_t tx_tthresh;
    u8_t tx_ntbd;
    u8_t tx_buf[ETH_MAX_PACK_SIZE_TAGGED];
};

#define TXS_C              0x8000 /* Transmit DMA has completed */
#define TXS_RES            0x4000 /* Reserved */
#define TXS_OK             0x2000 /* Command was executed without error */
#define TXS_U              0x1000 /* This or previous frame encountered underrun */
#define TXS_RES1           0x0FFF /* Reserved (0) */

#define TXC_EL             0x8000 /* End of List */
#define TXC_S              0x4000 /* Suspend after this CB */
#define TXC_I              0x2000 /* Interrupt after this CB */
#define TXC_CID_MASK       0x1F00 /* CNA Interrupt Delay */
#define TXC_RES            0x00E0 /* Reserved (0) */
#define TXC_NC             0x0010 /* No CRC and Source Address Insertion */
#define TXC_SF             0x0008 /* Not in Simplified Mode */
#define TXC_CMD            0x0007 /* Command */

#define TXSZ_EOF           0x8000 /* End of Frame */

```

```
#define TXSZ_RES      0x4000 /* Reserved (0) */
#define TXSZ_COUNT    0x3FFF /* Transmit Byte Count */

#define TX_TBDA_NIL    0xFFFFFFFF /* Null Pointer for TBD Array */

#define TXTT_MIN       0x01 /* Minimum for Transmit Threshold */
#define TXTT_MAX       0xE0 /* Maximum for Transmit Threshold */

/* Statistical Counters */
struct sc
{
    u32_t sc_tx_good; /* Transmit Good Frames */
    u32_t sc_tx_maxcol; /* Transmit Maximum Collisions errors */
    u32_t sc_tx_latecol; /* Transmit Late Collisions errors */
    u32_t sc_tx_underrun; /* Transmit Underrun errors */
    u32_t sc_tx_crs; /* Transmit Lost Carrier Sense */
    u32_t sc_tx_defered; /* Transmit Defered */
    u32_t sc_tx_scol; /* Transmit Single Collision */
    u32_t sc_tx_mcol; /* Transmit Multiple Collisions */
    u32_t sc_tx_totcol; /* Transmit Total Collisions */
    u32_t sc_rx_good; /* Receive Good Frames */
    u32_t sc_rx_crc; /* Receive CRC errors */
    u32_t sc_rx_align; /* Receive Alignment errors */
    u32_t sc_rx_resource; /* Receive Resource errors */
    u32_t sc_rx_overrun; /* Receive Overrun errors */
    u32_t sc_rx_cd; /* Receive Collision Detect errors */
    u32_t sc_rx_short; /* Receive Short Frame errors */

    /* Short form ends here. The magic number will
     * be stored in the next field.
     */

    u32_t sc_tx_fcp; /* Transmit Flow Control Pause */
    u32_t sc_rx_fcp; /* Receive Flow Control Pause */
    u32_t sc_rx_fcu; /* Receive Flow Control Unsupported */

    /* Longer form (82558 and later) ends here.
     * The magic number will be stored in the
     * next field.
     */

    u32_t sc_tx_tco; /* Transmit TCO frames */
    u32_t sc_rx_tco; /* Receive TCO frames */
    u32_t sc_magic; /* Dump of counters completed */
};

#define SCM_DSC      0x0000A005 /* Magic for SC_CU_DUMP_SC command */
#define SCM_DRSC     0x0000A007 /* Magic for SC_CU_DUMP_RSET_SC cmd */

/* Receive Frame Descriptor (RFD) */
struct rfd
{
    u16_t rfd_status;
    u16_t rfd_command;
    u32_t rfd_linkaddr;
    u32_t rfd_reserved;
    u16_t rfd_res;
    u16_t rfd_size;
    u8_t rfd_buf[ETH_MAX_PACK_SIZE_TAGGED];
};

#define RFDS_C      0x8000 /* Frame Reception Completed */
#define RFDS_RES     0x4000 /* Reserved (0) */
#define RFDS_OK      0x2000 /* Frame received without any errors */
#define RFDS_RES1     0x1000 /* Reserved */
#define RFDS_CRCERR   0x0800 /* CRC error */
#define RFDS_ALIGNERR 0x0400 /* Alignment error */
#define RFDS_OUTOFBUF 0x0200 /* Ran out of buffer space (frame is frager
     * than supplied buffer).
     */
#define RFDS_DMAOVR   0x0100 /* DMA overrun failure */
#define RFDS_TOOSHORT 0x0080 /* Frame Too Short */
#define RFDS_RES2     0x0040 /* Reserved */
#define RFDS_TYPED    0x0020 /* Frame Is Typed (Type/Length field is 0 or
```

```

        * >1500)
        */
#define RFDS_RXERR      0x0010 /* Receive Error */
#define RFDS_RES3       0x0008 /* Reserved */
#define RFDS_NOAM       0x0004 /* No Address Match */
#define RFDS_NOAIAM     0x0002 /* No IA Address Match */
#define RFDS_RXCOL      0x0001 /* Collision Detected During Reception (82557
        * and 82558 only)
        */
#define RFDS_TCO        0x0001 /* TCO Packet (82559 and later) */

#define RFDC_EL         0x8000 /* End of List */
#define RFDC_S          0x4000 /* Suspend */
#define RFDC_RES        0x3FE0 /* Reserved (0) */
#define RFDC_H          0x0010 /* Header RFD */
#define RFDC_SF         0x0008 /* (Not) Simplified Mode */
#define RFDC_RES1       0x0007 /* Reserved (0) */

#define RFDR_EOF        0x8000 /* End of Frame (all data is in the buffer) */
#define RFDR_F          0x4000 /* Finished updating the count field */
#define RFDR_COUNT      0x3FFF /* Actual Count */

#define RFDSZ_RES       0xC000 /* Reserved (0) */
#define RFDSZ_SIZE      0x3FFF /* Buffer Size */

/* EEPROM commands */
#define EEPROM_READ_PREFIX 0x6 /* Read command */
#define EEPROM_PREFIX_LEN 3 /* Start bit and two command bits */

/* EEPROM timing parameters */
#define EECS_DELAY      1 /* Keep EECS low for at least EECS_DELAY
        * microseconds
        */
#define EESK_PERIOD     4 /* A cycle of driving EESK high followed by
        * driving EESK low should take at least
        * EESK_PERIOD microseconds
        */

/* Special registers in the 82555 (and compatible) PHYs. Should be moved
 * to a separate file if other drivers need this too.
 */
#define MII_SCR          0x10 /* Status and Control Register */
#define MII_SCR_FC       0x8000 /* Flow Control */
#define MII_SCR_T4E      0x4000 /* Enable T4 unless auto-negotiation */
#define MII_SCR_CRSDC    0x2000 /* RX100 CRS Disconnect */
#define MII_SCR_RES      0x1000 /* Reserved */
#define MII_SCR_RCVSYNC  0x0800 /* RCV De-Serializer in sync */
#define MII_SCR_100DOWN  0x0400 /* 100Base-T Power Down */
#define MII_SCR_10DOWN   0x0200 /* 10Base-T Power Down */
#define MII_SCR_POLARITY 0x0100 /* 10Base-T Polarity */
#define MII_SCR_RES_1    0x00F8 /* Reserved */
#define MII_SCR_T4       0x0004 /* 100Base-T4 negotiated */
#define MII_SCR_100      0x0002 /* 100 Mbps negotiated */
#define MII_SCR_FD       0x0001 /* Full Duplex negotiated */

/*
 * $PchId: fxp.h,v 1.1 2004/11/23 14:34:03 philip Exp $
 */

```



```

/*
ibm/mii.c

Created:      Nov 2004 by Philip Homburg <philip@f-mnx.phicoh.com>

Media Independent (Ethernet) Interface functions
*/

#include "../drivers.h"
#include __minix_vmd
#include "config.h"
#endif

#include "mii.h"

/*=====
*                               mii_print_stat_speed                               *
*=====*/
PUBLIC void mii_print_stat_speed(stat, extstat)
ul6_t stat;
ul6_t extstat;
{
    int fs, ft;

    fs= 1;
    if (stat & MII_STATUS_EXT_STAT)
    {
        if (extstat & (MII_ESTAT_1000XFD | MII_ESTAT_1000XHD |
            MII_ESTAT_1000TFD | MII_ESTAT_1000THD))
        {
            printf("1000 Mbps: ");
            fs= 0;
            ft= 1;
            if (extstat & (MII_ESTAT_1000XFD | MII_ESTAT_1000XHD))
            {
                ft= 0;
                printf("X-");
                switch(extstat &
                    (MII_ESTAT_1000XFD|MII_ESTAT_1000XHD))
                {
                    case MII_ESTAT_1000XFD: printf("FD"); break;
                    case MII_ESTAT_1000XHD: printf("HD"); break;
                    default:                printf("FD/HD"); break;
                }
            }
            if (extstat & (MII_ESTAT_1000TFD | MII_ESTAT_1000THD))
            {
                if (!ft)
                    printf(",");
                ft= 0;
                printf("T-");
                switch(extstat &
                    (MII_ESTAT_1000TFD|MII_ESTAT_1000THD))
                {
                    case MII_ESTAT_1000TFD: printf("FD"); break;
                    case MII_ESTAT_1000THD: printf("HD"); break;
                    default:                printf("FD/HD"); break;
                }
            }
        }
    }
    if (stat & (MII_STATUS_100T4 |
        MII_STATUS_100XFD | MII_STATUS_100XHD |
        MII_STATUS_100T2FD | MII_STATUS_100T2HD))
    {
        if (!fs)
            printf(",");
        fs= 0;
        printf("100 Mbps: ");
        ft= 1;
        if (stat & MII_STATUS_100T4)
        {
            printf("T4");
            ft= 0;
        }
    }
}

```

```

    }
    if (stat & (MII_STATUS_100XFD | MII_STATUS_100XHD))
    {
        if (!ft)
            printf(",");
        ft= 0;
        printf("TX-");
        switch(stat & (MII_STATUS_100XFD|MII_STATUS_100XHD))
        {
            case MII_STATUS_100XFD: printf("FD"); break;
            case MII_STATUS_100XHD: printf("HD"); break;
            default:                 printf("FD/HD"); break;
        }
    }
    if (stat & (MII_STATUS_100T2FD | MII_STATUS_100T2HD))
    {
        if (!ft)
            printf(",");
        ft= 0;
        printf("T2-");
        switch(stat & (MII_STATUS_100T2FD|MII_STATUS_100T2HD))
        {
            case MII_STATUS_100T2FD:     printf("FD"); break;
            case MII_STATUS_100T2HD:     printf("HD"); break;
            default:                     printf("FD/HD"); break;
        }
    }
}
if (stat & (MII_STATUS_10FD | MII_STATUS_10HD))
{
    if (!fs)
        printf(",");
    printf("10 Mbps: ");
    fs= 0;
    printf("T-");
    switch(stat & (MII_STATUS_10FD|MII_STATUS_10HD))
    {
        case MII_STATUS_10FD:     printf("FD"); break;
        case MII_STATUS_10HD:     printf("HD"); break;
        default:                 printf("FD/HD"); break;
    }
}
}

/*=====
 *                               mii_print_techab                               *
 *=====*/
PUBLIC void mii_print_techab(techab)
ul6_t techab;
{
    int fs, ft;

    if ((techab & MII_ANA_SEL_M) != MII_ANA_SEL_802_3)
    {
        printf("strange selector 0x%x, value 0x%x",
            techab & MII_ANA_SEL_M,
            (techab & MII_ANA_TAF_M) >> MII_ANA_TAF_S);

        return;
    }
    fs= 1;
    if (techab & (MII_ANA_100T4 | MII_ANA_100TXFD | MII_ANA_100TXHD))
    {
        printf("100 Mbps: ");
        fs= 0;
        ft= 1;
        if (techab & MII_ANA_100T4)
        {
            printf("T4");
            ft= 0;
        }
        if (techab & (MII_ANA_100TXFD | MII_ANA_100TXHD))
        {
            if (!ft)
                printf(",");

```

```
        ft= 0;
        printf("TX-");
        switch(ttechab & (MII_ANA_100TXFD|MII_ANA_100TXHD))
        {
            case MII_ANA_100TXFD:    printf("FD"); break;
            case MII_ANA_100TXHD:    printf("HD"); break;
            default:                  printf("FD/HD"); break;
        }
    }
}
if (techab & (MII_ANA_10TFD | MII_ANA_10THD))
{
    if (!fs)
        printf(",");
    printf("10 Mbps: ");
    fs= 0;
    printf("T-");
    switch(ttechab & (MII_ANA_10TFD|MII_ANA_10THD))
    {
        case MII_ANA_10TFD:        printf("FD"); break;
        case MII_ANA_10THD:        printf("HD"); break;
        default:                    printf("FD/HD"); break;
    }
}
if (techab & MII_ANA_PAUSE_SYM)
{
    if (!fs)
        printf(",");
    fs= 0;
    printf("pause(SYM)");
}
if (techab & MII_ANA_PAUSE_ASYM)
{
    if (!fs)
        printf(",");
    fs= 0;
    printf("pause(ASYM)");
}
if (techab & MII_ANA_TAF_RES)
{
    if (!fs)
        printf(",");
    fs= 0;
    printf("0x%x", (techab & MII_ANA_TAF_RES) >> MII_ANA_TAF_S);
}
}

/*
 * $PchId: mii.c,v 1.2 2005/01/31 22:17:26 philip Exp $
 */
```

```
/*
ibm/mii.h
```

```
Created:      Nov 2004 by Philip Homburg <philip@f-mnx.phicoh.com>
```

```
Definitions for the Media Independent (Ethernet) Interface
*/
```

```
/* Registers in the Machine Independent Interface (MII) to the PHY.
```

```
 * IEEE 802.3 (2000 Edition) Clause 22.
```

```
 */
```

```
#define MII_CTRL      0x0      /* Control Register (basic) */
#define MII_CTRL_RST  0x8000 /* Reset PHY */
#define MII_CTRL_LB   0x4000 /* Enable Loopback Mode */
#define MII_CTRL_SP_LSB 0x2000 /* Speed Selection (LSB) */
#define MII_CTRL_ANE  0x1000 /* Auto Negotiation Enable */
#define MII_CTRL_PD    0x0800 /* Power Down */
#define MII_CTRL_ISO   0x0400 /* Isolate */
#define MII_CTRL_RAN   0x0200 /* Restart Auto-Negotiation Process */
#define MII_CTRL_DM    0x0100 /* Full Duplex */
#define MII_CTRL_CT    0x0080 /* Enable COL Signal Test */
#define MII_CTRL_SP_MSB 0x0040 /* Speed Selection (MSB) */
#define MII_CTRL_SP_10      0x0000 /* 10 Mb/s */
#define MII_CTRL_SP_100    0x2000 /* 100 Mb/s */
#define MII_CTRL_SP_1000   0x0040 /* 1000 Mb/s */
#define MII_CTRL_SP_RES    0x2040 /* Reserved */
#define MII_CTRL_RES      0x003F /* Reserved */
#define MII_STATUS      0x1      /* Status Register (basic) */
#define MII_STATUS_100T4 0x8000 /* 100Base-T4 support */
#define MII_STATUS_100XFD 0x4000 /* 100Base-X FD support */
#define MII_STATUS_100XHD 0x2000 /* 100Base-X HD support */
#define MII_STATUS_10FD   0x1000 /* 10 Mb/s FD support */
#define MII_STATUS_10HD   0x0800 /* 10 Mb/s HD support */
#define MII_STATUS_100T2FD 0x0400 /* 100Base-T2 FD support */
#define MII_STATUS_100T2HD 0x0200 /* 100Base-T2 HD support */
#define MII_STATUS_EXT_STAT 0x0100 /* Supports MII_EXT_STATUS */
#define MII_STATUS_RES    0x0080 /* Reserved */
#define MII_STATUS_MFPS   0x0040 /* MF Preamble Suppression */
#define MII_STATUS_ANC    0x0020 /* Auto-Negotiation Completed */
#define MII_STATUS_RF     0x0010 /* Remote Fault Detected */
#define MII_STATUS_ANA    0x0008 /* Auto-Negotiation Ability */
#define MII_STATUS_LS     0x0004 /* Link Up */
#define MII_STATUS_JD     0x0002 /* Jabber Condition Detected */
#define MII_STATUS_EC     0x0001 /* Ext Register Capabilities */
#define MII_PHYID_H      0x2      /* PHY ID (high) */
#define MII_PH_OUI_H_MASK 0xFFFF /* High part of OUI */
#define MII_PH_OUI_H_C_SHIFT 6 /* Shift up in OUI */
#define MII_PHYID_L      0x3      /* PHY ID (low) */
#define MII_PL_OUI_L_MASK 0xFC00 /* Low part of OUI */
#define MII_PL_OUI_L_SHIFT 10
#define MII_PL_MODEL_MASK 0x03F0 /* Model */
#define MII_PL_MODEL_SHIFT 4
#define MII_PL_REV_MASK   0x000F /* Revision */
#define MII_ANA          0x4      /* Auto-Negotiation Advertisement */
#define MII_ANA_NP        0x8000 /* Next Page */
#define MII_ANA_RES       0x4000 /* Reserved */
#define MII_ANA_RF        0x2000 /* Remote Fault */
#define MII_ANA_TAF_M     0x1FE0 /* Technology Ability Field */
#define MII_ANA_TAF_S     5 /* Shift */
#define MII_ANA_TAF_RES   0x1000 /* Reserved */
#define MII_ANA_PAUSE_ASYM 0x0800 /* Asym. Pause */
#define MII_ANA_PAUSE_SYM  0x0400 /* Sym. Pause */
#define MII_ANA_100T4      0x0200 /* 100Base-T4 */
#define MII_ANA_100TXFD    0x0100 /* 100Base-TX FD */
#define MII_ANA_100TXHD    0x0080 /* 100Base-TX HD */
#define MII_ANA_10TFD     0x0040 /* 10Base-T FD */
#define MII_ANA_10THD     0x0020 /* 10Base-T HD */
#define MII_ANA_SEL_M     0x001F /* Selector Field */
#define MII_ANA_SEL_802_3 0x0001 /* 802.3 */
#define MII_ANLPA        0x5      /* Auto-Neg Link Partner Ability Register */
#define MII_ANLPA_NP      0x8000 /* Next Page */
#define MII_ANLPA_ACK     0x4000 /* Acknowledge */
#define MII_ANLPA_RF      0x2000 /* Remote Fault */
#define MII_ANLPA_TAF_M   0x1FC0 /* Technology Ability Field */
```

```

#define MII_ANLPA_SEL_M 0x001F /* Selector Field */
#define MII_ANE 0x6 /* Auto-Negotiation Expansion */
#define MII_ANE_RES 0xFFE0 /* Reserved */
#define MII_ANE_PDF 0x0010 /* Parallel Detection Fault */
#define MII_ANE_LPNPA 0x0008 /* Link Partner is Next Page Able */
#define MII_ANE_NPA 0x0002 /* Local Device is Next Page Able */
#define MII_ANE_PR 0x0002 /* New Page has been received */
#define MII_ANE_LPANA 0x0001 /* Link Partner is Auto-Neg.able */
#define MII_ANNPT 0x7 /* Auto-Negotiation Next Page Transmit */
#define MII_ANLPRNP 0x8 /* Auto-Neg Link Partner Received Next Page */
#define MII_MS_CTRL 0x9 /* MASTER-SLAVE Control Register */
#define MII_MSC_TEST_MODE 0xE000 /* Test mode */
#define MII_MSC_MS_MANUAL 0x1000 /* Master/slave manual config */
#define MII_MSC_MS_VAL 0x0800 /* Master/slave value */
#define MII_MSC_MULTIPORT 0x0400 /* Multi-port device */
#define MII_MSC_1000T_FD 0x0200 /* 1000Base-T Full Duplex */
#define MII_MSC_1000T_HD 0x0100 /* 1000Base-T Half Duplex */
#define MII_MSC_RES 0x00FF /* Reserved */
#define MII_MS_STATUS 0xA /* MASTER-SLAVE Status Register */
#define MII_MSS_FAULT 0x8000 /* Master/slave config fault */
#define MII_MSS_MASTER 0x4000 /* Master */
#define MII_MSS_LOCREC 0x2000 /* Local Receiver OK */
#define MII_MSS_REMREC 0x1000 /* Remote Receiver OK */
#define MII_MSS_LP1000T_FD 0x0800 /* Link Partner 1000-T FD */
#define MII_MSS_LP1000T_HD 0x0400 /* Link Partner 1000-T HD */
#define MII_MSS_RES 0x0300 /* Reserved */
#define MII_MSS_IDLE_ERR 0x00FF /* Idle Error Counter */
/* 0xB ... 0xE */ /* Reserved */
#define MII_EXT_STATUS 0xF /* Extended Status */
#define MII_ESTAT_1000XFD 0x8000 /* 1000Base-X Full Duplex */
#define MII_ESTAT_1000XHD 0x4000 /* 1000Base-X Half Duplex */
#define MII_ESTAT_1000TFD 0x2000 /* 1000Base-T Full Duplex */
#define MII_ESTAT_1000THD 0x1000 /* 1000Base-T Half Duplex */
#define MII_ESTAT_RES 0x0FFF /* Reserved */
/* 0x10 ... 0x1F */ /* Vendor Specific */

_PROTOTYPE( void mii_print_stat_speed, (U16_t stat, U16_t extstat) );
_PROTOTYPE( void mii_print_techab, (U16_t techab) );

/*
 * $PchId: mii.h,v 1.1 2004/12/27 13:33:30 philip Exp $
 */

```

```
# Makefile for Intel Pro/100 driver (FXP)
DRIVER = lance

# directories
u = /usr
i = $u/include
s = $i/sys
m = $i/minix
b = $i/ibm
d = ..

# programs, flags, etc.
CC =      exec cc
CFLAGS = -I$i
LDFLAGS = -i
LIBS = -lsys -lsysutil
#-lutils -ltimers

OBJ = lance.o

# build local binary
all build:      $(DRIVER)
$(DRIVER):      $(OBJ)
                $(CC) -o $@ $(LDFLAGS) $(OBJ) $(LIBS)
                install -S 4k $(DRIVER)

# install with other drivers
install:      /usr/sbin/$(DRIVER)
/usr/sbin/$(DRIVER):      $(DRIVER)
                install -o root -cs $? $@

# clean up local files
clean:
                rm -f *.o *.bak $(DRIVER)

depend:
                /usr/bin/mkdep "$(CC) -E $(CPPFLAGS)" *.c > .depend

# Include generated dependencies.
include .depend
```

```

/*
 * lance.c
 *
 * This file contains a ethernet device driver for AMD LANCE based ethernet
 * cards.
 *
 * The valid messages and their parameters are:
 *
 *      m_type      DL_PORT      DL_PROC      DL_COUNT      DL_MODE      DL_ADDR
 *      /-----+-----+-----+-----+-----+-----/
 *      / HARDINT   /             /             /             /             /
 *      /-----+-----+-----+-----+-----+-----/
 *      / DL_WRITE  / port nr    / proc nr    / count      / mode       / address
 *      /-----+-----+-----+-----+-----+-----/
 *      / DL_WRITEV / port nr    / proc nr    / count      / mode       / address
 *      /-----+-----+-----+-----+-----+-----/
 *      / DL_READ   / port nr    / proc nr    / count      /             / address
 *      /-----+-----+-----+-----+-----+-----/
 *      / DL_READV  / port nr    / proc nr    / count      /             / address
 *      /-----+-----+-----+-----+-----+-----/
 *      / DL_INIT   / port nr    / proc nr    / mode       /             / address
 *      /-----+-----+-----+-----+-----+-----/
 *      / DL_GETSTAT / port nr    / proc nr    /             /             / address
 *      /-----+-----+-----+-----+-----+-----/
 *      / DL_STOP   / port_nr    /             /             /             /
 *      /-----+-----+-----+-----+-----+-----/
 *
 * The messages sent are:
 *
 *      m-type      DL_POR T      DL_PROC      DL_COUNT      DL_STAT      DL_CLK
 *      /-----+-----+-----+-----+-----+-----/
 *      / DL_TASK_REPL / port nr    / proc nr    / rd-count   / err/stat   / clock
 *      /-----+-----+-----+-----+-----+-----/
 *
 *      m_type      m3_i1      m3_i2      m3_ca1
 *      /-----+-----+-----+-----+-----/
 *      / DL_INIT_REPL / port nr    / last port  / ethernet   /
 *      /-----+-----+-----+-----+-----/
 *
 * Created: Jul 27, 2002 by Kazuya Kodama <kazuya@nii.ac.jp>
 * Adapted for Minix 3: Sep 05, 2005 by Joren l'Ami <jwlami@cs.vu.nl>
 */

```

```
#define VERBOSE 0
```

```
#include "../drivers.h"
```

```
#include <minix/keymap.h>
```

```
#include <net/hton.h>
```

```
#include <net/gen/ether.h>
```

```
#include <net/gen/eth_io.h>
```

```
#include <assert.h>
```

```
#include <minix/syslib.h>
```

```
#include <ibm/pci.h>
```

```
#include "lance.h"
```

```
/*#include "proc.h"*/
```

```
#include <sys/ioc_memory.h>
```

```
/* new I/O functions in Minix 3 */
```

```
#define out_byte( x, y ) sys_outb( x, y )
```

```
#define out_word( x, y ) sys_outw( x, y )
```

```
static U8_t in_byte(U16_t port)
```

```

{
    unsigned long value;
    int s;
    if ((s=sys_inb(port, &value)) != OK)
        printf( "lance: warning, sys_inb failed: %d\n", s );
    return (U8_t) value;
}

```

```

static U16_t in_word( U16_t port)
{
    unsigned long value;
    int s;
    if ((s=sys_inw(port, &value)) != OK)
        printf( "lance: warning, sys_inw failed: %d\n", s );
    return (U16_t) value;
}
/*
#define in_byte( x ) inb( x )
#define in_word( x ) inw( x )
*/

static ether_card_t ec_table[EC_PORT_NR_MAX];
static int eth_tasknr= ANY;
static ul6_t eth_ign_proto;

/* Configuration */
typedef struct ec_conf
{
    port_t ec_port;
    int ec_irq;
    phys_bytes ec_mem;
    char *ec_envvar;
} ec_conf_t;

/* We hardly use these. Just "LANCE0=on/off" "LANCE1=on/off" mean. */
ec_conf_t ec_conf[]= /* Card addresses */
{
    /* I/O port, IRQ, Buffer address, Env. var, Buf selector. */
    { 0x1000, 9, 0x00000, "LANCE0" },
    { 0xD000, 15, 0x00000, "LANCE1" },
};

/* Actually, we use PCI-BIOS info. */
PRIVATE struct pcitab
{
    ul6_t vid;
    ul6_t did;
    int checkclass;
} pcitab[]=
{
    { PCI_VENDOR_ID_AMD, PCI_DEVICE_ID_AMD_LANCE, 0 }, /* AMD LANCE */
    { 0x0000, 0x0000, 0 }
};
/*
struct pci_device pci_dev_list[] = {
    { PCI_VENDOR_ID_AMD, PCI_DEVICE_ID_AMD_LANCE,
      "AMD Lance/PCI", 0, 0, 0, 0, 0, 0 },
    { PCI_VENDOR_ID_AMD, PCI_DEVICE_ID_AMD_LANCE,
      "AMD Lance/PCI", 0, 0, 0, 0, 0, 0 },
    { 0, 0, NULL, 0, 0, 0, 0, 0, 0 }
};
*/

/* General */
_PROTOTYPE( static void do_init, (message *mp) );
_PROTOTYPE( static void ec_init, (ether_card_t *ec) );
_PROTOTYPE( static void ec_confaddr, (ether_card_t *ec) );
_PROTOTYPE( static void ec_reinit, (ether_card_t *ec) );
_PROTOTYPE( static void ec_check_ints, (ether_card_t *ec) );
_PROTOTYPE( static void conf_hw, (ether_card_t *ec) );
/* _PROTOTYPE( static int ec_handler, (irq_hook_t *hook) ); */
_PROTOTYPE( static void update_conf, (ether_card_t *ec, ec_conf_t *ecp) );
_PROTOTYPE( static void mess_reply, (message *req, message *reply) );
_PROTOTYPE( static void do_int, (ether_card_t *ec) );
_PROTOTYPE( static void reply,
    (ether_card_t *ec, int err, int may_block) );
_PROTOTYPE( static void ec_reset, (ether_card_t *ec) );
_PROTOTYPE( static void ec_send, (ether_card_t *ec) );
_PROTOTYPE( static void ec_recv, (ether_card_t *ec) );
_PROTOTYPE( static void do_vwrite,
    (message *mp, int from_int, int vectored) );

```



```

_PROTOTYPE( static void do_vread, (message *mp, int vectored) ) ;
_PROTOTYPE( static void get_userdata,
    (int user_proc, vir_bytes user_addr,
    vir_bytes count, void *loc_addr) ) ;
_PROTOTYPE( static void ec_user2nic,
    (ether_card_t *dep, iovector_dat_t *iovp,
    vir_bytes offset, int nic_addr,
    vir_bytes count) ) ;
_PROTOTYPE( static void ec_nic2user,
    (ether_card_t *ec, int nic_addr,
    iovector_dat_t *iovp, vir_bytes offset,
    vir_bytes count) ) ;
_PROTOTYPE( static int calc_iovec_size, (iovec_dat_t *iovp) ) ;
_PROTOTYPE( static void ec_next_iovec, (iovec_dat_t *iovp) ) ;
_PROTOTYPE( static void do_getstat, (message *mp) ) ;
_PROTOTYPE( static void put_userdata,
    (int user_proc,
    vir_bytes user_addr, vir_bytes count,
    void *loc_addr) ) ;
_PROTOTYPE( static void do_stop, (message *mp) ) ;
_PROTOTYPE( static void do_getname, (message *mp) ) ;

_PROTOTYPE( static void lance_dump, (void) ) ;
_PROTOTYPE( static void lance_stop, (void) ) ;
_PROTOTYPE( static void getAddressing, (int devind, ether_card_t *ec) ) ;

/* probe+init LANCE cards */
_PROTOTYPE( static int lance_probe, (ether_card_t *ec) ) ;
_PROTOTYPE( static void lance_init_card, (ether_card_t *ec) ) ;

/* --- LANCE --- */
/* General */
#define Address unsigned long

/* Minix 3 */
#define virt_to_bus(x) (vir2phys((unsigned long)x))
unsigned long vir2phys( unsigned long x )
{
    int r;
    unsigned long value;

    if ( (r=sys_umap( SELF, D, x, 4, &value )) != OK )
        panic( "lance", "sys_umap failed", r );

    return value;
}

/* DMA limitations */
#define DMA_ADDR_MASK 0xFFFFF /* mask to verify DMA address is 24-bit */

#define CORRECT_DMA_MEM() ( (virt_to_bus(lance + sizeof(lance)) & ~DMA_ADDR_MASK) == 0 )

#define ETH_FRAME_LEN 1518

#define LANCE_MUST_PAD 0x00000001
#define LANCE_ENABLE_AUTOSELECT 0x00000002
#define LANCE_SELECT_PHONELINE 0x00000004
#define LANCE_MUST_UNRESET 0x00000008

static const struct lance_chip_type
{
    int id_number;
    const char *name;
    int flags;
} chip_table[] = {
    {0x0000, "LANCE 7990", /* Ancient lance chip. */
    LANCE_MUST_PAD + LANCE_MUST_UNRESET},
    {0x0003, "PCnet/ISA 79C960", /* 79C960 PCnet/ISA. */
    LANCE_ENABLE_AUTOSELECT},
    {0x2260, "PCnet/ISA+ 79C961", /* 79C961 PCnet/ISA+, Plug-n-Play. */
    LANCE_ENABLE_AUTOSELECT},
    {0x2420, "PCnet/PCI 79C970", /* 79C970 or 79C974 PCnet-SCSI, PCI. */
    LANCE_ENABLE_AUTOSELECT},

```

```

{0x2430, "PCnet32", /* 79C965 PCnet for VL bus. */
  LANCE_ENABLE_AUTOSELECT},
{0x2621, "PCnet/PCI-II 79C970A", /* 79C970A PCInetPCI II. */
  LANCE_ENABLE_AUTOSELECT},
{0x2625, "PCnet-FAST III 79C973", /* 79C973 PCInet-FAST III. */
  LANCE_ENABLE_AUTOSELECT},
{0x2626, "PCnet/HomePNA 79C978",
  LANCE_ENABLE_AUTOSELECT|LANCE_SELECT_PHONELINE},
{0x0, "PCnet (unknown)",
  LANCE_ENABLE_AUTOSELECT},
};

/* ##### for LANCE device ##### */
#define LANCE_ETH_ADDR 0x0
#define LANCE_DATA 0x10
#define LANCE_ADDR 0x12
#define LANCE_RESET 0x14
#define LANCE_BUS_IF 0x16
#define LANCE_TOTAL_SIZE 0x18

/* Use 2^4=16 {Rx,Tx} buffers */
#define LANCE_LOG_RX_BUFFERS 4
#define RX_RING_SIZE (1 << (LANCE_LOG_RX_BUFFERS))
#define RX_RING_MOD_MASK (RX_RING_SIZE - 1)
#define RX_RING_LEN_BITS ((LANCE_LOG_RX_BUFFERS) << 29)

#define LANCE_LOG_TX_BUFFERS 4
#define TX_RING_SIZE (1 << (LANCE_LOG_TX_BUFFERS))
#define TX_RING_MOD_MASK (TX_RING_SIZE - 1)
#define TX_RING_LEN_BITS ((LANCE_LOG_TX_BUFFERS) << 29)

/* for lance_interface */
struct lance_init_block
{
  unsigned short mode;
  unsigned char phys_addr[6];
  unsigned long filter[2];
  Address rx_ring;
  Address tx_ring;
};

struct lance_rx_head
{
  union {
    Address base;
    unsigned char addr[4];
  } u;
  short buf_length; /* 2s complement */
  short msg_length;
};

struct lance_tx_head
{
  union {
    Address base;
    unsigned char addr[4];
  } u;
  short buf_length; /* 2s complement */
  short misc;
};

struct lance_interface
{
  struct lance_init_block init_block;
  struct lance_rx_head rx_ring[RX_RING_SIZE];
  struct lance_tx_head tx_ring[TX_RING_SIZE];
  unsigned char rbuf[RX_RING_SIZE][ETH_FRAME_LEN];
  unsigned char tbuf[TX_RING_SIZE][ETH_FRAME_LEN];
};

/* ===== global variables ===== */
static struct lance_interface *lp;
static char lance[sizeof(struct lance_interface)+8];
static int rx_slot_nr = 0; /* Rx-slot number */

```

```

static int tx_slot_nr = 0;          /* Tx-slot number */
static int cur_tx_slot_nr = 0;      /* Tx-slot number */
static char isstored[TX_RING_SIZE]; /* Tx-slot in-use */
static char *progrname;

/*=====
 *                               lance_task                               *
 *=====*/
void main( int argc, char **argv )
{
    message m;
    int i,irq,r, tasknr;
    ether_card_t *ec;
    long v;
    int fkeys, sfkeys;
    (progrname=strrchr(argv[0],'/')) ? progrname++ : (progrname=argv[0]);

    env_setargs( argc, argv );

    fkeys = sfkeys = 0; bit_set( sfkeys, 7 );

#ifdef 0
    if ( (r = fkey_map(&fkeys, &sfkeys)) != OK )
        printf( "Error registering key\n" );
#endif

    if((eth_tasknr=getprocnr()) < 0)
        panic("lance", "couldn't get own proc nr", i);

    v= 0;
    (void) env_parse("ETH_IGN_PROTO", "x", 0, &v, 0x0000L, 0xFFFFL);
    eth_ign_proto= htons((ul6_t) v);

    /* Try to notify inet that we are present (again) */
    r = _pm_findproc("inet", &tasknr);
    if (r == OK)
        notify(tasknr);

    while (TRUE)
    {
        for (i=0;i<EC_PORT_NR_MAX;++i)
        {
            ec= &ec_table[i];
            if (ec->ec_irq != 0)
                sys_irqenable(&ec->ec_hook);
        }

        if ((r= receive(ANY, &m)) != OK)
            panic( "lance", "receive failed", r);

        for (i=0;i<EC_PORT_NR_MAX;++i)
        {
            ec= &ec_table[i];
            if (ec->ec_irq != 0)
                sys_irqdisable(&ec->ec_hook);
        }

        /*printf( "." );*/

        switch (m.m_type){
        case DEV_PING:    notify(m.m_source);                continue;
        case DL_WRITE:    do_vwrite(&m, FALSE, FALSE);       break;
        case DL_WRITEV:   do_vwrite(&m, FALSE, TRUE);        break;
        case DL_READ:     do_vread(&m, FALSE);               break;
        case DL_READV:    do_vread(&m, TRUE);                break;
        case DL_INIT:     do_init(&m);                      break;
        case DL_GETSTAT:  do_getstat(&m);                   break;
        case DL_STOP:     do_stop(&m);                      break;
        case DL_GETNAME:  do_getname(&m);                   break;
        case FKEY_PRESSED: lance_dump();                   break;
        /*case HARD_STOP:  lance_stop();                     break;*/
        case SYS_SIG:
        {

```

```

    sigset_t set = m.NOTIFY_ARG;
    if ( sigismember( &set, SIGKSTOP ) )
        lance_stop();
}
break;
case HARD_INT:
    for (i=0; i<EC_PORT_NR_MAX; ++i)
    {
        ec= &ec_table[i];
        if (ec->mode != EC_ENABLED)
            continue;

        /*
        printf( "#.\n" );
        */

        irq=ec->ec_irq;
        /*if (ec->ec_int_pending)*/
        {
            ec->ec_int_pending = 0;
            ec_check_ints(ec);
            do_int(ec);
        }
    }
    break;
case PROC_EVENT:
    break;
default:
    panic( "lance", "illegal message", m.m_type);
}
}
}

/*=====
*                               lance_dump                               *
*=====*/
static void lance_dump()
{
    ether_card_t *ec;
    int i, isr;
    unsigned short ioaddr;

    printf("\n");
    for (i= 0, ec = &ec_table[0]; i<EC_PORT_NR_MAX; i++, ec++)
    {
        if (ec->mode == EC_DISABLED)
            printf("lance port %d is disabled\n", i);
        else if (ec->mode == EC_SINK)
            printf("lance port %d is in sink mode\n", i);

        if (ec->mode != EC_ENABLED)
            continue;

        printf("lance statistics of port %d:\n", i);

        printf("recvErr  :%8ld\t", ec->eth_stat.ets_recvErr);
        printf("sendErr  :%8ld\t", ec->eth_stat.ets_sendErr);
        printf("OVW      :%8ld\n", ec->eth_stat.ets_OVW);

        printf("CRCerr   :%8ld\t", ec->eth_stat.ets_CRCerr);
        printf("frameAll  :%8ld\t", ec->eth_stat.ets_frameAll);
        printf("missedP   :%8ld\n", ec->eth_stat.ets_missedP);

        printf("packetR   :%8ld\t", ec->eth_stat.ets_packetR);
        printf("packetT   :%8ld\t", ec->eth_stat.ets_packetT);
        printf("transDef  :%8ld\n", ec->eth_stat.ets_transDef);

        printf("collision :%8ld\t", ec->eth_stat.ets_collision);
        printf("transAb   :%8ld\t", ec->eth_stat.ets_transAb);
        printf("carrSense :%8ld\n", ec->eth_stat.ets_carrSense);

        printf("fifoUnder :%8ld\t", ec->eth_stat.ets_fifoUnder);
        printf("fifoOver  :%8ld\t", ec->eth_stat.ets_fifoOver);
        printf("CDheartbeat:%8ld\n", ec->eth_stat.ets_CDheartbeat);
    }
}

```

```

    printf("OWC      :%8ld\t", ec->eth_stat.ets_OWC);

    ioaddr = ec->ec_port;
    out_word(ioaddr+LANCE_ADDR, 0x00);
    isr=in_word(ioaddr+LANCE_DATA);
    printf("isr=0x%x+0x%x, flags=0x%x\n", isr,
           in_word(ioaddr+LANCE_DATA), ec->flags);

    printf("irq = %d\tioadr = %d\n", ec->ec_irq, ec->ec_port);
}
}

/*=====*
 *                               lance_stop                               *
 *=====*/
static void lance_stop()
{
    message mess;
    int i;

    for (i= 0; i<EC_PORT_NR_MAX; i++)
    {
        if (ec_table[i].mode != EC_ENABLED)
            continue;
        mess.m_type= DL_STOP;
        mess.DL_PORT= i;
        do_stop(&mess);
    }

    /*printf("LANCE driver stopped.\n");*/

    sys_exit( 0 );
}

/*=====*
 *                               do_init                               *
 *=====*/
static void do_init(mp)
message *mp;
{
    int port;
    ether_card_t *ec;
    message reply_mess;

pci_init();

    port = mp->DL_PORT;
    if (port < 0 || port >= EC_PORT_NR_MAX)
    {
        reply_mess.m_type= DL_INIT_REPLY;
        reply_mess.m3_il= ENXIO;
        mess_reply(mp, &reply_mess);
        return;
    }
    ec= &ec_table[port];
    strcpy(ec->port_name, "eth_card#0");
    ec->port_name[9] += port;
    if (ec->mode == EC_DISABLED)
    {
        /* This is the default, try to (re)locate the device. */
        /* only try to enable if memory is correct for DMA */
        if ( CORRECT_DMA_MEM() )
        {
            conf_hw(ec);
        }
        else
        {
            report( "LANCE", "DMA denied because address out of range", NO_NUM );
        }

        if (ec->mode == EC_DISABLED)
        {

```

```

        /* Probe failed, or the device is configured off. */
        reply_mess.m_type= DL_INIT_REPLY;
        reply_mess.m3_i1= ENXIO;
        mess_reply(mp, &reply_mess);
        return;
    }
    if (ec->mode == EC_ENABLED)
        ec_init(ec);
}

if (ec->mode == EC_SINK)
{
    ec->mac_address.ea_addr[0] =
        ec->mac_address.ea_addr[1] =
        ec->mac_address.ea_addr[2] =
        ec->mac_address.ea_addr[3] =
        ec->mac_address.ea_addr[4] =
        ec->mac_address.ea_addr[5] = 0;
    ec_confaddr(ec);
    reply_mess.m_type = DL_INIT_REPLY;
    reply_mess.m3_i1 = mp->DL_PORT;
    reply_mess.m3_i2 = EC_PORT_NR_MAX;
    *(ether_addr_t *) reply_mess.m3_cal = ec->mac_address;
    mess_reply(mp, &reply_mess);
    return;
}
assert(ec->mode == EC_ENABLED);
assert(ec->flags & ECF_ENABLED);

ec->flags &= ~(ECF_PROMISC | ECF_MULTI | ECF_BROAD);

if (mp->DL_MODE & DL_PROMISC_REQ)
    ec->flags |= ECF_PROMISC | ECF_MULTI | ECF_BROAD;
if (mp->DL_MODE & DL_MULTI_REQ)
    ec->flags |= ECF_MULTI;
if (mp->DL_MODE & DL_BROAD_REQ)
    ec->flags |= ECF_BROAD;

ec->client = mp->m_source;
ec_reinit(ec);

reply_mess.m_type = DL_INIT_REPLY;
reply_mess.m3_i1 = mp->DL_PORT;
reply_mess.m3_i2 = EC_PORT_NR_MAX;
*(ether_addr_t *) reply_mess.m3_cal = ec->mac_address;

mess_reply(mp, &reply_mess);
}

/*=====
 *                               do_int                               *
 *=====*/
static void do_int(ec)
ether_card_t *ec;
{
    if (ec->flags & (ECF_PACK_SEND | ECF_PACK_RECV))
        reply(ec, OK, TRUE);
}

#if 0
/*=====
 *                               ec_handler                           *
 *=====*/
static int ec_handler(hook)
irq_hook_t *hook;
{
    /* LANCE interrupt, send message and reenale interrupts. */
    #if 0
        printf(">> ec_handler():\n");
    #endif

    structof(ether_card_t, ec_hook, hook)->ec_int_pending= 1;

```

```

    notify(eth_tasknr);

    return 0;
}
#endif

/*=====
 *                               conf_hw                               *
 *=====*/
static void conf_hw(ec)
ether_card_t *ec;
{
    static eth_stat_t empty_stat = {0, 0, 0, 0, 0, 0      /* ,... */ };

    int ifnr;
    ec_conf_t *ecp;

    ec->mode= EC_DISABLED;      /* Superfluous */
    ifnr= ec->ec_table;

    ecp= &ec_conf[ifnr];
    update_conf(ec, ecp);
    if (ec->mode != EC_ENABLED)
        return;

    if (!lance_probe(ec))
    {
        printf("%s: No ethernet card found on PCI-BIOS info.\n",
               ec->port_name);
        ec->mode= EC_DISABLED;
        return;
    }

    /* Allocate a memory segment, programmed I/O should set the
     * memory segment (linmem) to zero.
     */
    if (ec->ec_linmem != 0)
    {
        assert( 0 );
        /*phys2seg(&ec->ec_memseg, &ec->ec_memoff, ec->ec_linmem);*/
    }

    /* XXX */ if (ec->ec_linmem == 0) ec->ec_linmem= 0xFFFFF0000;

    ec->flags = ECF_EMPTY;
    ec->eth_stat = empty_stat;
}

/*=====
 *                               update_conf                               *
 *=====*/
static void update_conf(ec, ecp)
ether_card_t *ec;
ec_conf_t *ecp;
{
    long v;
    static char ec_fmt[] = "x:d:x:x";

    /* Get the default settings and modify them from the environment. */
    ec->mode= EC_SINK;
    v= ecp->ec_port;
    switch (env_parse(ecp->ec_envvar, ec_fmt, 0, &v, 0x0000L, 0xFFFFL)) {
    case EP_OFF:
        ec->mode= EC_DISABLED;
        break;
    case EP_ON:
    case EP_SET:
        ec->mode= EC_ENABLED;      /* Might become disabled if
                                   * all probes fail */
        break;
    }

    ec->ec_port= v;

```

```

v= ecp->ec_irq | DEI_DEFAULT;
(void) env_parse(ecp->ec_envvar, ec_fmt, 1, &v, 0L,
                (long) NR_IRQ_VECTORS - 1);
ec->ec_irq= v;

v= ecp->ec_mem;
(void) env_parse(ecp->ec_envvar, ec_fmt, 2, &v, 0L, 0xFFFFFL);
ec->ec_linmem= v;

v= 0;
(void) env_parse(ecp->ec_envvar, ec_fmt, 3, &v, 0x2000L, 0x8000L);
ec->ec_ramsize= v;
}

/*=====
 *                               ec_init                               *
 *=====*/
static void ec_init(ec)
ether_card_t *ec;
{
    int i, r;

    /* General initialization */
    ec->flags = ECF_EMPTY;
    /*disable_irq(ec->ec_irq);*/
    lance_init_card(ec); /* Get mac_address, etc ...*/

    ec_confaddr(ec);

#ifdef VERBOSE
    printf("%s: Ethernet address ", ec->port_name);
    for (i= 0; i < 6; i++)
        printf("%x%c", ec->mac_address.ea_addr[i],
              i < 5 ? ':' : '\n');
#endif

    /* Finish the initialization */
    ec->flags |= ECF_ENABLED;

    /* Set the interrupt handler */
    /*put_irq_handler(&ec->ec_hook, ec->ec_irq, ec_handler);*/
    ec->ec_hook = ec->ec_irq;
    if ((r=sys_irqsetpolicy(ec->ec_irq, 0, &ec->ec_hook)) != OK)
        printf("lance: error, couldn't set IRQ policy: %d\n", r);

    /* enable_irq(ec->ec_irq); */

    /* enter_kdebug(">> ec_init():"); */

    return;
}

/*=====
 *                               reply                               *
 *=====*/
static void reply(ec, err, may_block)
ether_card_t *ec;
int err;
int may_block;
{
    message reply;
    int status, r;
    clock_t now;

    status = 0;
    if (ec->flags & ECF_PACK_SEND)
        status |= DL_PACK_SEND;
    if (ec->flags & ECF_PACK_RECV)
        status |= DL_PACK_RECV;

    reply.m_type = DL_TASK_REPLY;

```



```

reply.DL_PORT = ec - ec_table;
reply.DL_PROC = ec->client;
reply.DL_STAT = status | ((u32_t) err << 16);
reply.DL_COUNT = ec->read_s;
#if 1
    if ((r=getuptime(&now)) != OK)
        panic("lance", "getuptime() failed:", r);
    reply.DL_CLCK = now;
#else
    reply.DL_CLCK = 0;
#endif

    r = send(ec->client, &reply);
#if 1
    if (r == ELOCKED && may_block)
    {
/*      enter_kdebug(">> lance_task: ELOCKED!"); */
        return;
    }
#endif
    if (r < 0)
        panic("lance", "send failed:", r);

    ec->read_s = 0;
    ec->flags &= ~(ECF_PACK_SEND | ECF_PACK_RECV);
}

/*=====
*                                     mess_reply                                     *
*=====*/
static void mess_reply(req, reply_mess)
message *req;
message *reply_mess;
{
    if (send(req->m_source, reply_mess) != OK)
        panic("lance", "unable to mess_reply", NO_NUM);
}

/*=====
*                                     ec_confaddr                                 *
*=====*/
static void ec_confaddr(ec)
ether_card_t *ec;
{
    int i;
    char eakey[16];
    static char eafmt[] = "X:X:X:X:X";
    long v;

    /* User defined ethernet address? */
    strcpy(eakey, ec_conf[ec-ec_table].ec_envvar);
    strcat(eakey, "_EA");

    for (i = 0; i < 6; i++)
    {
        v = ec->mac_address.ea_addr[i];
        if (env_parse(eakey, eafmt, i, &v, 0x00L, 0xFFL) != EP_SET)
            break;
        ec->mac_address.ea_addr[i] = v;
    }

    if (i != 0 && i != 6)
    {
        /* It's all or nothing; force a panic. */
        (void) env_parse(eakey, "?", 0, &v, 0L, 0L);
    }
}

/*=====
*                                     ec_reinit                                   *
*=====*/

```

```

static void ec_reinit(ec)
ether_card_t *ec;
{
    int i;
    unsigned short ioaddr = ec->ec_port;

    out_word(ioaddr+LANCE_ADDR, 0x0);
    (void)in_word(ioaddr+LANCE_ADDR);
    out_word(ioaddr+LANCE_DATA, 0x4);          /* stop */

    /* purge Tx-ring */
    tx_slot_nr = cur_tx_slot_nr = 0;
    for (i=0; i<TX_RING_SIZE; i++) {
        lp->tx_ring[i].u.base = 0;
        isstored[i]=0;
    }

    /* re-init Rx-ring */
    rx_slot_nr = 0;
    for (i=0; i<RX_RING_SIZE; i++)
    {
        lp->rx_ring[i].buf_length = -ETH_FRAME_LEN;
        lp->rx_ring[i].u.addr[3] |= 0x80;
    }

    /* Set 'Receive Mode' */
    if (ec->flags & ECF_PROMISC)
    {
        out_word(ioaddr+LANCE_ADDR, 0xf);
        out_word(ioaddr+LANCE_DATA, 0x8000);
    }
    else
    {
        if (ec->flags & (ECF_BROAD | ECF_MULTI))
        {
            out_word(ioaddr+LANCE_ADDR, 0xf);
            out_word(ioaddr+LANCE_DATA, 0x0000);
        }
        else
        {
            out_word(ioaddr+LANCE_ADDR, 0xf);
            out_word(ioaddr+LANCE_DATA, 0x4000);
        }
    }

    out_word(ioaddr+LANCE_ADDR, 0x0);
    (void)in_word(ioaddr+LANCE_ADDR);
    out_word(ioaddr+LANCE_DATA, 0x142);      /* start && enable interrupt */

    return;
}

/*=====
 *                               ec_check_ints                               *
 *=====*/
static void ec_check_ints(ec)
ether_card_t *ec;
{
    int must_restart=0;
    int check,status;
    int isr=0x0000;
    unsigned short ioaddr = ec->ec_port;

    if (!(ec->flags & ECF_ENABLED))
        panic( "lance", "got premature interrupt", NO_NUM);

    for (;;)
    {
#ifdef 0
        printf("ETH: Reading ISR...");
#endif
        out_word(ioaddr+LANCE_ADDR, 0x00);
        isr=in_word(ioaddr+LANCE_DATA);
        if (isr & 0x8600)

```

```
        out_word( ioaddr+LANCE_DATA, isr & ~0x004f);
        out_word(ioaddr+LANCE_DATA, 0x7940);
#if 0
        printf("ISR=0x%x...",in_word(ioaddr+LANCE_DATA));
#endif
#define ISR_WINT 0x0200
#define ISR_RINT 0x0400
#define ISR_RERR 0x1000
#define ISR_WERR 0x4000
#define ISR_ERR 0x8000
#define ISR_RST 0x0000

        if ((isr & (ISR_WINT|ISR_RINT|ISR_RERR|ISR_WERR|ISR_ERR)) == 0x0000)
        {
#if 0
                printf("OK\n");
#endif
                break;
        }

        if (isr & ISR_RERR)
        {
#if 0
                printf("RERR\n");
#endif
                ec->eth_stat.ets_recvErr++;
        }
        if ((isr & ISR_WERR) || (isr & ISR_WINT))
        {
                if (isr & ISR_WERR)
                {
#if 0
                        printf("WERR\n");
#endif
                        ec->eth_stat.ets_sendErr++;
                }
                if (isr & ISR_WINT)
                {
#if 0
                        printf("WINT\n");
#endif

                        /* status check: restart if needed. */
                        status = lp->tx_ring[cur_tx_slot_nr].u.base;

                        /* ??? */
                        if (status & 0x40000000)
                        {
                                status = lp->tx_ring[cur_tx_slot_nr].misc;
                                ec->eth_stat.ets_sendErr++;
                                if (status & 0x0400)
                                        ec->eth_stat.ets_transAb++;
                                if (status & 0x0800)
                                        ec->eth_stat.ets_carrSense++;
                                if (status & 0x1000)
                                        ec->eth_stat.ets_OWC++;
                                if (status & 0x4000)
                                {
                                        ec->eth_stat.ets_fifoUnder++;
                                        must_restart=1;
                                }
                        }
                        else
                        {
                                if (status & 0x18000000)
                                        ec->eth_stat.ets_collision++;
                                ec->eth_stat.ets_packetT++;
                        }
                }
                /* transmit a packet on the next slot if it exists. */
                check = 0;
                if (isstored[cur_tx_slot_nr]==1)
                {
                        /* free the tx-slot just transmitted */
                        isstored[cur_tx_slot_nr]=0;
                }
        }
}
```

```

        cur_tx_slot_nr = (++cur_tx_slot_nr) & TX_RING_MOD_MASK;

        /* next tx-slot is ready? */
        if (isstored[cur_tx_slot_nr]==1)
            check=1;
        else
            check=0;
    }
    else
    {
        panic( "lance", "got premature WINT...", NO_NUM);
    }
    if (check==1)
    {
        lp->tx_ring[cur_tx_slot_nr].u.addr[3] = 0x83;
        out_word(ioaddr+LANCE_ADDR, 0x0000);
        out_word(ioaddr+LANCE_DATA, 0x0048);
    }
    else
        if (check==-1)
            continue;
    /* we set a buffered message in the slot if it exists. */
    /* and transmit it, if needed. */
    if (ec->flags & ECF_SEND_AVAIL)
        ec_send(ec);
}
if (isr & ISR_RINT)
{
#if 0
    printf("RINT\n");
#endif
    ec_recv(ec);
}

if (isr & ISR_RST)
{
    ec->flags = ECF_STOPPED;
    break;
}

/* ??? cf. lance driver on linux */
if (must_restart == 1)
{
#if 0
    printf("ETH: restarting...\n");
#endif
    out_word(ioaddr+LANCE_ADDR, 0x0);
    (void)in_word(ioaddr+LANCE_ADDR);
    out_word(ioaddr+LANCE_DATA, 0x4); /* stop */
    out_word(ioaddr+LANCE_DATA, 0x2); /* start */
}

if ((ec->flags & (ECF_READING|ECF_STOPPED)) == (ECF_READING|ECF_STOPPED))
{
#if 0
    printf("ETH: resetting...\n");
#endif
    ec_reset(ec);
}
}

/*=====
 *                               ec_reset                               *
 *=====*/
static void ec_reset(ec)
ether_card_t *ec;
{
    /* Stop/start the chip, and clear all RX,TX-slots */
    unsigned short ioaddr = ec->ec_port;
    int i;

    out_word(ioaddr+LANCE_ADDR, 0x0);
    (void)in_word(ioaddr+LANCE_ADDR);

```

```

out_word(ioaddr+LANCE_DATA, 0x4);          /* stop */
out_word(ioaddr+LANCE_DATA, 0x2);          /* start */

/* purge Tx-ring */
tx_slot_nr = cur_tx_slot_nr = 0;
for (i=0; i<TX_RING_SIZE; i++) {
    lp->tx_ring[i].u.base = 0;
    isstored[i]=0;
}

/* re-init Rx-ring */
rx_slot_nr = 0;
for (i=0; i<RX_RING_SIZE; i++)
{
    lp->rx_ring[i].buf_length = -ETH_FRAME_LEN;
    lp->rx_ring[i].u.addr[3] |= 0x80;
}

/* store a buffered message on the slot if exists */
ec_send(ec);
ec->flags &= ~ECF_STOPPED;
}

/*=====
 *                               ec_send                               *
 *=====*/
static void ec_send(ec)
ether_card_t *ec;
{
    /* from ec_check_ints() or ec_reset(). */
    /* this function processes the buffered message. (slot/transmit) */
    if (!(ec->flags & ECF_SEND_AVAIL))
        return;

    ec->flags &= ~ECF_SEND_AVAIL;
    switch(ec->sendmsg.m_type)
    {
        case DL_WRITE: do_vwrite(&ec->sendmsg, TRUE, FALSE);      break;
        case DL_WRITEV: do_vwrite(&ec->sendmsg, TRUE, TRUE);       break;
        default:
            panic( "lance", "wrong type:", ec->sendmsg.m_type);
            break;
    }
}

/*=====
 *                               do_vread                             *
 *=====*/
static void do_vread(mp, vectored)
message *mp;
int vectored;
{
    int port, count, size;
    ether_card_t *ec;

    port = mp->DL_PORT;
    count = mp->DL_COUNT;
    ec = &ec_table[port];
    ec->client = mp->DL_PROC;

    if (vectored)
    {
        get_userdata(mp->DL_PROC, (vir_bytes) mp->DL_ADDR,
            (count > IOVEC_NR ? IOVEC_NR : count) *
                sizeof(iovec_t), ec->read_iovec.iod_iovec);
        ec->read_iovec.iod_iovec_s = count;
        ec->read_iovec.iod_proc_nr = mp->DL_PROC;
        ec->read_iovec.iod_iovec_addr = (vir_bytes) mp->DL_ADDR;

        ec->tmp_iovec = ec->read_iovec;
        size = calc_iovec_size(&ec->tmp_iovec);
    }
    else
    {

```

```

    ec->read_iovec.iod_iovec[0].iov_addr = (vir_bytes) mp->DL_ADDR;
    ec->read_iovec.iod_iovec[0].iov_size = mp->DL_COUNT;
    ec->read_iovec.iod_iovec_s          = 1;
    ec->read_iovec.iod_proc_nr          = mp->DL_PROC;
    ec->read_iovec.iod_iovec_addr       = 0;

    size= count;
}
ec->flags |= ECF_READING;

ec_rcv(ec);

if ((ec->flags & (ECF_READING|ECF_STOPPED)) == (ECF_READING|ECF_STOPPED))
    ec_reset(ec);
reply(ec, OK, FALSE);
}

/*=====
 *                               ec_rcv                               *
 *=====*/
static void ec_rcv(ec)
ether_card_t *ec;
{
    vir_bytes length;
    int packet_processed;
    int status;
    unsigned short ioaddr = ec->ec_port;

    if ((ec->flags & ECF_READING)==0)
        return;
    if (!(ec->flags & ECF_ENABLED))
        return;

    /* we check all the received slots until find a properly received packet */
    packet_processed = FALSE;
    while (!packet_processed)
    {
        status = lp->rx_ring[rx_slot_nr].u.base >> 24;
        if ( (status & 0x80) == 0x00 )
        {
            status = lp->rx_ring[rx_slot_nr].u.base >> 24;

            /* ??? */
            if (status != 0x03)
            {
                if (status & 0x01)
                    ec->eth_stat.ets_rcvErr++;
                if (status & 0x04)
                    ec->eth_stat.ets_fifoOver++;
                if (status & 0x08)
                    ec->eth_stat.ets_CRCerr++;
                if (status & 0x10)
                    ec->eth_stat.ets_OVW++;
                if (status & 0x20)
                    ec->eth_stat.ets_frameAll++;
                length = 0;
            }
            else
            {
                ec->eth_stat.ets_packetR++;
                length = lp->rx_ring[rx_slot_nr].msg_length;
            }
            if (length > 0)
            {
                ec_nic2user(ec, (int)(lp->rbuf[rx_slot_nr]),
                           &ec->read_iovec, 0, length);

                ec->read_s = length;
                ec->flags |= ECF_PACK_RECV;
                ec->flags &= ~ECF_READING;
                packet_processed = TRUE;
            }
        }
        /* set up this slot again, and we move to the next slot */
        lp->rx_ring[rx_slot_nr].buf_length = -ETH_FRAME_LEN;
    }
}

```

```

        lp->rx_ring[rx_slot_nr].u.addr[3] |= 0x80;

        out_word(ioaddr+LANCE_ADDR, 0x00);
        out_word(ioaddr+LANCE_DATA, 0x7940);

        rx_slot_nr = (++rx_slot_nr) & RX_RING_MOD_MASK;
    }
    else
        break;
}
}

/*=====
 *                               do_vwrite                               *
 *=====*/
static void do_vwrite(mp, from_int, vectored)
message *mp;
int from_int;
int vectored;
{
    int port, count, check;
    ether_card_t *ec;
    unsigned short ioaddr;

    port = mp->DL_PORT;
    count = mp->DL_COUNT;
    ec = &ec_table[port];
    ec->client = mp->DL_PROC;

    if (isstored[tx_slot_nr]==1)
    {
        /* all slots are used, so this message is buffered */
        ec->sendmsg = *mp;
        ec->flags |= ECF_SEND_AVAIL;
        reply(ec, OK, FALSE);
        return;
    }

    /* convert the message to write_iovec */
    if (vectored)
    {
        get_userdata(mp->DL_PROC, (vir_bytes) mp->DL_ADDR,
                     (count > IOVEC_NR ? IOVEC_NR : count) *
                     sizeof(iovec_t), ec->write_iovec.iod_iovec);

        ec->write_iovec.iod_iovec_s = count;
        ec->write_iovec.iod_proc_nr = mp->DL_PROC;
        ec->write_iovec.iod_iovec_addr = (vir_bytes) mp->DL_ADDR;

        ec->tmp_iovec = ec->write_iovec;
        ec->write_s = calc_iovec_size(&ec->tmp_iovec);
    }
    else
    {
        ec->write_iovec.iod_iovec[0].iov_addr = (vir_bytes) mp->DL_ADDR;
        ec->write_iovec.iod_iovec[0].iov_size = mp->DL_COUNT;

        ec->write_iovec.iod_iovec_s = 1;
        ec->write_iovec.iod_proc_nr = mp->DL_PROC;
        ec->write_iovec.iod_iovec_addr = 0;

        ec->write_s = mp->DL_COUNT;
    }

    /* copy write_iovec to the slot on DMA address */
    ec_user2nic(ec, &ec->write_iovec, 0,
                (int)(lp->tbuf[tx_slot_nr]), ec->write_s);
    /* set-up for transmitting, and transmit it if needed. */
    lp->tx_ring[tx_slot_nr].buf_length = -ec->write_s;
    lp->tx_ring[tx_slot_nr].misc = 0x0;
    lp->tx_ring[tx_slot_nr].u.base
        = virt_to_bus(lp->tbuf[tx_slot_nr]) & 0xffffffff;
    isstored[tx_slot_nr]=1;
    if (cur_tx_slot_nr == tx_slot_nr)

```

```

    check=1;
else
    check=0;
tx_slot_nr = (++tx_slot_nr) & TX_RING_MOD_MASK;

if (check == 1)
{
    ioaddr = ec->ec_port;
    lp->tx_ring[cur_tx_slot_nr].u.addr[3] = 0x83;
    out_word(ioaddr+LANCE_ADDR, 0x0000);
    out_word(ioaddr+LANCE_DATA, 0x0048);
}

ec->flags |= ECF_PACK_SEND;

/* reply by calling do_int() if this function is called from interrupt. */
if (from_int)
    return;
reply(ec, OK, FALSE);
}

/*=====
*                               get_userdata                               *
*=====*/
static void get_userdata(user_proc, user_addr, count, loc_addr)
int user_proc;
vir_bytes user_addr;
vir_bytes count;
void *loc_addr;
{
    /*
    phys_bytes src;

    src = numap_local(user_proc, user_addr, count);
    if (!src)
        panic("lance", "umap failed", NO_NUM);

    phys_copy(src, vir2phys(loc_addr), (phys_bytes) count);
    */
    int cps;
    cps = sys_datacopy(user_proc, user_addr, SELF, (vir_bytes) loc_addr, count);
    if (cps != OK) printf("lance: warning,scopy failed: %d\n", cps);
}

/*=====
*                               ec_user2nic                               *
*=====*/
static void ec_user2nic(ec, iovep, offset, nic_addr, count)
ether_card_t *ec;
iovec_dat_t *iovp;
vir_bytes offset;
int nic_addr;
vir_bytes count;
{
    /*phys_bytes phys_hw, phys_user;*/
    int bytes, i, r;

    /*
    phys_hw = vir2phys(nic_addr);
    */
    i= 0;
    while (count > 0)
    {
        if (i >= IOVEC_NR)
        {
            ec_next_iovec(iovp);
            i= 0;
            continue;
        }
        if (offset >= iovep->iod_iovec[i].iov_size)
        {
            offset -= iovep->iod_iovec[i].iov_size;
            i++;

```



```

        continue;
    }
    bytes = iovp->iod_iovec[i].iov_size - offset;
    if (bytes > count)
        bytes = count;

    /*
    phys_user = numap_local(iovp->iod_proc_nr,
                           iovp->iod_iovec[i].iov_addr + offset, bytes);

    phys_copy(phys_user, phys_hw, (phys_bytes) bytes);
    */
    if ( (r=sys_datacopy(iovp->iod_proc_nr, iovp->iod_iovec[i].iov_addr + offset,
        SELF, nic_addr, count )) != OK )
        panic( "lance", "sys_datacopy failed", r );

    count -= bytes;
    nic_addr += bytes;
    offset += bytes;
}
}

/*=====
*                               ec_nic2user                               *
*=====*/
static void ec_nic2user(ec, nic_addr, iovp, offset, count)
ether_card_t *ec;
int nic_addr;
iovec_dat_t *iovp;
vir_bytes offset;
vir_bytes count;
{
    /*phys_bytes phys_hw, phys_user;*/
    int bytes, i, r;

    /*phys_hw = vir2phys(nic_addr);*/

    i= 0;
    while (count > 0)
    {
        if (i >= IOVEC_NR)
        {
            ec_next_iovec(iovp);
            i= 0;
            continue;
        }
        if (offset >= iovp->iod_iovec[i].iov_size)
        {
            offset -= iovp->iod_iovec[i].iov_size;
            i++;
            continue;
        }
        bytes = iovp->iod_iovec[i].iov_size - offset;
        if (bytes > count)
            bytes = count;
        /*
        phys_user = numap_local(iovp->iod_proc_nr,
                               iovp->iod_iovec[i].iov_addr + offset, bytes);

        phys_copy(phys_hw, phys_user, (phys_bytes) bytes);
        */
        if ( (r=sys_datacopy( SELF, nic_addr, iovp->iod_proc_nr, iovp->iod_iovec[i].iov_addr
r + offset, bytes )) != OK )
            panic( "lance", "sys_datacopy failed: ", r );

        count -= bytes;
        nic_addr += bytes;
        offset += bytes;
    }
}

/*=====
*                               calc_iovec_size                               *
*=====*/

```

```

/*=====*/
static int calc_iovec_size(iovp)
iovec_dat_t *iovp;
{
    int size,i;

    size = i = 0;

    while (i < iovp->iod_iovec_s)
    {
        if (i >= IOVEC_NR)
        {
            ec_next_iovec(iovp);
            i= 0;
            continue;
        }
        size += iovp->iod_iovec[i].iov_size;
        i++;
    }

    return size;
}

/*=====
 *                               ec_next_iovec                               *
 *=====*/
static void ec_next_iovec(iovp)
iovec_dat_t *iovp;
{
    iovp->iod_iovec_s -= IOVEC_NR;
    iovp->iod_iovec_addr += IOVEC_NR * sizeof(iovec_t);

    get_userdata(iovp->iod_proc_nr, iovp->iod_iovec_addr,
        (iovp->iod_iovec_s > IOVEC_NR ?
         IOVEC_NR : iovp->iod_iovec_s) * sizeof(iovec_t),
        iovp->iod_iovec);
}

/*=====
 *                               do_getstat                               *
 *=====*/
static void do_getstat(mp)
message *mp;
{
    int port;
    ether_card_t *ec;

    port = mp->DL_PORT;
    if (port < 0 || port >= EC_PORT_NR_MAX)
        panic( "lance", "illegal port", port);

    ec= &ec_table[port];
    ec->client= mp->DL_PROC;

    put_userdata(mp->DL_PROC, (vir_bytes) mp->DL_ADDR,
        (vir_bytes) sizeof(ec->eth_stat), &ec->eth_stat);
    reply(ec, OK, FALSE);
}

/*=====
 *                               put_userdata                               *
 *=====*/
static void put_userdata(user_proc, user_addr, count, loc_addr)
int user_proc;
vir_bytes user_addr;
vir_bytes count;
void *loc_addr;
{
    /*phys_bytes dst;

    dst = numap_local(user_proc, user_addr, count);
    if (!dst)
        panic( "lance", "umap failed", NO_NUM);

```

```

phys_copy(vir2phys(loc_addr), dst, (phys_bytes) count);
*/
    int cps;
    cps = sys_datacopy(SELF, (vir_bytes) loc_addr, user_proc, user_addr, count);
    if (cps != OK) printf("lance: warning,scopy failed: %d\n", cps);
}

/*=====
*                                     do_stop                                     *
*=====*/
static void do_stop(mp)
message *mp;
{
    int port;
    ether_card_t *ec;
    unsigned short ioaddr;

    port = mp->DL_PORT;
    if (port < 0 || port >= EC_PORT_NR_MAX)
        panic("lance", "illegal port", port);
    ec = &ec_table[port];

    if (!(ec->flags & ECF_ENABLED))
        return;

    ioaddr = ec->ec_port;

    out_word(ioaddr+LANCE_ADDR, 0x0);
    (void)in_word(ioaddr+LANCE_ADDR);
    out_word(ioaddr+LANCE_DATA, 0x4); /* stop */
    out_word(ioaddr+LANCE_RESET, in_word(ioaddr+LANCE_RESET)); /* reset */

    ec->flags = ECF_EMPTY;
}

static void getAddressing(devind, ec)
int devind;
ether_card_t *ec;
{
    unsigned int      membase, ioaddr;
    int reg, irq;

    for (reg = PCI_BASE_ADDRESS_0; reg <= PCI_BASE_ADDRESS_5; reg += 4)
    {
        ioaddr = pci_attr_r32(devind, reg);

        if ((ioaddr & PCI_BASE_ADDRESS_IO_MASK) == 0
            || (ioaddr & PCI_BASE_ADDRESS_SPACE_IO) == 0)
            continue;
        /* Strip the I/O address out of the returned value */
        ioaddr &= PCI_BASE_ADDRESS_IO_MASK;
        /* Get the memory base address */
        membase = pci_attr_r32(devind, PCI_BASE_ADDRESS_1);
        /* KK: Get the IRQ number */
        irq = pci_attr_r8(devind, PCI_INTERRUPT_PIN);
        if (irq)
            irq = pci_attr_r8(devind, PCI_INTERRUPT_LINE);
        /* Get the ROM base address */
        /*pci_attr_r32(devind, PCI_ROM_ADDRESS, &romaddr);
        romaddr >>= 10;*/
        /* Take the first one or the one that matches in boot ROM address */
        /*if (pci_ioaddr == 0
            || romaddr == ((unsigned long) rom.rom_segment << 4)) {*/
            ec->ec_linmem = membase;
            ec->ec_port = ioaddr;
            ec->ec_irq = irq;
        /*}*/
    }
}

/*=====
*                                     lance_probe                               *
*=====*/

```

```

static int lance_probe(ec)
ether_card_t *ec;
{
    unsigned short    pci_cmd, attached = 0;
    unsigned short    ioaddr;
    int               lance_version, chip_version;
    int devind, just_one, i, r;

    u16_t vid, did;
    u32_t bar;
    u8_t ilr;
    char *dname;

    if ((ec->ec_pcibus | ec->ec_pcidev | ec->ec_pcifunc) != 0)
    {
        /* Look for specific PCI device */
        r = pci_find_dev(ec->ec_pcibus, ec->ec_pcidev,
                        ec->ec_pcifunc, &devind);
        if (r == 0)
        {
            printf("%s: no PCI found at %d.%d.%d\n",
                   ec->port_name, ec->ec_pcibus,
                   ec->ec_pcidev, ec->ec_pcifunc);
            return 0;
        }
        pci_ids(devind, &vid, &did);
        just_one= TRUE;
    }
    else
    {
        r = pci_first_dev(&devind, &vid, &did);
        if (r == 0)
            return 0;
        just_one= FALSE;
    }

    for(;;)
    {
        for (i= 0; pcitab[i].vid != 0; i++)
        {
            if (pcitab[i].vid != vid)
                continue;
            if (pcitab[i].did != did)
                continue;
            if (pcitab[i].checkclass)
            {
                panic("lance",
                     "class check not implemented", NO_NUM);
            }
            break;
        }
        if (pcitab[i].vid != 0)
            break;

        if (just_one)
        {
            printf(
"%s: wrong PCI device (%04x/%04x) found at %d.%d.%d\n",
ec->port_name, vid, did,
ec->ec_pcibus,
ec->ec_pcidev, ec->ec_pcifunc);
            return 0;
        }

        r = pci_next_dev(&devind, &vid, &did);
        if (!r)
            return 0;
    }

    dname= pci_dev_name(vid, did);
    if (!dname)
        dname= "unknown device";

    /*

```

```

    printf("%s: ", ec->port_name);
    printf("%s ", dname);
    printf("(%x/", vid);
    printf("%x) ", did);
    printf("at %s\n", pci_slot_name(devind));
    */
    pci_reserve(devind);

/* for (i = 0; pci_dev_list[i].vendor != 0; i++) {
    if (pci_dev_list[i].suffix == 1)
    {
        ec->ec_port    = pci_dev_list[i].ioaddr;
        ec->ec_irq     = pci_dev_list[i].irq;
        ec->ec_linmem  = pci_dev_list[i].membase;
        ec->ec_bus     = pci_dev_list[i].bus;
        ec->ec_dev     = pci_dev_list[i].devfn;
        ec->ec_fnc     =
        pci_dev_list[i].suffix = -1;
        attached = 1;
        break;
    }
}
if (attached == 0)
    return 0;
*/

    getAddressing(devind, ec);

/* ===== Bus Master ? ===== */
/*pcibios_read_config_word(ec->ec_bus, ec->ec_devfn, PCI_COMMAND, &pci_cmd);*/
pci_cmd = pci_attr_r32(devind, PCI_CR);
if (!(pci_cmd & PCI_COMMAND_MASTER)) {
    pci_cmd |= PCI_COMMAND_MASTER;
    /*pcibios_write_config_word(ec->ec_bus, ec->ec_devfn, PCI_COMMAND, pci_cmd);*/
    pci_attr_w32(devind, PCI_CR, pci_cmd);
}

/* ===== Probe Details ===== */
ioaddr = ec->ec_port;

out_word(ioaddr+LANCE_RESET, in_word(ioaddr+LANCE_RESET)); /* Reset */
out_word(ioaddr+LANCE_ADDR, 0x0); /* Sw to win 0 */
if (in_word(ioaddr+LANCE_DATA) != 0x4)
{
    ec->mode=EC_DISABLED;
}
/* Probe Chip Version */
out_word(ioaddr+LANCE_ADDR, 88); /* Get the version of the chip */
if (in_word(ioaddr+LANCE_ADDR) != 88)
    lance_version = 0;
else
{
    chip_version = in_word(ioaddr+LANCE_DATA);
    out_word(ioaddr+LANCE_ADDR, 89);
    chip_version |= in_word(ioaddr+LANCE_DATA) << 16;
    if ((chip_version & 0xffff) != 0x3)
    {
        ec->mode=EC_DISABLED;
    }
    chip_version = (chip_version >> 12) & 0xffff;
    for (lance_version = 1; chip_table[lance_version].id_number != 0;
        ++lance_version)
        if (chip_table[lance_version].id_number == chip_version)
            break;
}

#if 0
    printf("%s: %s at %X:%d\n",
        ec->port_name, chip_table[lance_version].name,
        ec->ec_port, ec->ec_irq);
#endif

    return lance_version;
}

```

```

/*=====
 *
 *                               do_getname
 *=====*/
static void do_getname(mp)
message *mp;
{
    int r;

    strncpy(mp->DL_NAME, progname, sizeof(mp->DL_NAME));
    mp->DL_NAME[sizeof(mp->DL_NAME)-1] = '\0';
    mp->m_type = DL_NAME_REPLY;
    r = send(mp->m_source, mp);
    if (r != OK)
        panic("LANCE", "do_getname: send failed", r);
}

/*=====
 *
 *                               lance_init_card
 *=====*/
static void lance_init_card(ec)
ether_card_t *ec;
{
    int i;
    Address l;
    unsigned short ioaddr = ec->ec_port;

    /* ===== setup init_block(cf. lance_probe1) ===== */
    /* make sure data structure is 8-byte aligned */
    l = ((Address)lance + 7) & ~7;
    lp = (struct lance_interface *)l;
    lp->init_block.mode = 0x3; /* disable Rx and Tx */
    lp->init_block.filter[0] = lp->init_block.filter[1] = 0x0;
    /* using multiple Rx/Tx buffer */
    lp->init_block.rx_ring
        = (virt_to_bus(&lp->rx_ring) & 0xfffff) | RX_RING_LEN_BITS;
    lp->init_block.tx_ring
        = (virt_to_bus(&lp->tx_ring) & 0xfffff) | TX_RING_LEN_BITS;

    l = virt_to_bus(&lp->init_block);
    out_word(ioaddr+LANCE_ADDR, 0x1);
    (void)in_word(ioaddr+LANCE_ADDR);
    out_word(ioaddr+LANCE_DATA, (unsigned short)l);
    out_word(ioaddr+LANCE_ADDR, 0x2);
    (void)in_word(ioaddr+LANCE_ADDR);
    out_word(ioaddr+LANCE_DATA, (unsigned short)(l >> 16));
    out_word(ioaddr+LANCE_ADDR, 0x4);
    (void)in_word(ioaddr+LANCE_ADDR);
    out_word(ioaddr+LANCE_DATA, 0x915);
    out_word(ioaddr+LANCE_ADDR, 0x0);
    (void)in_word(ioaddr+LANCE_ADDR);

    /* ===== Get MAC address (cf. lance_probe1) ===== */
    for (i = 0; i < 6; ++i)
        ec->mac_address.ea_addr[i] = in_byte(ioaddr+LANCE_ETH_ADDR+i);

    /* ===== (re)start init_block(cf. lance_reset) ===== */
    /* Reset the LANCE */
    (void)in_word(ioaddr+LANCE_RESET);

    /* ----- Re-initialize the LANCE ----- */
    /* Set station address */
    for (i = 0; i < 6; ++i)
        lp->init_block.phys_addr[i] = ec->mac_address.ea_addr[i];
    /* Preset the receive ring headers */
    for (i=0; i<RX_RING_SIZE; i++) {
        lp->rx_ring[i].buf_length = -ETH_FRAME_LEN;
        /* OWN */
        lp->rx_ring[i].u.base = virt_to_bus(lp->rbuf[i]) & 0xfffff;
        /* we set the top byte as the very last thing */
        lp->rx_ring[i].u.addr[3] = 0x80;
    }
    /* Preset the transmitting ring headers */
    for (i=0; i<TX_RING_SIZE; i++) {
        lp->tx_ring[i].u.base = 0;
    }
}

```

```
    isstored[i] = 0;
}
lp->init_block.mode = 0x0;      /* enable Rx and Tx */

l = (Address)virt_to_bus(&lp->init_block);
out_word(ioaddr+LANCE_ADDR, 0x1);
(void)in_word(ioaddr+LANCE_ADDR);
out_word(ioaddr+LANCE_DATA, (short)l);
out_word(ioaddr+LANCE_ADDR, 0x2);
(void)in_word(ioaddr+LANCE_ADDR);
out_word(ioaddr+LANCE_DATA, (short)(l >> 16));
out_word(ioaddr+LANCE_ADDR, 0x4);
(void)in_word(ioaddr+LANCE_ADDR);
out_word(ioaddr+LANCE_DATA, 0x915);
out_word(ioaddr+LANCE_ADDR, 0x0);
(void)in_word(ioaddr+LANCE_ADDR);

/* ----- start when init done. ----- */
out_word(ioaddr+LANCE_DATA, 0x4);      /* stop */
out_word(ioaddr+LANCE_DATA, 0x1);      /* init */
for (i = 10000; i > 0; --i)
    if (in_word(ioaddr+LANCE_DATA) & 0x100)
        break;

/* Set 'Multicast Table' */
for (i=0;i<4;++i)
{
    out_word(ioaddr+LANCE_ADDR, 0x8 + i);
    out_word(ioaddr+LANCE_DATA, 0xffff);
}

/* Set 'Receive Mode' */
if (ec->flags & ECF_PROMISC)
{
    out_word(ioaddr+LANCE_ADDR, 0xf);
    out_word(ioaddr+LANCE_DATA, 0x8000);
}
else
{
    if (ec->flags & (ECF_BROAD | ECF_MULTI))
    {
        out_word(ioaddr+LANCE_ADDR, 0xf);
        out_word(ioaddr+LANCE_DATA, 0x0000);
    }
    else
    {
        out_word(ioaddr+LANCE_ADDR, 0xf);
        out_word(ioaddr+LANCE_DATA, 0x4000);
    }
}

out_word(ioaddr+LANCE_ADDR, 0x0);
(void)in_word(ioaddr+LANCE_ADDR);
out_word(ioaddr+LANCE_DATA, 0x142); /* start && enable interrupt */

return;
}
```

```
/*#include "kernel.h"*/
#include <net/gen/ether.h>
#include <net/gen/eth_io.h>

/* PCI STUFF */
#define PCI_BASE_ADDRESS_0          0x10
#define PCI_BASE_ADDRESS_1          0x14
#define PCI_BASE_ADDRESS_2          0x18
#define PCI_BASE_ADDRESS_3          0x1c
#define PCI_BASE_ADDRESS_4          0x20
#define PCI_BASE_ADDRESS_5          0x24

#define PCI_BASE_ADDRESS_IO_MASK    (~0x03)
#define PCI_BASE_ADDRESS_SPACE_IO   0x01
#define PCI_INTERRUPT_LINE          0x3c
#define PCI_INTERRUPT_PIN           0x3d

#define PCI_COMMAND_MASTER          0x4

#define PCI_VENDOR_ID_AMD           0x1022
#define PCI_DEVICE_ID_AMD_LANCE     0x2000

/* supported max number of ether cards */
#define EC_PORT_NR_MAX 2

/* macros for 'mode' */
#define EC_DISABLED 0x0
#define EC_SINK      0x1
#define EC_ENABLED  0x2

/* macros for 'flags' */
#define ECF_EMPTY      0x000
#define ECF_PACK_SEND  0x001
#define ECF_PACK_RECV  0x002
#define ECF_SEND_AVAIL 0x004
#define ECF_READING    0x010
#define ECF_PROMISC     0x040
#define ECF_MULTI      0x080
#define ECF_BROAD       0x100
#define ECF_ENABLED     0x200
#define ECF_STOPPED     0x400

/* === macros for ether cards (our generalized version) === */
#define EC_ISR_RINT      0x0001
#define EC_ISR_WINT      0x0002
#define EC_ISR_RERR      0x0010
#define EC_ISR_WERR      0x0020
#define EC_ISR_ERR       0x0040
#define EC_ISR_RST       0x0100

/* IOVEC */
#define IOVEC_NR 16
typedef struct iovec_dat
{
    iovec_t iod_iovec[IOVEC_NR];
    int iod_iovec_s;
    int iod_proc_nr;
    vir_bytes iod_iovec_addr;
} iovec_dat_t;

#define ETH0_SELECTOR 0x61
#define ETH1_SELECTOR 0x69

/* ===== ethernet card info. ===== */
typedef struct ether_card
{
    /* ##### MINIX style ##### */
    char port_name[sizeof("eth_card#n")];
    int flags;
    int mode;
    int transfer_mode;
    eth_stat_t eth_stat;
    iovec_dat_t read_iovec;
}
```



```
iovec_dat_t write_iovec;
iovec_dat_t tmp_iovec;
vir_bytes write_s;
vir_bytes read_s;
int client;
message sendmsg;

/* ##### device info. ##### */
port_t ec_port;
phys_bytes ec_linmem;
int ec_irq;
int ec_int_pending;
int ec_hook;

int ec_ramsize;
/* PCI */
u8_t ec_pcibus;
u8_t ec_pcidev;
u8_t ec_pcifunc;

/* Addrassing */
ul6_t ec_memseg;
vir_bytes ec_memoff;

ether_addr_t mac_address;
} ether_card_t;

#define DEI_DEFAULT      0x8000
```

```
# Makefile for driver library

# Directories
u = /usr
i = $u/include
s = $i/sys
b = $i/ibm
m = $i/minix

# Programs, flags, etc.
CC = exec cc
CFLAGS = -I$i
LDFLAGS = -i
LIBS = -lsysutil -lsys

OBJECTS = driver.o drvlib.o

all build install: $(OBJECTS)

# $(CC) -c $@ $(LDFLAGS) $(OBJ) $(LIBS)

clean:
    rm -f *.o *.bak

depend:
    /usr/bin/mkdep "$(CC) -E $(CPPFLAGS)" *.c > .depend

# Include generated dependencies.
include .depend
```

```

/* This file contains device independent device driver interface.
 *
 * Changes:
 *   Jul 25, 2005   added SYS_SIG type for signals (Jorrit N. Herder)
 *   Sep 15, 2004   added SYN_ALARM type for timeouts (Jorrit N. Herder)
 *   Jul 23, 2004   removed kernel dependencies (Jorrit N. Herder)
 *   Apr 02, 1992   constructed from AT wini and floppy driver (Kees J. Bot)
 *
 * The drivers support the following operations (using message format m2):
 *
 *   m_type      DEVICE      IO_ENDPT      COUNT      POSITION      ADDRESS
 *   -----
 *   | DEV_OPEN   | device   | proc nr |           |           |           |
 *   |-----|-----|-----|-----|-----|-----|
 *   | DEV_CLOSE  | device   | proc nr |           |           |           |
 *   |-----|-----|-----|-----|-----|-----|
 *   | DEV_READ   | device   | proc nr | bytes    | offset    | buf ptr   |
 *   |-----|-----|-----|-----|-----|-----|
 *   | DEV_WRITE  | device   | proc nr | bytes    | offset    | buf ptr   |
 *   |-----|-----|-----|-----|-----|-----|
 *   | DEV_GATHER | device   | proc nr | iov len  | offset    | iov ptr   |
 *   |-----|-----|-----|-----|-----|-----|
 *   | DEV_SCATTER| device   | proc nr | iov len  | offset    | iov ptr   |
 *   |-----|-----|-----|-----|-----|-----|
 *   | DEV_IOCTL  | device   | proc nr | func code|           | buf ptr   |
 *   |-----|-----|-----|-----|-----|-----|
 *   | CANCEL     | device   | proc nr | r/w      |           |           |
 *   |-----|-----|-----|-----|-----|-----|
 *   | HARD_STOP  |           |         |          |           |           |
 *   |-----|-----|-----|-----|-----|-----|
 *
 * The file contains one entry point:
 *
 *   driver_task:      called by the device dependent task entry
 */

#include "../drivers.h"
#include <sys/ioc_disk.h>
#include "driver.h"

#if (CHIP == INTEL)

#if USE_EXTRA_DMA_BUF && DMA_BUF_SIZE < 2048
/* A bit extra scratch for the Adaptec driver. */
#define BUF_EXTRA      (2048 - DMA_BUF_SIZE)
#else
#define BUF_EXTRA      0
#endif

/* Claim space for variables. */
PRIVATE u8_t buffer[(unsigned) 2 * DMA_BUF_SIZE + BUF_EXTRA];
u8_t *tmp_buf;          /* the DMA buffer eventually */
phys_bytes tmp_phys;    /* phys address of DMA buffer */

#else /* CHIP != INTEL */

/* Claim space for variables. */
u8_t tmp_buf[DMA_BUF_SIZE]; /* the DMA buffer */
phys_bytes tmp_phys;        /* phys address of DMA buffer */

#endif /* CHIP != INTEL */

FORWARD _PROTOTYPE( void init_buffer, (void) );
FORWARD _PROTOTYPE( int do_rdwt, (struct driver *dr, message *mp) );
FORWARD _PROTOTYPE( int do_vrdwt, (struct driver *dr, message *mp) );

int device_caller;

/*=====
 *                               driver_task                               *
 *=====*/
PUBLIC void driver_task(dp)
struct driver *dp;          /* Device dependent entry points. */

```

```

{
/* Main program of any device driver task. */

int r, proc_nr;
message mess;

/* Get a DMA buffer. */
init_buffer();

/* Here is the main loop of the disk task. It waits for a message, carries
 * it out, and sends a reply.
 */
while (TRUE) {

    /* Wait for a request to read or write a disk block. */
    if (receive(ANY, &mess) != OK) continue;

    device_caller = mess.m_source;
    proc_nr = mess.IO_ENDPT;

    /* Now carry out the work. */
    switch(mess.m_type) {
    case DEV_OPEN:      r = (*dp->dr_open)(dp, &mess); break;
    case DEV_CLOSE:    r = (*dp->dr_close)(dp, &mess); break;
    case DEV_IOCTL:    r = (*dp->dr_ioctl)(dp, &mess); break;
    case CANCEL:       r = (*dp->dr_cancel)(dp, &mess); break;
    case DEV_SELECT:   r = (*dp->dr_select)(dp, &mess); break;
    case DEV_READ:
    case DEV_WRITE:    r = do_rdwt(dp, &mess); break;
    case DEV_GATHER:
    case DEV_SCATTER:  r = do_vrdwt(dp, &mess); break;

    case HARD_INT:     /* leftover interrupt or expired timer. */
                      if(dp->dr_hw_int) {
                          (*dp->dr_hw_int)(dp, &mess);
                      }
                      continue;

    case PROC_EVENT:
    case SYS_SIG:      (*dp->dr_signal)(dp, &mess);
                      continue; /* don't reply */
    case SYN_ALARM:    (*dp->dr_alarm)(dp, &mess);
                      continue; /* don't reply */
    case DEV_PING:     notify(mess.m_source);
                      continue;

    default:
        if(dp->dr_other)
            r = (*dp->dr_other)(dp, &mess);
        else
            r = EINVAL;
        break;
    }

    /* Clean up leftover state. */
    (*dp->dr_cleanup)();

    /* Finally, prepare and send the reply message. */
    if (r != EDONTREPLY) {
        mess.m_type = TASK_REPLY;
        mess.REP_ENDPT = proc_nr;
        /* Status is # of bytes transferred or error code. */
        mess.REP_STATUS = r;
        send(device_caller, &mess);
    }
}
}

/*=====
 *                               init_buffer                               *
 *=====*/
PRIVATE void init_buffer()
{
/* Select a buffer that can safely be used for DMA transfers. It may also
 * be used to read partition tables and such. Its absolute address is

```

```

* 'tmp_phys', the normal address is 'tmp_buf'.
*/

#if (CHIP == INTEL)
    unsigned left;

    tmp_buf = buffer;
    sys_umap(SELFS, D, (vir_bytes)buffer, (phys_bytes)sizeof(buffer), &tmp_phys);

    if ((left = dma_bytes_left(tmp_phys)) < DMA_BUF_SIZE) {
        /* First half of buffer crosses a 64K boundary, can't DMA into that */
        tmp_buf += left;
        tmp_phys += left;
    }
#endif /* CHIP == INTEL */
}

/*=====
*
*                               do_rdwt
*=====*/
PRIVATE int do_rdwt(dp, mp)
struct driver *dp;          /* device dependent entry points */
message *mp;                /* pointer to read or write message */
{
    /* Carry out a single read or write request. */
    iovec_t iovecl;
    int r, opcode;
    phys_bytes phys_addr;

    /* Disk address? Address and length of the user buffer? */
    if (mp->COUNT < 0) return(EINVAL);

    /* Check the user buffer. */
    sys_umap(mp->IO_ENDPT, D, (vir_bytes) mp->ADDRESS, mp->COUNT, &phys_addr);
    if (phys_addr == 0) return(EFAULT);

    /* Prepare for I/O. */
    if ((*dp->dr_prepare)(mp->DEVICE) == NIL_DEV) return(ENXIO);

    /* Create a one element scatter/gather vector for the buffer. */
    opcode = mp->m_type == DEV_READ ? DEV_GATHER : DEV_SCATTER;
    iovecl.iov_addr = (vir_bytes) mp->ADDRESS;
    iovecl.iov_size = mp->COUNT;

    /* Transfer bytes from/to the device. */
    r = (*dp->dr_transfer)(mp->IO_ENDPT, opcode, mp->POSITION, &iovecl, 1);

    /* Return the number of bytes transferred or an error code. */
    return(r == OK ? (mp->COUNT - iovecl.iov_size) : r);
}

/*=====
*
*                               do_vrdwt
*=====*/
PRIVATE int do_vrdwt(dp, mp)
struct driver *dp;          /* device dependent entry points */
message *mp;                /* pointer to read or write message */
{
    /* Carry out an device read or write to/from a vector of user addresses.
     * The "user addresses" are assumed to be safe, i.e. FS transferring to/from
     * its own buffers, so they are not checked.
     */
    static iovec_t iovec[NR_IOREQS];
    iovec_t *iov;
    phys_bytes iovec_size;
    unsigned nr_req;
    int r;

    nr_req = mp->COUNT;    /* Length of I/O vector */

#if 0
    if (mp->m_source < 0) {
        /* Called by a task, no need to copy vector. */
        iov = (iovec_t *) mp->ADDRESS;
    }

```

```

    } else
#endif
    {
        /* Copy the vector from the caller to kernel space. */
        if (nr_req > NR_IOREQS) nr_req = NR_IOREQS;
        iovec_size = (phys_bytes) (nr_req * sizeof(iovec[0]));

        if (OK != sys_datacopy(mp->m_source, (vir_bytes) mp->ADDRESS,
                               SELF, (vir_bytes) iovec, iovec_size))
            panic((*dp->dr_name)(), "bad I/O vector by", mp->m_source);
        iov = iovec;
    }

    /* Prepare for I/O. */
    if ((*dp->dr_prepare)(mp->DEVICE) == NIL_DEV) return(ENXIO);

    /* Transfer bytes from/to the device. */
    r = (*dp->dr_transfer)(mp->IO_ENDPT, mp->m_type, mp->POSITION, iov, nr_req);

    /* Copy the I/O vector back to the caller. */
    #if 0
        if (mp->m_source >= 0) {
    #endif
        sys_datacopy(SELF, (vir_bytes) iovec,
                     mp->m_source, (vir_bytes) mp->ADDRESS, iovec_size);
        return(r);
    #endif
}

/*=====
 *                               no_name                               *
 *=====*/
PUBLIC char *no_name()
{
    /* Use this default name if there is no specific name for the device. This was
     * originally done by fetching the name from the task table for this process:
     * "return(tasktab[proc_number(proc_ptr) + NR_TASKS].name);", but currently a
     * real "noname" is returned. Perhaps, some system information service can be
     * queried for a name at a later time.
     */
    static char name[] = "noname";
    return name;
}

/*=====
 *                               do_nop                               *
 *=====*/
PUBLIC int do_nop(dp, mp)
struct driver *dp;
message *mp;
{
    /* Nothing there, or nothing to do. */

    switch (mp->m_type) {
        case DEV_OPEN:      return(ENODEV);
        case DEV_CLOSE:     return(OK);
        case DEV_IOCTL:     return(ENOTTY);
        default:             return(EIO);
    }
}

/*=====
 *                               nop_signal                             *
 *=====*/
PUBLIC void nop_signal(dp, mp)
struct driver *dp;
message *mp;
{
    /* Default action for signal is to ignore. */
}

/*=====
 *                               nop_alarm                             *
 *=====*/
PUBLIC void nop_alarm(dp, mp)

```

```

struct driver *dp;
message *mp;
{
/* Ignore the leftover alarm. */
}

/*=====
*                               nop_prepare                               *
*=====*/
PUBLIC struct device *nop_prepare(device)
{
/* Nothing to prepare for. */
return(NIL_DEV);
}

/*=====
*                               nop_cleanup                               *
*=====*/
PUBLIC void nop_cleanup()
{
/* Nothing to clean up. */
}

/*=====
*                               nop_cancel                               *
*=====*/
PUBLIC int nop_cancel(struct driver *dr, message *m)
{
/* Nothing to do for cancel. */
return(OK);
}

/*=====
*                               nop_select                               *
*=====*/
PUBLIC int nop_select(struct driver *dr, message *m)
{
/* Nothing to do for select. */
return(OK);
}

/*=====
*                               do_diocntl                               *
*=====*/
PUBLIC int do_diocntl(dp, mp)
struct driver *dp;
message *mp;                               /* pointer to ioctl request */
{
/* Carry out a partition setting/getting request. */
struct device *dv;
struct partition entry;
int s;

if (mp->REQUEST != DIOCSETP && mp->REQUEST != DIOCGETP) {
    if(dp->dr_other) {
        return dp->dr_other(dp, mp);
    } else return(ENOTTY);
}

/* Decode the message parameters. */
if ((dv = (*dp->dr_prepare)(mp->DEVICE)) == NIL_DEV) return(ENXIO);

if (mp->REQUEST == DIOCSETP) {
    /* Copy just this one partition table entry. */
    if (OK != (s=sys_datacopy(mp->IO_ENDPT, (vir_bytes) mp->ADDRESS,
        SELF, (vir_bytes) &entry, sizeof(entry))))
        return s;
    dv->dv_base = entry.base;
    dv->dv_size = entry.size;
} else {
    /* Return a partition table entry and the geometry of the drive. */
    entry.base = dv->dv_base;
    entry.size = dv->dv_size;
    (*dp->dr_geometry)(&entry);
}
}

```

```
    if (OK != (s=sys_datacopy(SELF, (vir_bytes) &entry,  
        mp->IO_ENDPT, (vir_bytes) mp->ADDRESS, sizeof(entry))))  
        return s;  
}  
return(OK);  
}
```



```

/* Types and constants shared between the generic and device dependent
 * device driver code.
 */

#define _POSIX_SOURCE      1      /* tell headers to include POSIX stuff */
#define _MINIX             1      /* tell headers to include MINIX stuff */
#define _SYSTEM           1      /* get negative error number in <errno.h> */

/* The following are so basic, all the *.c files get them automatically. */
#include <minix/config.h>      /* MUST be first */
#include <ansi.h>              /* MUST be second */
#include <minix/type.h>
#include <minix/ipc.h>
#include <minix/com.h>
#include <minix/callnr.h>
#include <sys/types.h>
#include <minix/const.h>
#include <minix/syslib.h>
#include <minix/sysutil.h>

#include <string.h>
#include <limits.h>
#include <stddef.h>
#include <errno.h>

#include <minix/partition.h>
#include <minix/u64.h>

/* Info about and entry points into the device dependent code. */
struct driver {
    _PROTOTYPE( char *(*dr_name), (void) );
    _PROTOTYPE( int (*dr_open), (struct driver *dp, message *m_ptr) );
    _PROTOTYPE( int (*dr_close), (struct driver *dp, message *m_ptr) );
    _PROTOTYPE( int (*dr_ioctl), (struct driver *dp, message *m_ptr) );
    _PROTOTYPE( struct device *(*dr_prepare), (int device) );
    _PROTOTYPE( int (*dr_transfer), (int proc_nr, int opcode, off_t position,
                                     iovect_t *iov, unsigned nr_req) );
    _PROTOTYPE( void (*dr_cleanup), (void) );
    _PROTOTYPE( void (*dr_geometry), (struct partition *entry) );
    _PROTOTYPE( void (*dr_signal), (struct driver *dp, message *m_ptr) );
    _PROTOTYPE( void (*dr_alarm), (struct driver *dp, message *m_ptr) );
    _PROTOTYPE( int (*dr_cancel), (struct driver *dp, message *m_ptr) );
    _PROTOTYPE( int (*dr_select), (struct driver *dp, message *m_ptr) );
    _PROTOTYPE( int (*dr_other), (struct driver *dp, message *m_ptr) );
    _PROTOTYPE( int (*dr_hw_int), (struct driver *dp, message *m_ptr) );
};

#if (CHIP == INTEL)

/* Number of bytes you can DMA before hitting a 64K boundary: */
#define dma_bytes_left(phys) \
    ((unsigned) (sizeof(int) == 2 ? 0 : 0x10000) - (unsigned) ((phys) & 0xFFFF))

#endif /* CHIP == INTEL */

/* Base and size of a partition in bytes. */
struct device {
    u64_t dv_base;
    u64_t dv_size;
};

#define NIL_DEV          ((struct device *) 0)

/* Functions defined by driver.c: */
_PROTOTYPE( void driver_task, (struct driver *dr) );
_PROTOTYPE( char *no_name, (void) );
_PROTOTYPE( int do_nop, (struct driver *dp, message *m_ptr) );
_PROTOTYPE( struct device *nop_prepare, (int device) );
_PROTOTYPE( void nop_cleanup, (void) );
_PROTOTYPE( void nop_task, (void) );
_PROTOTYPE( void nop_signal, (struct driver *dp, message *m_ptr) );
_PROTOTYPE( void nop_alarm, (struct driver *dp, message *m_ptr) );
_PROTOTYPE( int nop_cancel, (struct driver *dp, message *m_ptr) );
_PROTOTYPE( int nop_select, (struct driver *dp, message *m_ptr) );

```

```
_PROTOTYPE( int do_dioctl, (struct driver *dp, message *m_ptr) );

/* Parameters for the disk drive. */
#define SECTOR_SIZE      512      /* physical sector size in bytes */
#define SECTOR_SHIFT     9        /* for division */
#define SECTOR_MASK      511      /* and remainder */

/* Size of the DMA buffer buffer in bytes. */
#define USE_EXTRA_DMA_BUF 0        /* usually not needed */
#define DMA_BUF_SIZE      (DMA_SECTORS * SECTOR_SIZE)

#if (CHIP == INTEL)
extern u8_t *tmp_buf;              /* the DMA buffer */
#else
extern u8_t tmp_buf[];            /* the DMA buffer */
#endif
extern phys_bytes tmp_phys;        /* phys address of DMA buffer */
```

```

/* IBM device driver utility functions.                                Author: Kees J. Bot
*                                                                    7 Dec 1995
* Entry point:
* partition: partition a disk to the partition table(s) on it.
*/

#include "driver.h"
#include "drvlib.h"
#include <unistd.h>

/* Extended partition? */
#define ext_part(s)      ((s) == 0x05 || (s) == 0x0F)

FORWARD _PROTOTYPE( void extpartition, (struct driver *dp, int extdev,
                                         unsigned long extbase) );
FORWARD _PROTOTYPE( int get_part_table, (struct driver *dp, int device,
                                         unsigned long offset, struct part_entry *table));
FORWARD _PROTOTYPE( void sort, (struct part_entry *table) );

#ifndef CD_SECTOR_SIZE
#define CD_SECTOR_SIZE 2048
#endif

/*=====
*
* partition
*=====*/
PUBLIC void partition(dp, device, style, atapi)
struct driver *dp;      /* device dependent entry points */
int device;            /* device to partition */
int style;             /* partitioning style: floppy, primary, sub. */
int atapi;             /* atapi device */
{
/* This routine is called on first open to initialize the partition tables
* of a device. It makes sure that each partition falls safely within the
* device's limits. Depending on the partition style we are either making
* floppy partitions, primary partitions or subpartitions. Only primary
* partitions are sorted, because they are shared with other operating
* systems that expect this.
*/
struct part_entry table[NR_PARTITIONS], *pe;
int disk, par;
struct device *dv;
unsigned long base, limit, part_limit;

/* Get the geometry of the device to partition */
if ((dv = (*dp->dr_prepare)(device)) == NIL_DEV
    || cmp64u(dv->dv_size, 0) == 0) return;
base = div64u(dv->dv_base, SECTOR_SIZE);
limit = base + div64u(dv->dv_size, SECTOR_SIZE);

/* Read the partition table for the device. */
if(!get_part_table(dp, device, 0L, table)) {
    return;
}

/* Compute the device number of the first partition. */
switch (style) {
case P_FLOPPY:
    device += MINOR_fd0p0;
    break;
case P_PRIMARY:
    sort(table);          /* sort a primary partition table */
    device += 1;
    break;
case P_SUB:
    disk = device / DEV_PER_DRIVE;
    par = device % DEV_PER_DRIVE - 1;
    device = MINOR_d0p0s0 + (disk * NR_PARTITIONS + par) * NR_PARTITIONS;
}

/* Find an array of devices. */
if ((dv = (*dp->dr_prepare)(device)) == NIL_DEV) return;

/* Set the geometry of the partitions from the partition table. */

```

```

for (par = 0; par < NR_PARTITIONS; par++, dv++) {
    /* Shrink the partition to fit within the device. */
    pe = &table[par];
    part_limit = pe->lowsec + pe->size;
    if (part_limit < pe->lowsec) part_limit = limit;
    if (part_limit > limit) part_limit = limit;
    if (pe->lowsec < base) pe->lowsec = base;
    if (part_limit < pe->lowsec) part_limit = pe->lowsec;

    dv->dv_base = mul64u(pe->lowsec, SECTOR_SIZE);
    dv->dv_size = mul64u(part_limit - pe->lowsec, SECTOR_SIZE);

    if (style == P_PRIMARY) {
        /* Each Minix primary partition can be subpartitioned. */
        if (pe->sysind == MINIX_PART)
            partition(dp, device + par, P_SUB, atapi);

        /* An extended partition has logical partitions. */
        if (ext_part(pe->sysind))
            extpartition(dp, device + par, pe->lowsec);
    }
}
}

/*=====
 *                               extpartition                               *
 *=====*/
PRIVATE void extpartition(dp, extdev, extbase)
struct driver *dp;      /* device dependent entry points */
int extdev;             /* extended partition to scan */
unsigned long extbase;  /* sector offset of the base extended partition */
{
    /* Extended partitions cannot be ignored alas, because people like to move
     * files to and from DOS partitions. Avoid reading this code, it's no fun.
     */
    struct part_entry table[NR_PARTITIONS], *pe;
    int subdev, disk, par;
    struct device *dv;
    unsigned long offset, nextoffset;

    disk = extdev / DEV_PER_DRIVE;
    par = extdev % DEV_PER_DRIVE - 1;
    subdev = MINOR_d0p0s0 + (disk * NR_PARTITIONS + par) * NR_PARTITIONS;

    offset = 0;
    do {
        if (!get_part_table(dp, extdev, offset, table)) return;
        sort(table);

        /* The table should contain one logical partition and optionally
         * another extended partition. (It's a linked list.)
         */
        nextoffset = 0;
        for (par = 0; par < NR_PARTITIONS; par++) {
            pe = &table[par];
            if (ext_part(pe->sysind)) {
                nextoffset = pe->lowsec;
            } else
            if (pe->sysind != NO_PART) {
                if ((dv = (*dp->dr_prepare)(subdev)) == NIL_DEV) return;

                dv->dv_base = mul64u(extbase + offset + pe->lowsec,
                                     SECTOR_SIZE);
                dv->dv_size = mul64u(pe->size, SECTOR_SIZE);

                /* Out of devices? */
                if (++subdev % NR_PARTITIONS == 0) return;
            }
        }
    } while ((offset = nextoffset) != 0);
}

/*=====
 *                               get_part_table                               *
 *=====*/

```

```

*=====*/
PRIVATE int get_part_table(dp, device, offset, table)
struct driver *dp;
int device;
unsigned long offset;          /* sector offset to the table */
struct part_entry *table;      /* four entries */
{
/* Read the partition table for the device, return true iff there were no
 * errors.
 */
    iovec_t iovecl;
    off_t position;
    static unsigned char partbuf[CD_SECTOR_SIZE];

    position = offset << SECTOR_SHIFT;
    iovecl.iov_addr = (vir_bytes) partbuf;
    iovecl.iov_size = CD_SECTOR_SIZE;
    if ((*dp->dr_prepare)(device) != NIL_DEV) {
        (void) (*dp->dr_transfer)(SELF, DEV_GATHER, position, &iovecl, 1);
    }
    if (iovecl.iov_size != 0) {
        return 0;
    }
    if (partbuf[510] != 0x55 || partbuf[511] != 0xAA) {
        /* Invalid partition table. */
        return 0;
    }
    memcpy(table, (partbuf + PART_TABLE_OFF), NR_PARTITIONS * sizeof(table[0]));
    return 1;
}

/*=====*
 *                                     sort                                     *
 *=====*/
PRIVATE void sort(table)
struct part_entry *table;
{
/* Sort a partition table. */
    struct part_entry *pe, tmp;
    int n = NR_PARTITIONS;

    do {
        for (pe = table; pe < table + NR_PARTITIONS-1; pe++) {
            if (pe[0].sysind == NO_PART
                || (pe[0].lowsec > pe[1].lowsec
                    && pe[1].sysind != NO_PART)) {
                tmp = pe[0]; pe[0] = pe[1]; pe[1] = tmp;
            }
        }
    } while (--n > 0);
}

```

```
/* IBM device driver definitions
 *
 */
```

Author: Kees J. Bot  
7 Dec 1995

```
#include <ibm/partition.h>
```

```
_PROTOTYPE( void partition, (struct driver *dr, int device, int style, int atapi) );
```

```
/* BIOS parameter table layout. */
```

```
#define bp_cylinders(t)      (* (u16_t *) (&(t)[0]))
#define bp_heads(t)         (* (u8_t *)  (&(t)[2]))
#define bp_reduced_wr(t)    (* (u16_t *) (&(t)[3]))
#define bp_precomp(t)       (* (u16_t *) (&(t)[5]))
#define bp_max_ecc(t)        (* (u8_t *)  (&(t)[7]))
#define bp_ctlbyte(t)       (* (u8_t *)  (&(t)[8]))
#define bp_landingzone(t)   (* (u16_t *) (&(t)[12]))
#define bp_sectors(t)       (* (u8_t *)  (&(t)[14]))
```

```
/* Miscellaneous. */
```

```
#define DEV_PER_DRIVE      (1 + NR_PARTITIONS)
#define MINOR_t0           64
#define MINOR_r0           120
#define MINOR_d0p0s0       128
#define MINOR_fd0p0        (28<<2)
#define P_FLOPPY           0
#define P_PRIMARY          1
#define P_SUB               2
```

```
# Makefile for log driver
DRIVER = log

# directories
u = /usr
i = $u/include
s = $i/sys
m = $i/minix
d = ..

# programs, flags, etc.
MAKE = exec make
CC = exec cc
CFLAGS = -I$i
LDFLAGS = -i
LIBS = -lsys -lsysutil

OBJ = log.o diag.o kputc.o
LIBDRIVER = $d/libdriver/driver.o

# build local binary
all build: $(DRIVER)
$(DRIVER): $(OBJ) $(LIBDRIVER)
    $(CC) -o $@ $(LDFLAGS) $(OBJ) $(LIBDRIVER) $(LIBS)
    install -S 4kb $(DRIVER)

$(LIBDRIVER):
    cd $d/libdriver && $(MAKE)

# install with other drivers
install: $(DRIVER)
    install -o root -cs $? /sbin/$(DRIVER)

# clean up local files
clean:
    rm -f $(DRIVER) *.o *.bak

depend:
    /usr/bin/mkdep "$(CC) -E $(CPPFLAGS)" *.c ../libdriver/*.c > .depend

# Include generated dependencies.
include .depend
```

```

/* This file handle diagnostic output that is directly sent to the LOG driver.
 * This output can either be a kernel message (announced through a SYS_EVENT
 * with a SIGKMESS in the signal set) or output from another system process
 * (announced through a DIAGNOSTICS message).
 *
 * Changes:
 *      21 July 2005:   Created   (Jorrit N. Herder)
 */

#include <stdio.h>
#include <fcntl.h>

#include "log.h"
#include "../kernel/const.h"
#include "../kernel/config.h"
#include "../kernel/type.h"

/*=====
 *                               do_new_kmess                               *
 *=====*/
PUBLIC int do_new_kmess(m)
message *m;                               /* notification message */
{
    /* Notification for a new kernel message. */
    struct kmessages kmess;               /* entire kmess structure */
    char print_buf[KMESS_BUF_SIZE];      /* copy new message here */
    static int prev_next = 0;
    int bytes;
    int i, r;

    if (m->m_source == TTY_PROC_NR)
    {
        message mess;

        /* Ask TTY driver for log output */
        mess.GETKM_PTR= (char *) &kmess;
        mess.m_type = GET_KMESS;
        r= sendrec(TTY_PROC_NR, &mess);
        if (r == OK) r= mess.m_type;
        if (r != OK)
        {
            report("LOG", "couldn't get copy of kmessages from TTY", r);
            return EDONTREPLY;
        }
    }
    else
    {
        /* Try to get a fresh copy of the buffer with kernel messages. */
        if ((r=sys_getkmessages(&kmess)) != OK) {
            report("LOG", "couldn't get copy of kmessages", r);
            return EDONTREPLY;
        }
    }

    /* Print only the new part. Determine how many new bytes there are with
     * help of the current and previous 'next' index. Note that the kernel
     * buffer is circular. This works fine if less then KMESS_BUF_SIZE bytes
     * is new data; else we miss % KMESS_BUF_SIZE here.
     * Check for size being positive, the buffer might as well be emptied!
     */
    if (kmess.km_size > 0) {
        bytes = ((kmess.km_next + KMESS_BUF_SIZE) - prev_next) % KMESS_BUF_SIZE;
        r=prev_next;                               /* start at previous old */
        i=0;
        while (bytes > 0) {
            print_buf[i] = kmess.km_buf[(r%KMESS_BUF_SIZE)];
            bytes --;
            r ++;
            i ++;
        }
        /* Now terminate the new message and save it in the log. */
        print_buf[i] = 0;
        log_append(print_buf, i);
    }
}

```



```
/* Almost done, store 'next' so that we can determine what part of the
 * kernel messages buffer to print next time a notification arrives.
 */
prev_next = kmess.km_next;
return EDONTREPLY;
}

/*=====
 *                               do_diagnostics                               *
 *=====*/
PUBLIC int do_diagnostics(message *m)
{
/* The LOG server handles all diagnostic messages from servers and device
 * drivers. It forwards the message to the TTY driver to display it to the
 * user. It also saves a copy in a local buffer so that messages can be
 * reviewed at a later time.
 */
int proc_nr_e;
vir_bytes src;
int count;
char c;
int i = 0;
static char diagbuf[10240];

/* Change SELF to actual process number. */
if ((proc_nr_e = m->DIAG_ENDPT) == SELF)
    m->DIAG_ENDPT = proc_nr_e = m->m_source;

/* Now also make a copy for the private buffer at the LOG server, so
 * that the messages can be reviewed at a later time.
 */
src = (vir_bytes) m->DIAG_PRINT_BUF;
count = m->DIAG_BUF_COUNT;
while (count > 0 && i < sizeof(diagbuf)-1) {
    if (sys_datacopy(proc_nr_e, src, SELF, (vir_bytes) &c, 1) != OK)
        break; /* stop copying on error */
    src++;
    count--;
    diagbuf[i++] = c;
}
log_append(diagbuf, i);

return OK;
}
```

```
/* A server must occasionally print some message. It uses a simple version of
 * printf() found in the system library that calls putk() to output characters.
 * The LOG driver cannot use the regular putk(). Hence, it uses a special
 * version of putk() that directly sends to the TTY task.
 *
 * Changes:
 *      21 July 2005:    Created (Jorrit N. Herder)
 */
```

```
#include "log.h"
```

```
/*=====
 *                               kputc                               *
 *=====*/
void kputc(c)
int c;
{
    /* Accumulate another character. If 0 or buffer full, print it. */
    static int buf_count;      /* # characters in the buffer */
    static char print_buf[80]; /* output is buffered here */
    message m;

    if ((c == 0 && buf_count > 0) || buf_count == sizeof(print_buf)) {
        m.DIAG_BUF_COUNT = buf_count;
        m.DIAG_PRINT_BUF = print_buf;
        m.DIAG_ENDPT = SELF;
        m.m_type = DIAGNOSTICS;      /* request TTY to output this buffer */
        _sendrec(TTY_PROC_NR, &m); /* if it fails, we give up */
        buf_count = 0;              /* clear buffer for next batch */
    }
    if (c != 0) {
        print_buf[buf_count++] = c;
    }
}
```

```

/* This file contains a driver for:
 *    /dev/klog          - system log device
 *
 * Changes:
 *    21 July 2005      - Support for diagnostic messages (Jorrit N. Herder)
 *    7 July 2005      - Created (Ben Gras)
 */

#include "log.h"
#include <sys/time.h>
#include <sys/select.h>
#include "../kernel/const.h"
#include "../kernel/type.h"

#define LOG_DEBUG          0          /* enable/ disable debugging */

#define NR_DEVS            1          /* number of minor devices */
#define MINOR_KLOG         0          /* /dev/klog */

#define LOGINC(n, i)      do { (n) = (((n) + (i)) % LOG_SIZE); } while(0)

PUBLIC struct logdevice logdevices[NR_DEVS];
PRIVATE struct device log_geom[NR_DEVS];          /* base and size of devices */
PRIVATE int log_device = -1;                      /* current device */

FORWARD _PROTOTYPE( char *log_name, (void) );
FORWARD _PROTOTYPE( struct device *log_prepare, (int device) );
FORWARD _PROTOTYPE( int log_transfer, (int proc_nr, int opcode, off_t position,
                                     iovect_t *iov, unsigned nr_req) );
FORWARD _PROTOTYPE( int log_do_open, (struct driver *dp, message *m_ptr) );
FORWARD _PROTOTYPE( int log_cancel, (struct driver *dp, message *m_ptr) );
FORWARD _PROTOTYPE( int log_select, (struct driver *dp, message *m_ptr) );
FORWARD _PROTOTYPE( void log_signal, (struct driver *dp, message *m_ptr) );
FORWARD _PROTOTYPE( int log_other, (struct driver *dp, message *m_ptr) );
FORWARD _PROTOTYPE( void log_geometry, (struct partition *entry) );
FORWARD _PROTOTYPE( int subread, (struct logdevice *log, int count, int proc_nr, vir_byte
s user_vir) );

/* Entry points to this driver. */
PRIVATE struct driver log_dtab = {
    log_name,          /* current device's name */
    log_do_open,       /* open or mount */
    do_nop,            /* nothing on a close */
    do_nop,            /* ioctl nop */
    log_prepare,       /* prepare for I/O on a given minor device */
    log_transfer,      /* do the I/O */
    nop_cleanup,       /* no need to clean up */
    log_geometry,      /* geometry */
    log_signal,        /* handle system signal */
    nop_alarm,         /* no alarm */
    log_cancel,        /* CANCEL request */
    log_select,        /* DEV_SELECT request */
    log_other,         /* Unrecognized messages */
    NULL               /* HW int */
};

extern int device_caller;

/*=====
 *                               main                               *
 *=====*/
PUBLIC int main(void)
{
    int i;
    for(i = 0; i < NR_DEVS; i++) {
        log_geom[i].dv_size = cvul64(LOG_SIZE);
        log_geom[i].dv_base = cvul64((long)logdevices[i].log_buffer);
        logdevices[i].log_size = logdevices[i].log_read =
            logdevices[i].log_write =
            logdevices[i].log_select_alerted =
            logdevices[i].log_selected =
            logdevices[i].log_select_ready_ops = 0;
        logdevices[i].log_proc_nr = 0;
        logdevices[i].log_revive_alerted = 0;
    }
}

```

```

    }
    driver_task(&log_dtab);
    return(OK);
}

/*=====
 *                               log_name                               *
 *=====*/
PRIVATE char *log_name()
{
    /* Return a name for the current device. */
    static char name[] = "log";
    return name;
}

/*=====
 *                               log_prepare                           *
 *=====*/
PRIVATE struct device *log_prepare(device)
int device;
{
    /* Prepare for I/O on a device: check if the minor device number is ok. */

    if (device < 0 || device >= NR_DEVS) return(NIL_DEV);
    log_device = device;

    return(&log_geom[device]);
}

/*=====
 *                               subwrite                             *
 *=====*/
PRIVATE int
subwrite(struct logdevice *log, int count, int proc_nr, vir_bytes user_vir)
{
    char *buf;
    int r;
    if (log->log_write + count > LOG_SIZE)
        count = LOG_SIZE - log->log_write;
    buf = log->log_buffer + log->log_write;

    if(proc_nr == SELF) {
        memcpy(buf, (char *) user_vir, count);
    }
    else {
        if((r=sys_vircopy(proc_nr,D,user_vir, SELF,D,(int)buf, count)) != OK)
            return r;
    }

    LOGINC(log->log_write, count);
    log->log_size += count;

    if(log->log_size > LOG_SIZE) {
        int overflow;
        overflow = log->log_size - LOG_SIZE;
        log->log_size -= overflow;
        LOGINC(log->log_read, overflow);
    }

    if(log->log_size > 0 && log->log_proc_nr && !log->log_revive_alerted) {
        /* Someone who was suspended on read can now
         * be revived.
         */
        log->log_status = subread(log, log->log_iosize,
                                log->log_proc_nr, log->log_user_vir);
        notify(log->log_source);
        log->log_revive_alerted = 1;
    }

    if(log->log_size > 0)
        log->log_select_ready_ops |= SEL_RD;

    if(log->log_size > 0 && log->log_selected &&
        !(log->log_select_alerted)) {

```

```

        /* Someone(s) who was/were select()ing can now
        * be awoken. If there was a blocking read (above),
        * this can only happen if the blocking read didn't
        * swallow all the data (log_size > 0).
        */
        if(log->log_selected & SEL_RD) {
            notify(log->log_select_proc);
            log->log_select_alerted = 1;
#ifdef LOG_DEBUG
            printf("log notified %d\n", log->log_select_proc);
#endif
        }

        return count;
    }

/*=====
 *                                log_append                                *
 *=====*/
PUBLIC void
log_append(char *buf, int count)
{
    int w = 0, skip = 0;

    if(count < 1) return;
    if(count > LOG_SIZE) skip = count - LOG_SIZE;
    count -= skip;
    buf += skip;
    w = subwrite(&logdevices[0], count, SELF, (vir_bytes) buf);

    if(w > 0 && w < count)
        subwrite(&logdevices[0], count-w, SELF, (vir_bytes) buf+w);
    return;
}

/*=====
 *                                subread                                *
 *=====*/
PRIVATE int
subread(struct logdevice *log, int count, int proc_nr, vir_bytes user_vir)
{
    char *buf;
    int r;
    if (count > log->log_size)
        count = log->log_size;
    if (log->log_read + count > LOG_SIZE)
        count = LOG_SIZE - log->log_read;

    buf = log->log_buffer + log->log_read;
    if((r=sys_vircopy(SELF,D,(int)buf,proc_nr,D,user_vir, count)) != OK)
        return r;

    LOGINC(log->log_read, count);
    log->log_size -= count;

    return count;
}

/*=====
 *                                log_transfer                                *
 *=====*/
PRIVATE int log_transfer(proc_nr, opcode, position, iov, nr_req)
int proc_nr;          /* process doing the request */
int opcode;           /* DEV_GATHER or DEV_SCATTER */
off_t position;       /* offset on device to read or write */
iovec_t *iov;         /* pointer to read or write request vector */
unsigned nr_req;      /* length of request vector */
{
    /* Read or write one the driver's minor devices. */
    unsigned count;
    vir_bytes user_vir;
    struct device *dv;
    unsigned long dv_size;

```

```

int accumulated_read = 0;
struct logdevice *log;
static int f;

if(log_device < 0 || log_device >= NR_DEVS)
    return EIO;

/* Get minor device number and check for /dev/null. */
dv = &log_geom[log_device];
dv_size = cv64ul(dv->dv_size);
log = &logdevices[log_device];

while (nr_req > 0) {
    /* How much to transfer and where to / from. */
    count = iov->iov_size;
    user_vir = iov->iov_addr;

    switch (log_device) {

    case MINOR_KLOG:
        if (opcode == DEV_GATHER) {
            if (log->log_proc_nr || count < 1) {
                /* There's already someone hanging to read, or
                 * no real I/O requested.
                 */
                return(OK);
            }

            if (!log->log_size) {
                if(accumulated_read)
                    return OK;
                /* No data available; let caller block. */
                log->log_proc_nr = proc_nr;
                log->log_iosize = count;
                log->log_user_vir = user_vir;
                log->log_revive_alerted = 0;

                /* Device_caller is a global in drivers library. */
                log->log_source = device_caller;

                #if LOG_DEBUG
                printf("blocked %d (%d)\n",
                    log->log_source, log->log_proc_nr);
                #endif

                return(SUSPEND);
            }
            count = subread(log, count, proc_nr, user_vir);
            if(count < 0) {
                return count;
            }
            accumulated_read += count;
        } else {
            count = subwrite(log, count, proc_nr, user_vir);
            if(count < 0)
                return count;
        }
        break;
    /* Unknown (illegal) minor device. */
    default:
        return(EINVAL);
    }

    /* Book the number of bytes transferred. */
    iov->iov_addr += count;
    if ((iov->iov_size -= count) == 0) { iov++; nr_req--; }
}
return(OK);
}

/*=====
 *                               log_do_open                               *
 *=====*/
PRIVATE int log_do_open(dp, m_ptr)
struct driver *dp;
message *m_ptr;

```

```

{
    if (log_prepare(m_ptr->DEVICE) == NIL_DEV) return(ENXIO);
    return(OK);
}

/*=====
 *                               log_geometry                               *
 *=====*/
PRIVATE void log_geometry(entry)
struct partition *entry;
{
    /* take a page from the fake memory device geometry */
    entry->heads = 64;
    entry->sectors = 32;
    entry->cylinders = div64u(log_geom[log_device].dv_size, SECTOR_SIZE) /
        (entry->heads * entry->sectors);
}

/*=====
 *                               log_cancel                               *
 *=====*/
PRIVATE int log_cancel(dp, m_ptr)
struct driver *dp;
message *m_ptr;
{
    int d;
    d = m_ptr->TTY_LINE;
    if(d < 0 || d >= NR_DEVS)
        return EINVAL;
    logdevices[d].log_proc_nr = 0;
    logdevices[d].log_revive_alerted = 0;
    return(OK);
}

/*=====
 *                               do_status                               *
 *=====*/
PRIVATE void do_status(message *m_ptr)
{
    int d;
    message m;

    /* Caller has requested pending status information, which currently
     * can be pending available select()s, or REVIVE events. One message
     * is returned for every event, or DEV_NO_STATUS if no (more) events
     * are to be returned.
     */

    for(d = 0; d < NR_DEVS; d++) {
        /* Check for revive callback. */
        if(logdevices[d].log_proc_nr && logdevices[d].log_revive_alerted
            && logdevices[d].log_source == m_ptr->m_source) {
            m.m_type = DEV_REVIVE;
            m.REP_ENDPT = logdevices[d].log_proc_nr;
            m.REP_STATUS = logdevices[d].log_status;
            send(m_ptr->m_source, &m);
            logdevices[d].log_proc_nr = 0;
            logdevices[d].log_revive_alerted = 0;
        }

        /* Check for select callback. */
        if(logdevices[d].log_selected && logdevices[d].log_select_proc == m_ptr->
m_source
            && logdevices[d].log_select_alerted) {
            m.m_type = DEV_IO_READY;
            m.DEV_SEL_OPS = logdevices[d].log_select_ready_ops;
            m.DEV_MINOR = d;
        }

        #if LOG_DEBUG
        printf("revived %d with %d bytes\n",
            m.REP_ENDPT, m.REP_STATUS);
        #endif
        return;
    }

    #if LOG_DEBUG
    printf("select sending sent\n");
    #endif
}

```

```

#endif
                send(m_ptr->m_source, &m);
                logdevices[d].log_selected &= ~logdevices[d].log_select_ready_ops
;
                logdevices[d].log_select_alerted = 0;
#if LOG_DEBUG
                printf("select send sent\n");
#endif
                return;
        }

        /* No event found. */
        m.m_type = DEV_NO_STATUS;
        send(m_ptr->m_source, &m);

        return;
}

/*=====
 *                                log_signal                                *
 *=====*/
PRIVATE void log_signal(dp, m_ptr)
struct driver *dp;
message *m_ptr;
{
    sigset_t sigset = m_ptr->NOTIFY_ARG;
    if (sigismember(&sigset, SIGKMESS)) {
        do_new_kmess(m_ptr);
    }
}

/*=====
 *                                log_other                                *
 *=====*/
PRIVATE int log_other(dp, m_ptr)
struct driver *dp;
message *m_ptr;
{
    int r;

    /* This function gets messages that the generic driver doesn't
     * understand.
     */
    switch(m_ptr->m_type) {
    case DIAGNOSTICS: {
        r = do_diagnostics(m_ptr);
        break;
    }
    case DEV_STATUS: {
        do_status(m_ptr);
        r = EDONTREPLY;
        break;
    }
    case NOTIFY_FROM(TTY_PROC_NR):
        do_new_kmess(m_ptr);
        r = EDONTREPLY;
        break;
    default:
        r = EINVAL;
        break;
    }
    return r;
}

/*=====
 *                                log_select                                *
 *=====*/
PRIVATE int log_select(dp, m_ptr)
struct driver *dp;
message *m_ptr;
{
    int d, ready_ops = 0, ops = 0;

```



```
d = m_ptr->TTY_LINE;
if(d < 0 || d >= NR_DEVS) {
#ifdef LOG_DEBUG
    printf("line %d? EINVAL\n", d);
#endif
    return EINVAL;
}

ops = m_ptr->IO_ENDPT & (SEL_RD|SEL_WR|SEL_ERR);

/* Read blocks when there is no log. */
if((m_ptr->IO_ENDPT & SEL_RD) && logdevices[d].log_size > 0) {
#ifdef LOG_DEBUG
    printf("log can read; size %d\n", logdevices[d].log_size);
#endif
    ready_ops |= SEL_RD; /* writes never block */
}

/* Write never blocks. */
if(m_ptr->IO_ENDPT & SEL_WR) ready_ops |= SEL_WR;

/* Enable select callback if no operations were
 * ready to go, but operations were requested,
 * and notify was enabled.
 */
if((m_ptr->IO_ENDPT & SEL_NOTIFY) && ops && !ready_ops) {
    logdevices[d].log_selected |= ops;
    logdevices[d].log_select_proc = m_ptr->m_source;
#ifdef LOG_DEBUG
    printf("log setting selector.\n");
#endif
}

#ifdef LOG_DEBUG
    printf("log returning ops %d\n", ready_ops);
#endif
return(ready_ops);
}
```

```
/* Includes. */
#include "../drivers.h"
#include "../libdriver/driver.h"
#include <minix/type.h>
#include <minix/const.h>
#include <minix/com.h>
#include <sys/types.h>
#include <minix/ipc.h>

/* Constants and types. */

#define LOG_SIZE      (50*1024)
#define SUSPENDABLE  1

struct logdevice {
    char log_buffer[LOG_SIZE];
    int   log_size,      /* no. of bytes in log buffer */
         log_read,      /* read mark */
         log_write;     /* write mark */
#if SUSPENDABLE
    int log_proc_nr,
        log_source,
        log_iosize,
        log_revive_alerted,
        log_status;     /* proc that is blocking on read */
    vir_bytes log_user_vir;
#endif
    int   log_selected, log_select_proc,
         log_select_alerted, log_select_ready_ops;
};

/* Function prototypes. */
_PROTOTYPE( void kputc, (int c) ) ;
_PROTOTYPE( int do_new_kmess, (message *m) ) ;
_PROTOTYPE( int do_diagnostics, (message *m) ) ;
_PROTOTYPE( void log_append, (char *buf, int len) ) ;
```

```
# Makefile for memory driver (MEMORY)
DRIVER = memory

# directories
u = /usr
i = $u/include
s = $i/sys
m = $i/minix
b = $i/ibm
d = ..

# programs, flags, etc.
MAKE = exec make
CC = exec cc
CFLAGS = -I$i
LDFLAGS = -i
LIBS = -lsys -lsysutil

# imgrd.s is the ACK assembler version of the ramdisk. For more portability,
# use the C version imgrd.c. However, the C compiler takes too much memory
# compiling imgrd.c.
IMGRD=imgrd_s.o
#IMGRD=imgrd.c

OBJ = memory.o allocmem.o $(IMGRD)
LIBDRIVER = $d/libdriver/driver.o

# build local binary
all build: $(DRIVER)

$(DRIVER): ramdisk_image $(OBJ) $(LIBDRIVER)
$(CC) -o $@ $(LDFLAGS) $(OBJ) $(LIBDRIVER) $(LIBS)
install -S 8k $(DRIVER)

imgrd.o: ramdisk/image.c
imgrd_s.o: ramdisk/image.s

$(LIBDRIVER):
cd $d/libdriver && $(MAKE)

ramdisk_image:
cd ramdisk && make

# install with other drivers
install: /usr/sbin/$(DRIVER)
/usr/sbin/$(DRIVER): $(DRIVER)
install -o root -cs $? $@

# clean up local files
clean:
rm -f $(DRIVER) *.o *.bak
cd ramdisk && make clean

depend:
/usr/bin/mkdep "$(CC) -E $(CPPFLAGS)" memory.c ../libdriver/*.c > .depend
cd ramdisk && make depend

# Include generated dependencies.
include .depend
```

```
/*  
Ramdisk that is part of the image  
*/  
  
#include <stddef.h>  
  
#include "local.h"  
  
unsigned char imgrd[]=  
{  
#include "ramdisk/image.c"  
};  
  
size_t imgrd_size= sizeof(imgrd);
```

```
#
.sect .text; .sect .rom; .sect .data

! export symbols
.define _imgrd, _imgrd_size

.sect .data
_imgrd:
0:
#include "ramdisk/image.s"
1:

! Use local labels to compute the size of _imgrd.
_imgrd_size:
.data4 [1b] - [0b]
```

```
/*  
local defines and declarations  
*/  
  
extern unsigned char imgrd[];  
extern size_t imgrd_size;
```

```

/* This file contains the device dependent part of the drivers for the
 * following special files:
 *    /dev/ram      - RAM disk
 *    /dev/mem      - absolute memory
 *    /dev/kmem     - kernel virtual memory
 *    /dev/null     - null device (data sink)
 *    /dev/boot     - boot device loaded from boot image
 *    /dev/zero     - null byte stream generator
 *
 * Changes:
 *    Apr 29, 2005    added null byte generator (Jorrit N. Herder)
 *    Apr 09, 2005    added support for boot device (Jorrit N. Herder)
 *    Jul 26, 2004    moved RAM driver to user-space (Jorrit N. Herder)
 *    Apr 20, 1992    device dependent/independent split (Kees J. Bot)
 */

#include "../drivers.h"
#include "../libdriver/driver.h"
#include <sys/ioc_memory.h>
#include "../kernel/const.h"
#include "../kernel/config.h"
#include "../kernel/type.h"

#include <sys/vm.h>

#include "assert.h"

#include "local.h"

#define NR_DEVS          7                /* number of minor devices */

PRIVATE struct device m_geom[NR_DEVS]; /* base and size of each device */
PRIVATE int m_seg[NR_DEVS];           /* segment index of each device */
PRIVATE int m_device;                 /* current device */
PRIVATE struct kinfo kinfo;           /* kernel information */
PRIVATE struct machine machine;        /* machine information */

extern int errno;                     /* error number for PM calls */

FORWARD _PROTOTYPE( char *m_name, (void) );
FORWARD _PROTOTYPE( struct device *m_prepare, (int device) );
FORWARD _PROTOTYPE( int m_transfer, (int proc_nr, int opcode, off_t position,
                                     iovec_t *iov, unsigned nr_req) );
FORWARD _PROTOTYPE( int m_do_open, (struct driver *dp, message *m_ptr) );
FORWARD _PROTOTYPE( void m_init, (void) );
FORWARD _PROTOTYPE( int m_ioctl, (struct driver *dp, message *m_ptr) );
FORWARD _PROTOTYPE( void m_geometry, (struct partition *entry) );

/* Entry points to this driver. */
PRIVATE struct driver m_dtab = {
    m_name,          /* current device's name */
    m_do_open,       /* open or mount */
    do_nop,          /* nothing on a close */
    m_ioctl,         /* specify ram disk geometry */
    m_prepare,       /* prepare for I/O on a given minor device */
    m_transfer,      /* do the I/O */
    nop_cleanup,     /* no need to clean up */
    m_geometry,      /* memory device "geometry" */
    nop_signal,      /* system signals */
    nop_alarm,
    nop_cancel,
    nop_select,
    NULL,
    NULL
};

/* Buffer for the /dev/zero null byte feed. */
#define ZERO_BUF_SIZE    1024
PRIVATE char dev_zero[ZERO_BUF_SIZE];

#define click_to_round_k(n) \
    (((unsigned) (((unsigned long) (n) << CLICK_SHIFT) + 512) / 1024))

/*=====

```

```

*                               main                               *
*=====*/
PUBLIC int main(void)
{
/* Main program. Initialize the memory driver and start the main loop. */
    struct sigaction sa;

    sa.sa_handler = SIG_MESS;
    sigemptyset(&sa.sa_mask);
    sa.sa_flags = 0;
    if (sigaction(SIGTERM,&sa,NULL)<0) panic("MEM","sigaction failed", errno);

    m_init();
    driver_task(&m_dtab);
    return(OK);
}

/*=====*
*                               m_name                             *
*=====*/
PRIVATE char *m_name()
{
/* Return a name for the current device. */
    static char name[] = "memory";
    return name;
}

/*=====*
*                               m_prepare                           *
*=====*/
PRIVATE struct device *m_prepare(device)
int device;
{
/* Prepare for I/O on a device: check if the minor device number is ok. */
    if (device < 0 || device >= NR_DEVS) return(NIL_DEV);
    m_device = device;

    return(&m_geom[device]);
}

/*=====*
*                               m_transfer                          *
*=====*/
PRIVATE int m_transfer(proc_nr, opcode, position, iov, nr_req)
int proc_nr;           /* process doing the request */
int opcode;            /* DEV_GATHER or DEV_SCATTER */
off_t position;        /* offset on device to read or write */
iovec_t *iov;          /* pointer to read or write request vector */
unsigned nr_req;        /* length of request vector */
{
/* Read or write one the driver's minor devices. */
    phys_bytes mem_phys;
    int seg;
    unsigned count, left, chunk;
    vir_bytes user_vir;
    struct device *dv;
    unsigned long dv_size;
    int s;

    /* Get minor device number and check for /dev/null. */
    dv = &m_geom[m_device];
    dv_size = cv64ul(dv->dv_size);

    while (nr_req > 0) {

        /* How much to transfer and where to / from. */
        count = iov->iov_size;
        user_vir = iov->iov_addr;

        switch (m_device) {

            /* No copying; ignore request. */
            case NIL_DEV:
                if (opcode == DEV_GATHER) return(OK);           /* always at EOF */

```



```

        break;

/* Virtual copying. For RAM disk, kernel memory and boot device. */
case RAM_DEV:
case KMEM_DEV:
case BOOT_DEV:
    if (position >= dv_size) return(OK);          /* check for EOF */
    if (position + count > dv_size) count = dv_size - position;
    seg = m_seg[m_device];

    if (opcode == DEV_GATHER) {                  /* copy actual data */
        sys_vircopy(SELF, seg, position, proc_nr, D, user_vir, count);
    } else {
        sys_vircopy(proc_nr, D, user_vir, SELF, seg, position, count);
    }
    break;

/* Physical copying. Only used to access entire memory. */
case MEM_DEV:
    if (position >= dv_size) return(OK);          /* check for EOF */
    if (position + count > dv_size) count = dv_size - position;
    mem_phys = cv64ul(dv->dv_base) + position;

    if (opcode == DEV_GATHER) {                  /* copy data */
        sys_physcopy(NONE, PHYS_SEG, mem_phys,
            proc_nr, D, user_vir, count);
    } else {
        sys_physcopy(proc_nr, D, user_vir,
            NONE, PHYS_SEG, mem_phys, count);
    }
    break;

/* Null byte stream generator. */
case ZERO_DEV:
    if (opcode == DEV_GATHER) {
        left = count;
        while (left > 0) {
            chunk = (left > ZERO_BUF_SIZE) ? ZERO_BUF_SIZE : left;
            if (OK != (s=sys_vircopy(SELF, D, (vir_bytes) dev_zero,
                proc_nr, D, user_vir, chunk)))
                report("MEM", "sys_vircopy failed", s);
            left -= chunk;
            user_vir += chunk;
        }
    }
    break;

case IMGRD_DEV:
    if (position >= dv_size) return(OK);          /* check for EOF */
    if (position + count > dv_size) count = dv_size - position;

    if (opcode == DEV_GATHER) {                  /* copy actual data */
        sys_vircopy(SELF, D, (vir_bytes)&imgrd[position],
            proc_nr, D, user_vir, count);
    } else {
        sys_vircopy(proc_nr, D, user_vir,
            SELF, D, (vir_bytes)&imgrd[position], count);
    }
    break;

/* Unknown (illegal) minor device. */
default:
    return(EINVAL);
}

/* Book the number of bytes transferred. */
position += count;
iov->iov_addr += count;
if ((iov->iov_size -= count) == 0) { iov++; nr_req--; }

}
return(OK);
}

```

```

/*=====
 *                               m_do_open                               *
 *=====*/
PRIVATE int m_do_open(dp, m_ptr)
struct driver *dp;
message *m_ptr;
{
    int r;

/* Check device number on open. */
    if (m_prepare(m_ptr->DEVICE) == NIL_DEV) return(ENXIO);
    if (m_device == MEM_DEV)
    {
        r = sys_enable_iop(m_ptr->IO_ENDPT);
        if (r != OK)
        {
            printf("m_do_open: sys_enable_iop failed for %d: %d\n",
                m_ptr->IO_ENDPT, r);
            return r;
        }
    }
    return(OK);
}

/*=====
 *                               m_init                               *
 *=====*/
PRIVATE void m_init()
{
    /* Initialize this task. All minor devices are initialized one by one. */
    phys_bytes ramdev_size;
    phys_bytes ramdev_base;
    message m;
    int i, s;

    if (OK != (s=sys_getkinfo(&kinfo))) {
        panic("MEM", "Couldn't get kernel information.", s);
    }

    /* Install remote segment for /dev/kmem memory. */
    m_geom[KMEM_DEV].dv_base = cvul64(kinfo.kmem_base);
    m_geom[KMEM_DEV].dv_size = cvul64(kinfo.kmem_size);
    if (OK != (s=sys_segctl(&m_seg[KMEM_DEV], (ul6_t *) &s, (vir_bytes *) &s,
        kinfo.kmem_base, kinfo.kmem_size))) {
        panic("MEM", "Couldn't install remote segment.", s);
    }

    /* Install remote segment for /dev/boot memory, if enabled. */
    m_geom[BOOT_DEV].dv_base = cvul64(kinfo.bootdev_base);
    m_geom[BOOT_DEV].dv_size = cvul64(kinfo.bootdev_size);
    if (kinfo.bootdev_base > 0) {
        if (OK != (s=sys_segctl(&m_seg[BOOT_DEV], (ul6_t *) &s, (vir_bytes *) &s,
            kinfo.bootdev_base, kinfo.bootdev_size))) {
            panic("MEM", "Couldn't install remote segment.", s);
        }
    }

    /* See if there are already RAM disk details at the Data Store server. */
    m.DS_KEY = MEMORY_MAJOR;
    if (OK == (s = _taskcall(DS_PROC_NR, DS_RETRIEVE, &m))) {
        ramdev_size = m.DS_VAL_L1;
        ramdev_base = m.DS_VAL_L2;
        printf("MEM retrieved size %u and base %u from DS, status %d\n",
            ramdev_size, ramdev_base, s);
        if (OK != (s=sys_segctl(&m_seg[RAM_DEV], (ul6_t *) &s,
            (vir_bytes *) &s, ramdev_base, ramdev_size))) {
            panic("MEM", "Couldn't install remote segment.", s);
        }
        m_geom[RAM_DEV].dv_base = cvul64(ramdev_base);
        m_geom[RAM_DEV].dv_size = cvul64(ramdev_size);
        printf("MEM stored retrieved details as new RAM disk\n");
    }

    /* Ramdisk image built into the memory driver */

```

```

m_geom[IMGRD_DEV].dv_base= cvul64(0);
m_geom[IMGRD_DEV].dv_size= cvul64(imgrd_size);

/* Initialize /dev/zero. Simply write zeros into the buffer. */
for (i=0; i<ZERO_BUF_SIZE; i++) {
    dev_zero[i] = '\0';
}

/* Set up memory ranges for /dev/mem. */
#if (CHIP == INTEL)
    if (OK != (s=sys_getmachine(&machine))) {
        panic("MEM", "Couldn't get machine information.", s);
    }
    if (! machine.protected) {
        m_geom[MEM_DEV].dv_size = cvul64(0x100000); /* 1M for 8086 systems */
    } else {
#endif
#if _WORD_SIZE == 2
        m_geom[MEM_DEV].dv_size = cvul64(0x1000000); /* 16M for 286 systems */
#else
        m_geom[MEM_DEV].dv_size = cvul64(0xFFFFFFFF); /* 4G-1 for 386 systems */
#endif
    }
#else /* !(CHIP == INTEL) */
#if (CHIP == M68000)
    m_geom[MEM_DEV].dv_size = cvul64(MEM_BYTES);
#else /* !(CHIP == M68000) */
#error /* memory limit not set up */
#endif /* !(CHIP == M68000) */
#endif /* !(CHIP == INTEL) */
}

/*=====
*
*                               m_ioctl
*=====*/
PRIVATE int m_ioctl(dp, m_ptr)
struct driver *dp;                /* pointer to driver structure */
message *m_ptr;                  /* pointer to control message */
{
    /* I/O controls for the memory driver. Currently there is one I/O control:
    * - MIOCRAMSIZE: to set the size of the RAM disk.
    */
    struct device *dv;

    switch (m_ptr->REQUEST) {
        case MIOCRAMSIZE: {
            /* Someone wants to create a new RAM disk with the given size. */
            static int first_time= 1;

            u32_t ramdev_size;
            phys_bytes ramdev_base;
            message m;
            int s;

            /* A ramdisk can be created only once, and only on RAM disk device. */
            if (!first_time) return(EPERM);
            if (m_ptr->DEVICE != RAM_DEV) return(EINVAL);
            if ((dv = m_prepare(m_ptr->DEVICE)) == NIL_DEV) return(ENXIO);

            if 0
                ramdev_size= m_ptr->POSITION;
            else
                /* Get request structure */
                s= sys_vircopy(m_ptr->IO_ENDPT, D, (vir_bytes)m_ptr->ADDRESS,
                    SELF, D, (vir_bytes)&ramdev_size, sizeof(ramdev_size));
                if (s != OK)
                    return s;
            endif

            if DEBUG
                printf("allocating ramdisk of size 0x%x\n", ramdev_size);
            endif

            /* Try to allocate a piece of memory for the RAM disk. */
            if (allocmem(ramdev_size, &ramdev_base) < 0) {

```

```

        report("MEM", "warning, allocmem failed", errno);
        return(ENOMEM);
    }

    /* Store the values we got in the data store so we can retrieve
     * them later on, in the unfortunate event of a crash.
     */
    m.DS_KEY = MEMORY_MAJOR;
    m.DS_VAL_L1 = ramdev_size;
    m.DS_VAL_L2 = ramdev_base;
    if (OK != (s = _taskcall(DS_PROC_NR, DS_PUBLISH, &m))) {
        panic("MEM", "Couldn't store RAM disk details at DS.", s);
    }
#ifdef DEBUG
    printf("MEM stored size %u and base %u at DS, status %d\n",
        ramdev_size, ramdev_base, s);
#endif

    if (OK != (s=sys_segctl(&m_seg[RAM_DEV], (ul6_t *) &s,
        (vir_bytes *) &s, ramdev_base, ramdev_size))) {
        panic("MEM", "Couldn't install remote segment.", s);
    }

    dv->dv_base = cvul64(ramdev_base);
    dv->dv_size = cvul64(ramdev_size);
    /* first_time= 0; */
    break;
}
case MIOCMAP:
case MIOCUNMAP: {
    int r, do_map;
    struct mapreq mapreq;

    if ((*dp->dr_prepare)(m_ptr->DEVICE) == NIL_DEV) return(ENXIO);
    if (m_device != MEM_DEV)
        return ENOTTY;

    do_map= (m_ptr->REQUEST == MIOCMAP);    /* else unmap */

    /* Get request structure */
    r= sys_vircopy(m_ptr->IO_ENDPT, D, (vir_bytes)m_ptr->ADDRESS,
        SELF, D, (vir_bytes)&mapreq, sizeof(mapreq));
    if (r != OK)
        return r;
    r= sys_vm_map(m_ptr->IO_ENDPT, do_map,
        (phys_bytes)mapreq.base, mapreq.size, mapreq.offset);
    return r;
}

default:
    return(do_diocntl(&m_dtab, m_ptr));
}
return(OK);
}

/*=====
 *                               m_geometry                               *
 *=====*/
PRIVATE void m_geometry(entry)
struct partition *entry;
{
    /* Memory devices don't have a geometry, but the outside world insists. */
    entry->cylinders = div64u(m_geom[m_device].dv_size, SECTOR_SIZE) / (64 * 32);
    entry->heads = 64;
    entry->sectors = 32;
}

```

```
# Makefile for ramdisk image
```

```
PROGRAMS=at_wini bios_wini cdprobe dev2name floppy loadramdisk newroot \  
pci sh service sysenv
```

```
MAKEDEV=/usr/bin/MAKEDEV
```

```
all:      image.c image.s
```

```
clean:    rm -rf $(PROGRAMS) bintoc image image.c t proto.gen
```

```
image.c:  bintoc image  
./bintoc -o $@ image
```

```
image.s:  image.c  
sed < image.c > $@ 's/^.datal;/s,$$/ ' || { rm -f $@; false; }
```

```
# Note for cross compilation: this executable has to be compiled for the  
# host system
```

```
bintoc:   bintoc.c  
$(CC) -o $@ bintoc.c
```

```
image:    proto.gen mtab rc $(PROGRAMS)  
mkfs -B 2048 image proto.gen || { rm -f image; false; }
```

```
at_wini:  ../../at_wini/at_wini  
install -s ../../$@/$@ $@
```

```
../../at_wini/at_wini:  
cd ../../at_wini && make
```

```
bios_wini: ../../bios_wini/bios_wini  
install -s ../../$@/$@ $@
```

```
../../bios_wini/bios_wini:  
cd ../../bios_wini && make
```

```
floppy:   ../../floppy/floppy  
install -s ../../$@/$@ $@
```

```
../../floppy/floppy:  
cd ../../floppy && make
```

```
pci:      ../../pci/pci  
install -s ../../$@/$@ $@
```

```
../../pci/pci:  
cd ../../pci && make
```

```
cdprobe:  ../../../../commands/simple/cdprobe  
install -s ../../../../commands/simple/$@ $@
```

```
../../../../commands/simple/cdprobe:  
cd ../../../../commands/simple && make cdprobe
```

```
dev2name:  ../../../../commands/simple/dev2name  
install -s ../../../../commands/simple/$@ $@
```

```
../../../../commands/simple/dev2name:  
cd ../../../../commands/simple && make dev2name
```

```
loadramdisk:  ../../../../commands/simple/loadramdisk  
install -s ../../../../commands/simple/$@ $@
```

```
../../../../commands/simple/loadramdisk:  
cd ../../../../commands/simple && make loadramdisk
```

```
newroot:  ../../../../commands/simple/newroot  
install -s ../../../../commands/simple/$@ $@
```

```
../../../../commands/simple/newroot:  
cd ../../../../commands/simple && make newroot
```

```
sysenv: ../../../../commands/simple/sysenv
install -s ../../../../commands/simple/$@ $@

../../../../commands/simple/sysenv:
cd ../../../../commands/simple && make sysenv

sh: ../../../../commands/ash/sh
install -s ../../../../commands/ash/$@ $@

../../../../commands/ash/sh:
cd ../../../../commands/ash && make sh

service: ../../../../servers/rs/service
install -s ../../../../servers/rs/$@ $@

../../../../servers/rs/service:
cd ../../../../servers/rs && make service

depend:
/usr/bin/mkdep "$(CC) -E $(CPPFLAGS)" *.c > .depend

proto.gen: $(MAKEDEV) proto.sh proto
sh -e proto.sh >proto.gen

# Include generated dependencies.
include .depend
```

```
/*
bintoc.c

Convert a (binary) file to a series of comma separated hex values suitable
for initializing a character array in C.
*/

#define _POSIX_C_SOURCE 2

#include <errno.h>
#include <stdarg.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <unistd.h>

char *progname;
unsigned char buf[1024];

static void fatal(char *fmt, ...);
static void usage(void);

int main(int argc, char *argv[])
{
    int c, i, r, first;
    FILE *file_in, *file_out;
    char *in_name;
    char *o_arg;

    (progname=strrchr(argv[0], '/')) ? progname++ : (progname=argv[0]);

    o_arg= NULL;
    while (c= getopt(argc, argv, "?o:"), c != -1)
    {
        switch(c)
        {
            case '?': usage();
            case 'o': o_arg= optarg; break;
            default: fatal("getopt failed: '%c'\n", c);
        }
    }

    if (o_arg)
    {
        file_out= fopen(o_arg, "w");
        if (file_out == NULL)
        {
            fatal("unable to create '%s': %s\n",
                o_arg, strerror(errno));
            exit(1);
        }
    }
    else
        file_out= stdout;

    if (optind < argc)
    {
        in_name= argv[optind];
        optind++;
        file_in= fopen(in_name, "r");
        if (file_in == NULL)
        {
            fatal("unable to open '%s': %s",
                in_name, strerror(errno));
        }
    }
    else
    {
        in_name= "(stdin)";
        file_in= stdin;
    }

    if (optind != argc)
        usage();
}
```

```
first= 1;
for (;;)
{
    r= fread(buf, 1, sizeof(buf), file_in);
    if (r == 0)
        break;
    for (i= 0; i<r; i++)
    {
        if ((i % 8) == 0)
        {
            if (first)
            {
                fprintf(file_out, "\t");
                first= 0;
            }
            else
                fprintf(file_out, ",\n\t");
        }
        else
            fprintf(file_out, ",");
        fprintf(file_out, "0x%02x", buf[i]);
    }

    if (ferror(file_in))
    {
        fatal("read error on %s: %s\n",
            in_name, strerror(errno));
    }
    fprintf(file_out, "\n");

    exit(0);
}

static void fatal(char *fmt, ...)
{
    va_list ap;

    fprintf(stderr, "%s: ", progname);

    va_start(ap, fmt);
    vfprintf(stderr, fmt, ap);
    va_end(ap);

    fprintf(stderr, "\n");

    exit(1);
}

static void usage(void)
{
    fprintf(stderr, "Usage: bintoc [-o <out-file>] [<in-file>]\n");
    exit(1);
}
```



/dev/imgrd / 3 rw

```
boot
140 310
d--755 0 0
    bin d--755 0 0
        at_wini ---755 0 0 at_wini
        bios_wini ---755 0 0 bios_wini
        cdprobe ---755 0 0 cdprobe
        dev2name ---755 0 0 dev2name
        floppy ---755 0 0 floppy
        loadramdisk ---755 0 0 loadramdisk
        newroot ---755 0 0 newroot
        pci ---755 0 0 pci
        sh ---755 0 0 sh
        service ---755 0 0 service
        sysenv ---755 0 0 sysenv
    $
    dev d--755 0 0
@DEV@
    $
    etc d--755 0 0
        mtab ---644 0 0 mtab
        rc ---755 0 0 rc
    $
$
```

```
#!/bin/sh
sed -n '1,/@DEV/p' <proto | grep -v @DEV@
(
cd /dev
ls -aln | grep '^[bc]' | egrep -v ' (fd1|fd0p|tcp|eth|ip|udp|tty[pq]|pty)' | \
sed      -e 's/^[bc]/&/' -e 's/rw-/6/g' -e 's/r--/4/g' \
          -e 's/-w-/2/g' -e 's/---/0/g' | \
awk '{ printf "\t\t%s %s--%s %d %d %d %d\n", $11, $1, $2, $4, $5, $6, $7; }'
)
sed -n '/@DEV/, $p' <proto | grep -v @DEV@
```

```
#!/bin/sh
set -e
/bin/service up /bin/pci
/bin/service -c up /bin/floppy -dev /dev/fd0
if [ X`/bin/sysenv bios_wini` = Xyes ]
then
    echo Using bios_wini.
    /bin/service -c up /bin/bios_wini -dev /dev/c0d0
else
    /bin/service -c up /bin/at_wini -dev /dev/c0d0
fi

rootdev=`sysenv rootdev` || echo 'No rootdev?'
rootdevname=`/bin/dev2name "$rootdev"` ||
{ echo 'No device name for root device'; exit 1; }

if sysenv cdproberoot >/dev/null
then
    echo
    echo 'Looking for boot CD. This may take a minute.'
    echo 'Please ignore any error messages.'
    echo
    cddev=`cdprobe` || { echo 'No CD found'; exit 1; }
    export cddev
    echo "Loading ramdisk from ${cddev}p1"
    loadramdisk "$cddev"p1
elif [ "$rootdevname" = "/dev/ram" ]
then
    ramimagedev=`sysenv ramimagedev` ||
    { echo 'ramimagedev not found'; exit 1; }
    ramimagenamename=`/bin/dev2name "$ramimagedev"` ||
    { echo 'No device name for ramimagedev'; exit 1; }
    echo "Loading ramdisk from $ramimagenamename"
    loadramdisk "$ramimagenamename"
fi
echo "Root device name is $rootdevname"
/bin/newroot "$rootdevname"
exec /bin/sh /etc/rc "$@"
```

```
#include <lib.h>
#include <unistd.h>

PUBLIC int allocmem(size, base)
    phys_bytes size;          /* size of mem chunk requested */
    phys_bytes *base;         /* return base address */
{
    message m;
    m.m4_l1 = size;
    if (_syscall(MM, ALLOCMEM, &m) < 0) return(-1);
    *base = m.m4_l2;
    return(0);
}
```

```
# Makefile for PCI support
DRIVER = pci

# directories
u = /usr
i = $u/include
s = $i/sys
m = $i/minix
b = $i/ibm
d = ..

# programs, flags, etc.
CC = cc
CFLAGS = -I$i
LDFLAGS = -i
LIBS = -lsys -lsysutil -ltimers

OBJ = main.o pci.o pci_table.o

# build local binary
all build: $(DRIVER)
$(DRIVER): $(OBJ)
    $(CC) -o $@ $(LDFLAGS) $(OBJ) $(LIBS)
    install -S 4096 $(DRIVER)

# install with other drivers
install: /usr/sbin/$(DRIVER)
/usr/sbin/$(DRIVER): $(DRIVER)
    install -o root -cs $? $@

# clean up local files
clean:
    rm -f *.o *.bak $(DRIVER)

depend:
    /usr/bin/mkdep "$(CC) -E $(CPPFLAGS)" *.c > .depend

# Include generated dependencies.
include .depend
```

```

/*
main.c
*/

#include "../drivers.h"

#include <ibm/pci.h>

#include "pci.h"

#define NR_DRIVERS      16

PRIVATE struct name
{
    char name[M3_STRING];
    int tasknr;
} names[NR_DRIVERS];

FORWARD _PROTOTYPE( void do_init, (message *mp) ) ;
FORWARD _PROTOTYPE( void do_sig_handler, (void) ) ;
FORWARD _PROTOTYPE( void do_first_dev, (message *mp) ) ;
FORWARD _PROTOTYPE( void do_next_dev, (message *mp) ) ;
FORWARD _PROTOTYPE( void do_find_dev, (message *mp) ) ;
FORWARD _PROTOTYPE( void do_ids, (message *mp) ) ;
FORWARD _PROTOTYPE( void do_dev_name, (message *mp) ) ;
FORWARD _PROTOTYPE( void do_slot_name, (message *mp) ) ;
FORWARD _PROTOTYPE( void do_reserve, (message *mp) ) ;
FORWARD _PROTOTYPE( void do_attr_r8, (message *mp) ) ;
FORWARD _PROTOTYPE( void do_attr_r16, (message *mp) ) ;
FORWARD _PROTOTYPE( void do_attr_r32, (message *mp) ) ;
FORWARD _PROTOTYPE( void do_attr_w8, (message *mp) ) ;
FORWARD _PROTOTYPE( void do_attr_w16, (message *mp) ) ;
FORWARD _PROTOTYPE( void do_attr_w32, (message *mp) ) ;
FORWARD _PROTOTYPE( void do_rescan_bus, (message *mp) ) ;

int main(void)
{
    int i, r;
    message m;

    pci_init();

    for (i= 0; i<NR_DRIVERS; i++)
        names[i].tasknr= ANY;

    for(;;)
    {
        r= receive(ANY, &m);
        if (r < 0)
        {
            printf("PCI: receive from ANY failed: %d\n", r);
            break;
        }
        switch(m.m_type)
        {
            case BUSC_PCI_INIT: do_init(&m); break;
            case BUSC_PCI_FIRST_DEV: do_first_dev(&m); break;
            case BUSC_PCI_NEXT_DEV: do_next_dev(&m); break;
            case BUSC_PCI_FIND_DEV: do_find_dev(&m); break;
            case BUSC_PCI_IDS: do_ids(&m); break;
            case BUSC_PCI_DEV_NAME: do_dev_name(&m); break;
            case BUSC_PCI_SLOT_NAME: do_slot_name(&m); break;
            case BUSC_PCI_RESERVE: do_reserve(&m); break;
            case BUSC_PCI_ATTR_R8: do_attr_r8(&m); break;
            case BUSC_PCI_ATTR_R16: do_attr_r16(&m); break;
            case BUSC_PCI_ATTR_R32: do_attr_r32(&m); break;
            case BUSC_PCI_ATTR_W8: do_attr_w8(&m); break;
            case BUSC_PCI_ATTR_W16: do_attr_w16(&m); break;
            case BUSC_PCI_ATTR_W32: do_attr_w32(&m); break;
            case BUSC_PCI_RESCAN: do_rescan_bus(&m); break;
            case PROC_EVENT: do_sig_handler(); break;
            default:
                printf("PCI: got message from %d, type %d\n",
                    m.m_source, m.m_type);
        }
    }
}

```

```

        break;
    }
}

return 0;
}

/*=====
 *                               do_sig_handler                               *
 *=====*/
PRIVATE void do_sig_handler()
{
    sigset_t sigset;
    int sig;

    /* Try to obtain signal set from PM. */
    if (getsigset(&sigset) != 0) return;

    /* Check for known signals. */
    if (sigismember(&sigset, SIGTERM)) {
        exit(0);
    }
}

PRIVATE void do_init(mp)
message *mp;
{
    int i, r, empty;

#ifdef DEBUG
    printf("pci_init: called by '%s'\n", mp->m3_cal);
#endif
    empty = -1;
    for (i = 0; i < NR_DRIVERS; i++)
    {
        if (empty == -1 && names[i].tasknr == ANY)
            empty = i;
        if (strcmp(names[i].name, mp->m3_cal) == 0)
            break;
    }
    if (i < NR_DRIVERS)
        pci_release(names[i].name);
    else
    {
        i = empty;
        strcpy(names[i].name, mp->m3_cal);
    }
    names[i].tasknr = mp->m_source;

    mp->m_type = 0;
    r = send(mp->m_source, mp);
    if (r != 0)
        printf("do_init: unable to send to %d: %d\n", mp->m_source, r);
}

PRIVATE void do_first_dev(mp)
message *mp;
{
    int r, devind;
    ul6_t vid, did;

    r = pci_first_dev(&devind, &vid, &did);
    if (r == 1)
    {
        mp->m1_i1 = devind;
        mp->m1_i2 = vid;
        mp->m1_i3 = did;
    }
    mp->m_type = r;
    r = send(mp->m_source, mp);
    if (r != 0)
    {
        printf("do_first_dev: unable to send to %d: %d\n",
            mp->m_source, r);
    }
}

```



```
    }  
}  
  
PRIVATE void do_next_dev(mp)  
message *mp;  
{  
    int r, devind;  
    ul6_t vid, did;  
  
    devind= mp->m1_i1;  
  
    r= pci_next_dev(&devind, &vid, &did);  
    if (r == 1)  
    {  
        mp->m1_i1= devind;  
        mp->m1_i2= vid;  
        mp->m1_i3= did;  
    }  
    mp->m_type= r;  
    r= send(mp->m_source, mp);  
    if (r != 0)  
    {  
        printf("do_next_dev: unable to send to %d: %d\n",  
               mp->m_source, r);  
    }  
}  
  
PRIVATE void do_find_dev(mp)  
message *mp;  
{  
    int r, devind;  
    u8_t bus, dev, func;  
  
    bus= mp->m1_i1;  
    dev= mp->m1_i2;  
    func= mp->m1_i3;  
  
    r= pci_find_dev(bus, dev, func, &devind);  
    if (r == 1)  
        mp->m1_i1= devind;  
    mp->m_type= r;  
    r= send(mp->m_source, mp);  
    if (r != 0)  
    {  
        printf("do_find_dev: unable to send to %d: %d\n",  
               mp->m_source, r);  
    }  
}  
  
PRIVATE void do_ids(mp)  
message *mp;  
{  
    int r, devind;  
    ul6_t vid, did;  
  
    devind= mp->m1_i1;  
  
    pci_ids(devind, &vid, &did);  
    mp->m1_i1= vid;  
    mp->m1_i2= did;  
    mp->m_type= OK;  
    r= send(mp->m_source, mp);  
    if (r != 0)  
    {  
        printf("do_ids: unable to send to %d: %d\n",  
               mp->m_source, r);  
    }  
}  
  
PRIVATE void do_dev_name(mp)  
message *mp;  
{  
    int r, name_len, len;  
    ul6_t vid, did;
```

```
    char *name_ptr, *name;

    vid= mp->m1_i1;
    did= mp->m1_i2;
    name_len= mp->m1_i3;
    name_ptr= mp->m1_p1;

    name= pci_dev_name(vid, did);
    if (name == NULL)
    {
        /* No name */
        r= ENOENT;
    }
    else
    {
        len= strlen(name)+1;
        if (len > name_len)
            len= name_len;
        r= sys_vircopy(SELF, D, (vir_bytes)name, mp->m_source, D,
            (vir_bytes)name_ptr, len);
    }

    mp->m_type= r;
    r= send(mp->m_source, mp);
    if (r != 0)
    {
        printf("do_dev_name: unable to send to %d: %d\n",
            mp->m_source, r);
    }
}

PRIVATE void do_slot_name(mp)
message *mp;
{
    int r, devind, name_len, len;
    char *name_ptr, *name;

    devind= mp->m1_i1;
    name_len= mp->m1_i2;
    name_ptr= mp->m1_p1;

    name= pci_slot_name(devind);

    len= strlen(name)+1;
    if (len > name_len)
        len= name_len;
    r= sys_vircopy(SELF, D, (vir_bytes)name, mp->m_source, D,
        (vir_bytes)name_ptr, len);

    mp->m_type= r;
    r= send(mp->m_source, mp);
    if (r != 0)
    {
        printf("do_slot_name: unable to send to %d: %d\n",
            mp->m_source, r);
    }
}

PRIVATE void do_reserve(mp)
message *mp;
{
    int i, r, devind;

    /* Find the name of the caller */
    for (i= 0; i<NR_DRIVERS; i++)
    {
        if (names[i].tasknr == mp->m_source)
            break;
    }
    if (i >= NR_DRIVERS)
    {
        printf("pci'do_reserve: task %d did not call pci_init\n",
            mp->m_source);
        return;
    }
}
```

```
    }

    devind= mp->m1_i1;

    pci_reserve3(devind, mp->m_source, names[i].name);
    mp->m_type= OK;
    r= send(mp->m_source, mp);
    if (r != 0)
    {
        printf("do_reserve: unable to send to %d: %d\n",
               mp->m_source, r);
    }
}

PRIVATE void do_attr_r8(mp)
message *mp;
{
    int r, devind, port;
    u8_t v;

    devind= mp->m2_i1;
    port= mp->m2_i2;

    v= pci_attr_r8(devind, port);
    mp->m2_l1= v;
    mp->m_type= OK;
    r= send(mp->m_source, mp);
    if (r != 0)
    {
        printf("do_attr_r8: unable to send to %d: %d\n",
               mp->m_source, r);
    }
}

PRIVATE void do_attr_r16(mp)
message *mp;
{
    int r, devind, port;
    u32_t v;

    devind= mp->m2_i1;
    port= mp->m2_i2;

    v= pci_attr_r16(devind, port);
    mp->m2_l1= v;
    mp->m_type= OK;
    r= send(mp->m_source, mp);
    if (r != 0)
    {
        printf("do_attr_r16: unable to send to %d: %d\n",
               mp->m_source, r);
    }
}

PRIVATE void do_attr_r32(mp)
message *mp;
{
    int r, devind, port;
    u32_t v;

    devind= mp->m2_i1;
    port= mp->m2_i2;

    v= pci_attr_r32(devind, port);
    mp->m2_l1= v;
    mp->m_type= OK;
    r= send(mp->m_source, mp);
    if (r != 0)
    {
        printf("do_attr_r32: unable to send to %d: %d\n",
               mp->m_source, r);
    }
}
```

```
PRIVATE void do_attr_w8(mp)
message *mp;
{
    int r, devind, port;
    u8_t v;

    devind= mp->m2_i1;
    port= mp->m2_i2;
    v= mp->m2_l1;

    pci_attr_w8(devind, port, v);
    mp->m_type= OK;
    r= send(mp->m_source, mp);
    if (r != 0)
    {
        printf("do_attr_w8: unable to send to %d: %d\n",
               mp->m_source, r);
    }
}

PRIVATE void do_attr_w16(mp)
message *mp;
{
    int r, devind, port;
    u16_t v;

    devind= mp->m2_i1;
    port= mp->m2_i2;
    v= mp->m2_l1;

    pci_attr_w16(devind, port, v);
    mp->m_type= OK;
    r= send(mp->m_source, mp);
    if (r != 0)
    {
        printf("do_attr_w16: unable to send to %d: %d\n",
               mp->m_source, r);
    }
}

PRIVATE void do_attr_w32(mp)
message *mp;
{
    int r, devind, port;
    u32_t v;

    devind= mp->m2_i1;
    port= mp->m2_i2;
    v= mp->m2_l1;

    pci_attr_w32(devind, port, v);
    mp->m_type= OK;
    r= send(mp->m_source, mp);
    if (r != 0)
    {
        printf("do_attr_w32: unable to send to %d: %d\n",
               mp->m_source, r);
    }
}

PRIVATE void do_rescan_bus(mp)
message *mp;
{
    int r, busnr;

    busnr= mp->m2_i1;

    pci_rescan_bus(busnr);
    mp->m_type= OK;
    r= send(mp->m_source, mp);
    if (r != 0)
    {
        printf("do_rescan_bus: unable to send to %d: %d\n",
               mp->m_source, r);
    }
}
```

```
}  
}
```

```
#define USER_SPACE 1
/*
pci.c

Configure devices on the PCI bus

Created:      Jan 2000 by Philip Homburg <philip@cs.vu.nl>
*/

#include "../drivers.h"
#define NDEBUG /* disable assertions */
#include <assert.h>
#include <ibm/pci.h>
#include <sys/vm.h>
#include <minix/com.h>
#include <minix/syslib.h>

#include "pci.h"
#include "pci_amd.h"
#include "pci_intel.h"
#include "pci_sis.h"
#include "pci_via.h"
#if __minix_vmd
#include "config.h"
#endif

#if !__minix_vmd
#define irq_mode_pci(irq) ((void)0)
#endif

#include <stdlib.h>
#include <stdio.h>
#include <string.h>
#include <minix/sysutil.h>

#define NR_PCIBUS 10
#define NR_PCIDEV 40

#define PBT_INTEL_HOST 1
#define PBT_PCIBRIDGE 2
#define PBT_CARDBUS 3

#define BAM_NR 6 /* Number of base-address registers */

PRIVATE int debug= 0;

PRIVATE struct pcibus
{
    int pb_type;
    int pb_needinit;
    int pb_isabridge_dev;
    int pb_isabridge_type;

    int pb_devind;
    int pb_busnr;
    u8_t (*pb_rreg8)(int busind, int devind, int port);
    u16_t (*pb_rreg16)(int busind, int devind, int port);
    u32_t (*pb_rreg32)(int busind, int devind, int port);
    void (*pb_wreg8)(int busind, int devind, int port, U8_t value);
    void (*pb_wreg16)(int busind, int devind, int port, U16_t value);
    void (*pb_wreg32)(int busind, int devind, int port, u32_t value);
    u16_t (*pb_rsts)(int busind);
    void (*pb_wsts)(int busind, U16_t value);
} pcibus[NR_PCIBUS];
PRIVATE int nr_pcibus= 0;

PRIVATE struct pcidev
{
    u8_t pd_busnr;
    u8_t pd_dev;
    u8_t pd_func;
    u8_t pd_baseclass;
    u8_t pd_subclass;
    u8_t pd_infclass;
```

```

    u16_t pd_vid;
    u16_t pd_did;
    u8_t pd_ilr;
    u8_t pd_inuse;

    struct bar
    {
        int pb_flags;
        int pb_nr;
        u32_t pb_base;
        u32_t pb_size;
    } pd_bar[BAM_NR];
    int pd_bar_nr;

    char pd_name[M3_STRING];
} pcidev[NR_PCIDEV];

/* pb_flags */
#define PBF_IO 1 /* I/O else memory */
#define PBF_INCOMPLETE 2 /* not allocated */

PRIVATE int nr_pcidev= 0;

/* Work around the limitations of the PCI emulation in QEMU 0.7.1 */
PRIVATE int qemu_pci= 0;

FORWARD _PROTOTYPE( void pci_intel_init, (void) );
FORWARD _PROTOTYPE( void probe_bus, (int busind) );
FORWARD _PROTOTYPE( int is_duplicate, (U8_t busnr, U8_t dev, U8_t func) );
FORWARD _PROTOTYPE( void record_irq, (int devind) );
FORWARD _PROTOTYPE( void recordBars, (int devind) );
FORWARD _PROTOTYPE( void recordBars_bridge, (int devind) );
FORWARD _PROTOTYPE( void recordBars_cardbus, (int devind) );
FORWARD _PROTOTYPE( void record_bar, (int devind, int bar_nr) );
FORWARD _PROTOTYPE( void complete_bridges, (void) );
FORWARD _PROTOTYPE( void complete_bars, (void) );
FORWARD _PROTOTYPE( void update_bridge4dev_io, (int devind,
                                                u32_t io_base, u32_t io_size) );
FORWARD _PROTOTYPE( int get_freebus, (void) );
FORWARD _PROTOTYPE( int do_isabridge, (int busind) );
FORWARD _PROTOTYPE( void do_pcibridge, (int busind) );
FORWARD _PROTOTYPE( int get_busind, (int busnr) );
FORWARD _PROTOTYPE( int do_piix, (int devind) );
FORWARD _PROTOTYPE( int do_amd_isabr, (int devind) );
FORWARD _PROTOTYPE( int do_sis_isabr, (int devind) );
FORWARD _PROTOTYPE( int do_via_isabr, (int devind) );
FORWARD _PROTOTYPE( void report_vga, (int devind) );
FORWARD _PROTOTYPE( char *pci_vid_name, (U16_t vid) );
FORWARD _PROTOTYPE( char *pci_baseclass_name, (U8_t baseclass) );
FORWARD _PROTOTYPE( char *pci_subclass_name, (U8_t baseclass,
                                                U8_t subclass, U8_t infclass) );
FORWARD _PROTOTYPE( void ntostr, (unsigned n, char **str, char *end) );
FORWARD _PROTOTYPE( u16_t pci_attr_rsts, (int devind) );
FORWARD _PROTOTYPE( void pci_attr_wsts, (int devind, U16_t value) );
FORWARD _PROTOTYPE( u16_t pcibr_std_rsts, (int busind) );
FORWARD _PROTOTYPE( void pcibr_std_wsts, (int busind, U16_t value) );
FORWARD _PROTOTYPE( u16_t pcibr_cb_rsts, (int busind) );
FORWARD _PROTOTYPE( void pcibr_cb_wsts, (int busind, U16_t value) );
FORWARD _PROTOTYPE( u16_t pcibr_via_rsts, (int busind) );
FORWARD _PROTOTYPE( void pcibr_via_wsts, (int busind, U16_t value) );
FORWARD _PROTOTYPE( u8_t pcii_rreg8, (int busind, int devind, int port) );
FORWARD _PROTOTYPE( u16_t pcii_rreg16, (int busind, int devind,
                                        int port) );
FORWARD _PROTOTYPE( u32_t pcii_rreg32, (int busind, int devind,
                                        int port) );
FORWARD _PROTOTYPE( void pcii_wreg8, (int busind, int devind, int port,
                                     U8_t value) );
FORWARD _PROTOTYPE( void pcii_wreg16, (int busind, int devind, int port,
                                       U16_t value) );
FORWARD _PROTOTYPE( void pcii_wreg32, (int busind, int devind, int port,
                                       u32_t value) );
FORWARD _PROTOTYPE( u16_t pcii_rsts, (int busind) );
FORWARD _PROTOTYPE( void pcii_wsts, (int busind, U16_t value) );
FORWARD _PROTOTYPE( void print_capabilities, (int devind) );

```

```

/*=====
 *
 *                               helper functions for I/O
 *
 *=====*/
PUBLIC unsigned pci_inb(U16_t port) {
    u32_t value;
    int s;
    if ((s=sys_inb(port, &value)) !=OK)
        printf("PCI: warning, sys_inb failed: %d\n", s);
    return value;
}
PUBLIC unsigned pci_inw(U16_t port) {
    u32_t value;
    int s;
    if ((s=sys_inw(port, &value)) !=OK)
        printf("PCI: warning, sys_inw failed: %d\n", s);
    return value;
}
PUBLIC unsigned pci_inl(U16_t port) {
    U32_t value;
    int s;
    if ((s=sys_inl(port, &value)) !=OK)
        printf("PCI: warning, sys_inl failed: %d\n", s);
    return value;
}
PUBLIC void pci_outb(U16_t port, U8_t value) {
    int s;
    if ((s=sys_outb(port, value)) !=OK)
        printf("PCI: warning, sys_outb failed: %d\n", s);
}
PUBLIC void pci_outw(U16_t port, U16_t value) {
    int s;
    if ((s=sys_outw(port, value)) !=OK)
        printf("PCI: warning, sys_outw failed: %d\n", s);
}
PUBLIC void pci_outl(U16_t port, U32_t value) {
    int s;
    if ((s=sys_outl(port, value)) !=OK)
        printf("PCI: warning, sys_outl failed: %d\n", s);
}

/*=====
 *
 *                               pci_init
 *
 *=====*/
PUBLIC void pci_init()
{
    static int first_time= 1;

    long v;

    if (!first_time)
        return;

    v= 0;
    env_parse("qemu_pci", "d", 0, &v, 0, 1);
    qemu_pci= v;

    v= 0;
    env_parse("pci_debug", "d", 0, &v, 0, 1);
    debug= v;

    /* We don't expect to interrupted */
    assert(first_time == 1);
    first_time= -1;

    /* Only Intel (compatible) PCI controllers are supported at the
     * moment.
     */
    pci_intel_init();

    first_time= 0;
}

/*=====

```



```
*                               pci_find_dev                               *
*=====*/
PUBLIC int pci_find_dev(bus, dev, func, devindp)
u8_t bus;
u8_t dev;
u8_t func;
int *devindp;
{
    int devind;

    for (devind= 0; devind < nr_pcidev; devind++)
    {
        if (pcidev[devind].pd_busnr == bus &&
            pcidev[devind].pd_dev == dev &&
            pcidev[devind].pd_func == func)
        {
            break;
        }
    }
    if (devind >= nr_pcidev)
        return 0;
    if (pcidev[devind].pd_inuse)
        return 0;
    *devindp= devind;
    return 1;
}

/*=====*/
*                               pci_first_dev                             *
*=====*/
PUBLIC int pci_first_dev(devindp, vidp, didp)
int *devindp;
ul6_t *vidp;
ul6_t *didp;
{
    int devind;

    for (devind= 0; devind < nr_pcidev; devind++)
    {
        if (!pcidev[devind].pd_inuse)
            break;
    }
    if (devind >= nr_pcidev)
        return 0;
    *devindp= devind;
    *vidp= pcidev[devind].pd_vid;
    *didp= pcidev[devind].pd_did;
    return 1;
}

/*=====*/
*                               pci_next_dev                             *
*=====*/
PUBLIC int pci_next_dev(devindp, vidp, didp)
int *devindp;
ul6_t *vidp;
ul6_t *didp;
{
    int devind;

    for (devind= *devindp+1; devind < nr_pcidev; devind++)
    {
        if (!pcidev[devind].pd_inuse)
            break;
    }
    if (devind >= nr_pcidev)
        return 0;
    *devindp= devind;
    *vidp= pcidev[devind].pd_vid;
    *didp= pcidev[devind].pd_did;
    return 1;
}

/*=====*/
```

```

*                               pci_reserve3                               *
*=====*/
PUBLIC void pci_reserve3(devind, proc, name)
int devind;
int proc;
char *name;
{
    int i, r;
    u8_t ilr;
    struct io_range ior;
    struct mem_range mr;

    assert(devind <= nr_pcidev);
    assert(!pcidev[devind].pd_inuse);
    pcidev[devind].pd_inuse = 1;
    strcpy(pcidev[devind].pd_name, name);

    for (i = 0; i < pcidev[devind].pd_bar_nr; i++)
    {
        if (pcidev[devind].pd_bar[i].pb_flags & PBF_INCOMPLETE)
        {
            printf("pci_reserve3: BAR %d is incomplete\n", i);
            continue;
        }
        if (pcidev[devind].pd_bar[i].pb_flags & PBF_IO)
        {
            ior.ior_base = pcidev[devind].pd_bar[i].pb_base;
            ior.ior_limit = ior.ior_base +
                pcidev[devind].pd_bar[i].pb_size - 1;

            if (debug) {
                printf(
                    "pci_reserve3: for proc %d, adding I/O range [0x%x..0x%x]\n",
                    proc, ior.ior_base, ior.ior_limit);
            }
            r = sys_privctl(proc, SYS_PRIV_ADD_IO, 0, &ior);
            if (r != OK)
            {
                printf("sys_privctl failed for proc %d: %d\n",
                    proc, r);
            }
        }
        else
        {
            mr.mr_base = pcidev[devind].pd_bar[i].pb_base;
            mr.mr_limit = mr.mr_base +
                pcidev[devind].pd_bar[i].pb_size - 1;

            if (debug) {
                printf(
                    "pci_reserve3: for proc %d, should add memory range [0x%x..0x%x]\n",
                    proc, mr.mr_base, mr.mr_limit);
            }
            r = sys_privctl(proc, SYS_PRIV_ADD_MEM, 0, &mr);
            if (r != OK)
            {
                printf("sys_privctl failed for proc %d: %d\n",
                    proc, r);
            }
        }
    }
    ilr = pcidev[devind].pd_ilr;
    if (ilr != PCI_ILR_UNKNOWN)
    {
        if (debug) printf("pci_reserve3: adding IRQ %d\n", ilr);
        r = sys_privctl(proc, SYS_PRIV_ADD_IRQ, ilr, NULL);
        if (r != OK)
        {
            printf("sys_privctl failed for proc %d: %d\n",
                proc, r);
        }
    }
}

```

```

/*=====
 *
 *                               pci_release
 *=====*/
PUBLIC void pci_release(name)
char *name;
{
    int i;

    for (i= 0; i<nr_pcidev; i++)
    {
        if (!pcidev[i].pd_inuse)
            continue;
        if (strcmp(pcidev[i].pd_name, name) != 0)
            continue;
        pcidev[i].pd_inuse= 0;
    }
}

/*=====
 *
 *                               pci_ids
 *=====*/
PUBLIC void pci_ids(devind, vidp, didp)
int devind;
ul6_t *vidp;
ul6_t *didp;
{
    assert(devind <= nr_pcidev);
    *vidp= pcidev[devind].pd_vid;
    *didp= pcidev[devind].pd_did;
}

/*=====
 *
 *                               pci_rescan_bus
 *=====*/
PUBLIC void pci_rescan_bus(busnr)
u8_t busnr;
{
    int busind;

    busind= get_busind(busnr);
    probe_bus(busind);

    /* Allocate bus numbers for uninitialized bridges */
    complete_bridges();

    /* Allocate I/O and memory resources for uninitialized devices */
    completeBars();
}

/*=====
 *
 *                               pci_slot_name
 *=====*/
PUBLIC char *pci_slot_name(devind)
int devind;
{
    static char label[]= "ddd.ddd.ddd";
    char *end;
    char *p;

    p= label;
    end= label+sizeof(label);

    ntostr(pcidev[devind].pd_busnr, &p, end);
    *p++= '.';

    ntostr(pcidev[devind].pd_dev, &p, end);
    *p++= '.';

    ntostr(pcidev[devind].pd_func, &p, end);

    return label;
}

/*=====

```

```

*                               pci_dev_name                               *
*=====*/
PUBLIC char *pci_dev_name(vid, did)
ul6_t vid;
ul6_t did;
{
    int i;

    for (i= 0; pci_device_table[i].name; i++)
    {
        if (pci_device_table[i].vid == vid &&
            pci_device_table[i].did == did)
        {
            return pci_device_table[i].name;
        }
    }
    return NULL;
}

/*=====*
*                               pci_attr_r8                               *
*=====*/
PUBLIC u8_t pci_attr_r8(devind, port)
int devind;
int port;
{
    int busnr, busind;

    busnr= pcidev[devind].pd_busnr;
    busind= get_busind(busnr);
    return pcibus[busind].pb_rreg8(busind, devind, port);
}

/*=====*
*                               pci_attr_r16                              *
*=====*/
PUBLIC ul6_t pci_attr_r16(devind, port)
int devind;
int port;
{
    int busnr, busind;

    busnr= pcidev[devind].pd_busnr;
    busind= get_busind(busnr);
    return pcibus[busind].pb_rreg16(busind, devind, port);
}

/*=====*
*                               pci_attr_r32                              *
*=====*/
PUBLIC u32_t pci_attr_r32(devind, port)
int devind;
int port;
{
    int busnr, busind;

    busnr= pcidev[devind].pd_busnr;
    busind= get_busind(busnr);
    return pcibus[busind].pb_rreg32(busind, devind, port);
}

/*=====*
*                               pci_attr_w8                               *
*=====*/
PUBLIC void pci_attr_w8(devind, port, value)
int devind;
int port;
ul6_t value;
{
    int busnr, busind;

    busnr= pcidev[devind].pd_busnr;
    busind= get_busind(busnr);
    pcibus[busind].pb_wreg8(busind, devind, port, value);
}

```

```

}

/*=====
 *                               pci_attr_w16                               *
 *=====*/
PUBLIC void pci_attr_w16(devind, port, value)
int devind;
int port;
u16_t value;
{
    int busnr, busind;

    busnr= pcidev[devind].pd_busnr;
    busind= get_busind(busnr);
    pcibus[busind].pb_wreg16(busind, devind, port, value);
}

/*=====
 *                               pci_attr_w32                               *
 *=====*/
PUBLIC void pci_attr_w32(devind, port, value)
int devind;
int port;
u32_t value;
{
    int busnr, busind;

    busnr= pcidev[devind].pd_busnr;
    busind= get_busind(busnr);
    pcibus[busind].pb_wreg32(busind, devind, port, value);
}

/*=====
 *                               pci_intel_init                             *
 *=====*/
PRIVATE void pci_intel_init()
{
    /* Try to detect a know PCI controller. Read the Vendor ID and
     * the Device ID for function 0 of device 0.
     * Two times the value 0xffff suggests a system without a (compatible)
     * PCI controller.
     */
    u32_t bus, dev, func;
    u16_t vid, did;
    int s, i, r, busind, busnr;
    char *dstr;

    bus= 0;
    dev= 0;
    func= 0;

    vid= PCII_RREG16_(bus, dev, func, PCI_VID);
    did= PCII_RREG16_(bus, dev, func, PCI_DID);
#if USER_SPACE
    if (OK != (s=sys_outl(PCII_CONFADD, PCII_UNSEL)))
        printf("PCI: warning, sys_outl failed: %d\n", s);
else
    outl(PCII_CONFADD, PCII_UNSEL);
endif

    if (vid == 0xffff && did == 0xffff)
        return; /* Nothing here */

#if 0
    for (i= 0; pci_intel_ctrl[i].vid; i++)
    {
        if (pci_intel_ctrl[i].vid == vid &&
            pci_intel_ctrl[i].did == did)
        {
            break;
        }
    }

    if (!pci_intel_ctrl[i].vid)

```

```

    {
        printf("pci_intel_init (warning): unknown PCI-controller:\n"
               "\tvendor %04X (%s), device %04X\n",
               vid, pci_vid_name(vid), did);
    }
#endif

    if (nr_pcibus >= NR_PCIBUS)
        panic("PCI", "too many PCI busses", nr_pcibus);
    busind= nr_pcibus;
    nr_pcibus++;
    pcibus[busind].pb_type= PBT_INTEL_HOST;
    pcibus[busind].pb_needinit= 0;
    pcibus[busind].pb_isabridge_dev= -1;
    pcibus[busind].pb_isabridge_type= 0;
    pcibus[busind].pb_devind= -1;
    pcibus[busind].pb_busnr= 0;
    pcibus[busind].pb_rreg8= pcii_rreg8;
    pcibus[busind].pb_rreg16= pcii_rreg16;
    pcibus[busind].pb_rreg32= pcii_rreg32;
    pcibus[busind].pb_wreg8= pcii_wreg8;
    pcibus[busind].pb_wreg16= pcii_wreg16;
    pcibus[busind].pb_wreg32= pcii_wreg32;
    pcibus[busind].pb_rsts= pcii_rsts;
    pcibus[busind].pb_wsts= pcii_wsts;

    dstr= pci_dev_name(vid, did);
    if (!dstr)
        dstr= "unknown device";
    if (debug)
    {
        printf("pci_intel_init: %s (%04X/%04X)\n",
               dstr, vid, did);
    }

    probe_bus(busind);

    r= do_isabridge(busind);
    if (r != OK)
    {
        busnr= pcibus[busind].pb_busnr;

        /* Disable all devices for this bus */
        for (i= 0; i<nr_pcidev; i++)
        {
            if (pcidev[i].pd_busnr != busnr)
                continue;
            pcidev[i].pd_inuse= 1;
        }
        return;
    }

    /* Look for PCI bridges */
    do_pcibridge(busind);

    /* Allocate bus numbers for uninitialized bridges */
    complete_bridges();

    /* Allocate I/O and memory resources for uninitialized devices */
    completeBars();
}

/*=====
*
*                               probe_bus
*=====*/
PRIVATE void probe_bus(busind)
int busind;
{
    u32_t dev, func, t3;
    u16_t vid, did, sts;
    u8_t headt;
    u8_t baseclass, subclass, infclass;
    int devind, busnr;
    char *s, *dstr;

```

```

#if DEBUG
printf("probe_bus(%d)\n", busind);
#endif

    if (nr_pcidev >= NR_PCIDEV)
        panic("PCI", "too many PCI devices", nr_pcidev);
    devind= nr_pcidev;

    busnr= pcibus[busind].pb_busnr;
    for (dev= 0; dev<32; dev++)
    {

        for (func= 0; func < 8; func++)
        {
            pcidev[devind].pd_busnr= busnr;
            pcidev[devind].pd_dev= dev;
            pcidev[devind].pd_func= func;

            pci_attr_wsts(devind,
                PSR_SSE|PSR_RMAS|PSR_RTAS);
            vid= pci_attr_rl6(devind, PCI_VID);
            did= pci_attr_rl6(devind, PCI_DID);
            headt= pci_attr_r8(devind, PCI_HEADT);
            sts= pci_attr_rsts(devind);

#if 0

            printf("vid 0x%x, did 0x%x, headt 0x%x, sts 0x%x\n",
                vid, did, headt, sts);

#endif

            if (vid == NO_VID)
            {
                if (func == 0)
                    break; /* Nothing here */

                /* Scan all functions of a multifunction
                 * device.
                 */
                continue;
            }

            if (sts & (PSR_SSE|PSR_RMAS|PSR_RTAS))
            {
                if (qemu_pci)
                {
                    printf(
"pci: ignoring bad value 0x%x in sts for QEMU\n",
                    sts & (PSR_SSE|PSR_RMAS|PSR_RTAS));
                }
                else
                {
                    if (func == 0)
                        break; /* Nothing here */

                    /* Scan all functions of a
                     * multifunction device.
                     */
                    continue;
                }
            }

            dstr= pci_dev_name(vid, did);
            if (debug)
            {
                if (dstr)
                {
                    printf("%d.%lu.%lu: %s (%04X/%04X)\n",
                        busind, (unsigned long)dev,
                        (unsigned long)func, dstr,
                        vid, did);
                }
                else
                {
                    printf(

```

```

"%d.%lu.%lu: Unknown device, vendor %04X (%s), device %04X\n",
    busind, (unsigned long)dev,
    (unsigned long)func, vid,
    pci_vid_name(vid), did);
    }
    printf("Device index: %d\n", devind);
    printf("Subsystem: Vid 0x%x, did 0x%x\n",
        pci_attr_r16(devind, PCI_SUBVID),
        pci_attr_r16(devind, PCI_SUBDID));
}

baseclass= pci_attr_r8(devind, PCI_BCR);
subclass= pci_attr_r8(devind, PCI_SCR);
infclass= pci_attr_r8(devind, PCI_PIFR);
s= pci_subclass_name(baseclass, subclass, infclass);
if (!s)
    s= pci_baseclass_name(baseclass);
{
    if (!s)
        s= "(unknown class)";
}
if (debug)
{
    printf("\tclass %s (%X/%X/%X)\n", s,
        baseclass, subclass, infclass);
}

if (is_duplicate(busnr, dev, func))
{
    printf("\tduplicate!\n");
    if (func == 0 && !(headt & PHT_MULTIFUNC))
        break;
    continue;
}

devind= nr_pcidev;
nr_pcidev++;
pcidev[devind].pd_baseclass= baseclass;
pcidev[devind].pd_subclass= subclass;
pcidev[devind].pd_infclass= infclass;
pcidev[devind].pd_vid= vid;
pcidev[devind].pd_did= did;
pcidev[devind].pd_inuse= 0;
pcidev[devind].pd_bar_nr= 0;
record_irq(devind);
switch(headt & PHT_MASK)
{
case PHT_NORMAL:
    recordBars(devind);
    break;
case PHT_BRIDGE:
    recordBars_bridge(devind);
    break;
case PHT_CARDBUS:
    recordBars_cardbus(devind);
    break;
default:
    printf("\t%d.%d.%d: unknown header type %d\n",
        busind, dev, func,
        headt & PHT_MASK);
    break;
}
if (debug)
    print_capabilities(devind);

t3= ((baseclass << 16) | (subclass << 8) | infclass);
if (t3 == PCI_T3_VGA || t3 == PCI_T3_VGA_OLD)
    report_vga(devind);

if (nr_pcidev >= NR_PCIDEV)
    panic("PCI", "too many PCI devices", nr_pcidev);
devind= nr_pcidev;

if (func == 0 && !(headt & PHT_MULTIFUNC))

```



```

        break;
    }
}

/*=====
 *                               is_duplicate                               *
 *=====*/
PRIVATE int is_duplicate(busnr, dev, func)
u8_t busnr;
u8_t dev;
u8_t func;
{
    int i;

    for (i= 0; i<nr_pcidev; i++)
    {
        if (pcidev[i].pd_busnr == busnr &&
            pcidev[i].pd_dev == dev &&
            pcidev[i].pd_func == func)
        {
            return 1;
        }
    }
    return 0;
}

/*=====
 *                               record_irq                               *
 *=====*/
PRIVATE void record_irq(devind)
int devind;
{
    int ilr, ipr, busnr, busind, cb_devind;

    ilr= pci_attr_r8(devind, PCI_ILR);
    ipr= pci_attr_r8(devind, PCI_IPR);
    if (ilr == 0)
    {
        static int first= 1;
        if (ipr && first && debug)
        {
            first= 0;
            printf("PCI: strange, BIOS assigned IRQ0\n");
        }
        ilr= PCI_ILR_UNKNOWN;
    }
    pcidev[devind].pd_ilr= ilr;
    if (ilr == PCI_ILR_UNKNOWN && !ipr)
    {
    }
    else if (ilr != PCI_ILR_UNKNOWN && ipr)
    {
        if (debug)
            printf("\tIRQ %d for INT%c\n", ilr, 'A' + ipr-1);
    }
    else if (ilr != PCI_ILR_UNKNOWN)
    {
        printf(
            "PCI: IRQ %d is assigned, but device %d.%d.%d does not need it\n",
            ilr, pcidev[devind].pd_busnr, pcidev[devind].pd_dev,
            pcidev[devind].pd_func);
    }
    else
    {
        /* Check for cardbus devices */
        busnr= pcidev[devind].pd_busnr;
        busind= get_busind(busnr);
        if (pcibus[busind].pb_type == PBT_CARDBUS)
        {
            cb_devind= pcibus[busind].pb_devind;
            ilr= pcidev[cb_devind].pd_ilr;
            if (ilr != PCI_ILR_UNKNOWN)
            {

```

```

        if (debug)
        {
            printf(
                "assigning IRQ %d to Cardbus device\n",
                ilr);
        }
        pci_attr_w8(devind, PCI_ILR, ilr);
        pcidev[devind].pd_ilr= ilr;
        return;
    }
}
if(debug) {
    printf(
        "PCI: device %d.%d.%d uses INT%c but is not assigned any IRQ\n",
        pcidev[devind].pd_busnr, pcidev[devind].pd_dev,
        pcidev[devind].pd_func, 'A' + ipr-1);
}
}

/*=====
 *                               record_bars                               *
 *=====*/
PRIVATE void record_bars(devind)
int devind;
{
    int i, j, reg, prefetch, type, clear_01, clear_23, pb_nr;
    u32_t bar, bar2;

    for (i= 0, reg= PCI_BAR; reg <= PCI_BAR_6; i++, reg += 4)
    {
        record_bar(devind, i);
    }

    /* Special case code for IDE controllers in compatibility mode */
    if (pcidev[devind].pd_baseclass == PCI_BCR_MASS_STORAGE &&
        pcidev[devind].pd_subclass == PCI_MS_IDE)
    {
        /* IDE device */
        clear_01= 0;
        clear_23= 0;
        if (!(pcidev[devind].pd_infclass & PCI_IDE_PRI_NATIVE))
        {
            if (debug)
            {
                printf(
                    "primary channel is not in native mode, clearing BARs 0 and 1\n" );
            }
            clear_01= 1;
        }
        if (!(pcidev[devind].pd_infclass & PCI_IDE_SEC_NATIVE))
        {
            if (debug)
            {
                printf(
                    "primary channel is not in native mode, clearing BARs 2 and 3\n" );
            }
            clear_23= 1;
        }

        j= 0;
        for (i= 0; i<pcidev[devind].pd_bar_nr; i++)
        {
            pb_nr= pcidev[devind].pd_bar[i].pb_nr;
            if ((pb_nr == 0 || pb_nr == 1) && clear_01)
            {
                if (debug) printf("skipping bar %d\n", pb_nr);
                continue; /* Skip */
            }
            if ((pb_nr == 2 || pb_nr == 3) && clear_23)
            {
                if (debug) printf("skipping bar %d\n", pb_nr);
                continue; /* Skip */
            }
        }
    }
}

```

```

        if (i == j)
            continue; /* No need to copy */
        pcidev[devind].pd_bar[j]=
            pcidev[devind].pd_bar[i];
        j++;
    }
    pcidev[devind].pd_bar_nr= j;
}

/*=====
 *                      recordBarsBridge
 *=====*/
PRIVATE void recordBarsBridge(devind)
int devind;
{
    u32_t base, limit, size;

    record_bar(devind, 0);
    record_bar(devind, 1);

    base= ((pci_attr_r8(devind, PPB_IOBASE) & PPB_IOB_MASK) << 8) |
        (pci_attr_r16(devind, PPB_IOBASEU16) << 16);
    limit= 0xff |
        ((pci_attr_r8(devind, PPB_IOLIMIT) & PPB_IOL_MASK) << 8) |
        ((~PPB_IOL_MASK & 0xff) << 8) |
        (pci_attr_r16(devind, PPB_IOLIMITU16) << 16);
    size= limit-base + 1;
    if (debug)
    {
        printf("\tI/O window: base 0x%x, limit 0x%x, size %d\n",
            base, limit, size);
    }

    base= ((pci_attr_r16(devind, PPB_MEMBASE) & PPB_MEMB_MASK) << 16);
    limit= 0xffff |
        ((pci_attr_r16(devind, PPB_MEMLIMIT) & PPB_MEML_MASK) << 16) |
        ((~PPB_MEML_MASK & 0xffff) << 16);
    size= limit-base + 1;
    if (debug)
    {
        printf("\tMemory window: base 0x%x, limit 0x%x, size 0x%x\n",
            base, limit, size);
    }

    /* Ignore the upper 32 bits */
    base= ((pci_attr_r16(devind, PPB_PFMEMBASE) & PPB_PFMEMB_MASK) << 16);
    limit= 0xffff |
        ((pci_attr_r16(devind, PPB_PFMEMLIMIT) &
            PPB_PFMEML_MASK) << 16) |
        ((~PPB_PFMEML_MASK & 0xffff) << 16);
    size= limit-base + 1;
    if (debug)
    {
        printf(
            "\tPrefetchable memory window: base 0x%x, limit 0x%x, size 0x%x\n",
            base, limit, size);
    }
}

/*=====
 *                      recordBarsCardbus
 *=====*/
PRIVATE void recordBarsCardbus(devind)
int devind;
{
    u32_t base, limit, size;

    record_bar(devind, 0);

    base= pci_attr_r32(devind, CBB_MEMBASE_0);
    limit= pci_attr_r32(devind, CBB_MEMLIMIT_0) |
        (~CBB_MEML_MASK & 0xffffffff);
    size= limit-base + 1;

```

```

    if (debug)
    {
        printf("\tMemory window 0: base 0x%x, limit 0x%x, size %d\n",
               base, limit, size);
    }

    base= pci_attr_r32(devind, CBB_MEMBASE_1);
    limit= pci_attr_r32(devind, CBB_MEMLIMIT_1) |
           (~CBB_MEML_MASK & 0xffffffff);
    size= limit-base + 1;
    if (debug)
    {
        printf("\tMemory window 1: base 0x%x, limit 0x%x, size %d\n",
               base, limit, size);
    }

    base= pci_attr_r32(devind, CBB_IOBASE_0);
    limit= pci_attr_r32(devind, CBB_IOLIMIT_0) |
           (~CBB_IOL_MASK & 0xffffffff);
    size= limit-base + 1;
    if (debug)
    {
        printf("\tI/O window 0: base 0x%x, limit 0x%x, size %d\n",
               base, limit, size);
    }

    base= pci_attr_r32(devind, CBB_IOBASE_1);
    limit= pci_attr_r32(devind, CBB_IOLIMIT_1) |
           (~CBB_IOL_MASK & 0xffffffff);
    size= limit-base + 1;
    if (debug)
    {
        printf("\tI/O window 1: base 0x%x, limit 0x%x, size %d\n",
               base, limit, size);
    }
}

/*=====
 *                                     record_bar                                     *
 *=====*/
PRIVATE void record_bar(devind, bar_nr)
int devind;
int bar_nr;
{
    int reg, prefetch, type, dev_bar_nr;
    u32_t bar, bar2;

    reg= PCI_BAR+4*bar_nr;

    bar= pci_attr_r32(devind, reg);
    if (bar & PCI_BAR_IO)
    {
        /* Size register */
        pci_attr_w32(devind, reg, 0xffffffff);
        bar2= pci_attr_r32(devind, reg);
        pci_attr_w32(devind, reg, bar);

        bar &= ~(u32_t)3;          /* Clear non-address bits */
        bar2 &= ~(u32_t)3;
        bar2= (~bar2 & 0xffff)+1;
        if (debug)
        {
            printf("\tbar_%d: %d bytes at 0x%x I/O\n",
                   bar_nr, bar2, bar);
        }

        dev_bar_nr= pcidev[devind].pd_bar_nr++;
        assert(dev_bar_nr < BAR_NR);
        pcidev[devind].pd_bar[dev_bar_nr].pb_flags= PBF_IO;
        pcidev[devind].pd_bar[dev_bar_nr].pb_base= bar;
        pcidev[devind].pd_bar[dev_bar_nr].pb_size= bar2;
        pcidev[devind].pd_bar[dev_bar_nr].pb_nr= bar_nr;
        if (bar == 0)
        {

```

```

        pcidev[devind].pd_bar[dev_bar_nr].pb_flags |=
            PBF_INCOMPLETE;
    }
}
else
{
    /* Size register */
    pci_attr_w32(devind, reg, 0xffffffff);
    bar2= pci_attr_r32(devind, reg);
    pci_attr_w32(devind, reg, bar2);

    if (bar2 == 0)
        return; /* Reg. is not implemented */

    prefetch= !(bar & PCI_BAR_PREFETCH);
    type= (bar & PCI_BAR_TYPE);
    bar &= ~(u32_t)0xf; /* Clear non-address bits */
    bar2 &= ~(u32_t)0xf;
    bar2= (~bar2)+1;
    if (debug)
    {
        printf("\tbar_%d: 0x%x bytes at 0x%x%s memory\n",
            bar_nr, bar2, bar,
            prefetch ? " prefetchable" : "");
        if (type != 0)
            printf("type= 0x%x\n", type);
    }

    dev_bar_nr= pcidev[devind].pd_bar_nr++;
    assert(dev_bar_nr < BAR_NR);
    pcidev[devind].pd_bar[dev_bar_nr].pb_flags= 0;
    pcidev[devind].pd_bar[dev_bar_nr].pb_base= bar;
    pcidev[devind].pd_bar[dev_bar_nr].pb_size= bar2;
    pcidev[devind].pd_bar[dev_bar_nr].pb_nr= bar_nr;
    if (bar == 0)
    {
        pcidev[devind].pd_bar[dev_bar_nr].pb_flags |=
            PBF_INCOMPLETE;
    }
}
}

/*=====
 *                               complete_bridges                               *
 *=====*/
PRIVATE void complete_bridges()
{
    int i, freebus, devind, prim_busnr;

    for (i= 0; i<nr_pcibus; i++)
    {
        if (!pcibus[i].pb_needinit)
            continue;
        printf("should allocate bus number for bus %d\n", i);
        freebus= get_freebus();
        printf("got bus number %d\n", freebus);

        devind= pcibus[i].pb_devind;

        prim_busnr= pcidev[devind].pd_busnr;
        if (prim_busnr != 0)
        {
            printf(
"complete_bridge: updating subordinate bus number not implemented\n" );
        }

        pcibus[i].pb_needinit= 0;
        pcibus[i].pb_busnr= freebus;

        printf("devind= %d\n", devind);
        printf("prim_busnr= %d\n", prim_busnr);

        pci_attr_w8(devind, PPB_PRIMBN, prim_busnr);
        pci_attr_w8(devind, PPB_SECBN, freebus);
    }
}

```

```

        pci_attr_w8(devind, PPB_SUBORDBN, freebus);

        printf("CR=0x%x\n", pci_attr_r16(devind, PCI_CR));
        printf("SECBLT=0x%x\n", pci_attr_r8(devind, PPB_SECBLT));
        printf("BRIDGECTRL=0x%x\n",
                pci_attr_r16(devind, PPB_BRIDGECTRL));
    }
}

/*=====
 *                               completeBars                               *
 *=====*/
PRIVATE void completeBars()
{
    int i, j, r, bar_nr, reg;
    u32_t memgap_low, memgap_high, iogap_low, iogap_high, io_high,
        base, size, v32, diff1, diff2;
    char *cp, *next;
    char memstr[256];

    r= env_get_param("memory", memstr, sizeof(memstr));
    if (r != OK)
        panic("pci", "env_get_param failed", r);

    /* Set memgap_low to just above physical memory */
    memgap_low= 0;
    cp= memstr;
    while (*cp != '\0')
    {
        base= strtoul(cp, &next, 16);
        if (next == cp || *next != ':')
        {
            printf("pci: bad memory environment string '%s'\n",
                    memstr);
            panic(NULL, NULL, NO_NUM);
        }
        cp= next+1;
        size= strtoul(cp, &next, 16);
        if (next == cp || (*next != ',' && *next != '\0'))
        {
            printf("pci: bad memory environment string '%s'\n",
                    memstr);
            panic(NULL, NULL, NO_NUM);
        }
        cp= next+1;

        if (base+size > memgap_low)
            memgap_low= base+size;
    }

    memgap_high= 0xfe000000;          /* Leave space for the CPU (APIC) */

    if (debug)
    {
        printf("completeBars: initial gap: [0x%x .. 0x%x>\n",
                memgap_low, memgap_high);
    }

    /* Find the lowest memory base */
    for (i= 0; i<nr_pcidev; i++)
    {
        for (j= 0; j<pcidev[i].pd_bar_nr; j++)
        {
            if (pcidev[i].pd_bar[j].pb_flags & PBF_IO)
                continue;
            if (pcidev[i].pd_bar[j].pb_flags & PBF_INCOMPLETE)
                continue;
            base= pcidev[i].pd_bar[j].pb_base;
            size= pcidev[i].pd_bar[j].pb_size;

            if (base >= memgap_high)
                continue;          /* Not in the gap */
            if (base+size <= memgap_low)
                continue;          /* Not in the gap */

```

```

        /* Reduce the gap by the smallest amount */
        diff1= base+size-memgap_low;
        diff2= memgap_high-base;

        if (diff1 < diff2)
            memgap_low= base+size;
        else
            memgap_high= base;
    }
}

if (debug)
{
    printf("complete_bars: intermediate gap: [0x%x .. 0x%x>\n",
        memgap_low, memgap_high);
}

/* Should check main memory size */
if (memgap_high < memgap_low)
{
    printf("pci: bad memory gap: [0x%x .. 0x%x>\n",
        memgap_low, memgap_high);
    panic(NULL, NULL, NO_NUM);
}

iogap_high= 0x10000;
iogap_low= 0x400;

/* Find the free I/O space */
for (i= 0; i<nr_pcidev; i++)
{
    for (j= 0; j<pcidev[i].pd_bar_nr; j++)
    {
        if (!(pcidev[i].pd_bar[j].pb_flags & PBF_IO))
            continue;
        if (pcidev[i].pd_bar[j].pb_flags & PBF_INCOMPLETE)
            continue;
        base= pcidev[i].pd_bar[j].pb_base;
        size= pcidev[i].pd_bar[j].pb_size;
        if (base >= iogap_high)
            continue;
        if (base+size <= iogap_low)
            continue;

        if (debug)
        {
            printf(
                "pci device %d (%04x/%04x), bar %d: base 0x%x, size 0x%x\n",
                i, pcidev[i].pd_vid, pcidev[i].pd_did,
                j, base, size);
        }

        if (base+size-iogap_low < iogap_high-base)
            iogap_low= base+size;
        else
            iogap_high= base;
    }
}

if (iogap_high < iogap_low)
{
    if (debug)
    {
        printf("iogap_high too low, should panic\n");
    }
    else
        panic("pci", "iogap_high too low", iogap_high);
}

if (debug)
    printf("I/O range = [0x%x..0x%x>\n", iogap_low, iogap_high);

for (i= 0; i<nr_pcidev; i++)
{

```

```

for (j= 0; j<pcidev[i].pd_bar_nr; j++)
{
    if (pcidev[i].pd_bar[j].pb_flags & PBF_IO)
        continue;
    if (!(pcidev[i].pd_bar[j].pb_flags & PBF_INCOMPLETE))
        continue;
    size= pcidev[i].pd_bar[j].pb_size;
    if (size < PAGE_SIZE)
        size= PAGE_SIZE;
    base= memgap_high-size;
    base &= ~(u32_t)(size-1);
    if (base < memgap_low)
        panic("pci", "memory base too low", base);
    memgap_high= base;
    bar_nr= pcidev[i].pd_bar[j].pb_nr;
    reg= PCI_BAR + 4*bar_nr;
    v32= pci_attr_r32(i, reg);
    pci_attr_w32(i, reg, v32 | base);
    if (debug)
    {
        printf(
"complete_bars: allocated 0x%x size %d to %d.%d.%d, bar_%d\n",
                base, size, pcidev[i].pd_busr,
                pcidev[i].pd_dev, pcidev[i].pd_func,
                bar_nr);
    }
    pcidev[i].pd_bar[j].pb_base= base;
    pcidev[i].pd_bar[j].pb_flags &= ~PBF_INCOMPLETE;
}

io_high= iogap_high;
for (j= 0; j<pcidev[i].pd_bar_nr; j++)
{
    if (!(pcidev[i].pd_bar[j].pb_flags & PBF_IO))
        continue;
    if (!(pcidev[i].pd_bar[j].pb_flags & PBF_INCOMPLETE))
        continue;
    size= pcidev[i].pd_bar[j].pb_size;
    base= iogap_high-size;
    base &= ~(u32_t)(size-1);

    /* Assume that ISA compatibility is required. Only
     * use the lowest 256 bytes out of every 1024 bytes.
     */
    base &= 0xfcff;

    if (base < iogap_low)
        panic("pci", "I/O base too low", base);

    iogap_high= base;
    bar_nr= pcidev[i].pd_bar[j].pb_nr;
    reg= PCI_BAR + 4*bar_nr;
    v32= pci_attr_r32(i, reg);
    pci_attr_w32(i, reg, v32 | base);
    if (debug)
    {
        printf(
"complete_bars: allocated 0x%x size %d to %d.%d.%d, bar_%d\n",
                base, size, pcidev[i].pd_busr,
                pcidev[i].pd_dev, pcidev[i].pd_func,
                bar_nr);
    }
    pcidev[i].pd_bar[j].pb_base= base;
    pcidev[i].pd_bar[j].pb_flags &= ~PBF_INCOMPLETE;
}
if (iogap_high != io_high)
{
    update_bridge4dev_io(i, iogap_high,
        io_high-iogap_high);
}
}

for (i= 0; i<nr_pcidev; i++)

```



```

    {
        for (j= 0; j<pcidev[i].pd_bar_nr; j++)
        {
            if (!(pcidev[i].pd_bar[j].pb_flags & PBF_INCOMPLETE))
                continue;
            printf("should allocate resources for device %d\n", i);
        }
    }
}

/*=====
 *                               update_bridge4dev_io                               *
 *=====*/
PRIVATE void update_bridge4dev_io(devind, io_base, io_size)
int devind;
u32_t io_base;
u32_t io_size;
{
    int busnr, busind, type, br_devind;
    ul6_t vl6;

    busnr= pcidev[devind].pd_busnr;
    busind= get_busind(busnr);
    type= pcibus[busind].pb_type;
    if (type == PBT_INTEL_HOST)
        return; /* Nothing to do for host controller */
    if (type == PBT_PCIBRIDGE)
    {
        printf(
            "update_bridge4dev_io: not implemented for PCI bridges\n" );
        return;
    }
    if (type != PBT_CARDBUS)
        panic("pci", "update_bridge4dev_io: strange bus type", type);

    if (debug)
    {
        printf("update_bridge4dev_io: adding 0x%x at 0x%x\n",
            io_size, io_base);
    }
    br_devind= pcibus[busind].pb_devind;
    pci_attr_w32(br_devind, CBB_IOLIMIT_0, io_base+io_size-1);
    pci_attr_w32(br_devind, CBB_IOBASE_0, io_base);

    /* Enable I/O access. Enable busmaster access as well. */
    vl6= pci_attr_rl6(devind, PCI_CR);
    pci_attr_wl6(devind, PCI_CR, vl6 | PCI_CR_IO_EN | PCI_CR_MAST_EN);
}

/*=====
 *                               get_freebus                               *
 *=====*/
PRIVATE int get_freebus()
{
    int i, freebus;

    freebus= 1;
    for (i= 0; i<nr_pcibus; i++)
    {
        if (pcibus[i].pb_needinit)
            continue;
        if (pcibus[i].pb_type == PBT_INTEL_HOST)
            continue;
        if (pcibus[i].pb_busnr <= freebus)
            freebus= pcibus[i].pb_busnr+1;
        printf("get_freebus: should check subordinate bus number\n" );
    }
    return freebus;
}

/*=====
 *                               do_isabridge                               *
 *=====*/
PRIVATE int do_isabridge(busind)

```

```

int busind;
{
    int i, j, r, type, busnr, unknown_bridge, bridge_dev;
    u16_t vid, did;
    u32_t t3;
    char *dstr;

    unknown_bridge= -1;
    bridge_dev= -1;
    j= 0; /* lint */
    vid= did= 0; /* lint */
    busnr= pcibus[busind].pb_busnr;
    for (i= 0; i< nr_pcidev; i++)
    {
        if (pcidev[i].pd_busnr != busnr)
            continue;
        t3= ((pcidev[i].pd_baseclass << 16) |
            (pcidev[i].pd_subclass << 8) | pcidev[i].pd_infclass);
        if (t3 == PCI_T3_ISA)
        {
            /* ISA bridge. Report if no supported bridge is
             * found.
             */
            unknown_bridge= i;
        }

        vid= pcidev[i].pd_vid;
        did= pcidev[i].pd_did;
        for (j= 0; pci_isabridge[j].vid != 0; j++)
        {
            if (pci_isabridge[j].vid != vid)
                continue;
            if (pci_isabridge[j].did != did)
                continue;
            if (pci_isabridge[j].checkclass &&
                unknown_bridge != i)
            {
                /* This part of multifunction device is
                 * not the bridge.
                 */
                continue;
            }
            break;
        }
        if (pci_isabridge[j].vid)
        {
            bridge_dev= i;
            break;
        }
    }

    if (bridge_dev != -1)
    {
        dstr= pci_dev_name(vid, did);
        if (!dstr)
            dstr= "unknown device";
        if (debug)
        {
            printf("found ISA bridge (%04X/%04X) %s\n",
                vid, did, dstr);
        }
        pcibus[busind].pb_isabridge_dev= bridge_dev;
        type= pci_isabridge[j].type;
        pcibus[busind].pb_isabridge_type= type;
        switch(type)
        {
            case PCI_IB_PIIIX:
                r= do_piix(bridge_dev);
                break;
            case PCI_IB_VIA:
                r= do_via_isabr(bridge_dev);
                break;
            case PCI_IB_AMD:
                r= do_amd_isabr(bridge_dev);

```

```

        break;
    case PCI_IB_SIS:
        r= do_sis_isabr(bridge_dev);
        break;
    default:
        panic("PCI", "unknown ISA bridge type", type);
    }
    return r;
}

if (unknown_bridge == -1)
{
    if (debug)
    {
        printf("(warning) no ISA bridge found on bus %d\n",
            busind);
    }
    return 0;
}
if (debug)
{
    printf(
        "(warning) unsupported ISA bridge %04X/%04X for bus %d\n",
        pcidev[unknown_bridge].pd_vid,
        pcidev[unknown_bridge].pd_did, busind);
}
return 0;
}

/*=====
 *                               do_pcibridge                               *
 *=====*/
PRIVATE void do_pcibridge(busind)
int busind;
{
    int i, devind, busnr;
    int ind, type;
    ul6_t vid, did;
    u8_t sbusn, baseclass, subclass, infclass, headt;
    u32_t t3;

    vid= did= 0;    /* lint */
    busnr= pcibus[busind].pb_busnr;
    for (devind= 0; devind< nr_pcidev; devind++)
    {
        #if 0
            printf("do_pcibridge: trying %u.%u.%u\n",
                pcidev[devind].pd_busind, pcidev[devind].pd_dev,
                pcidev[devind].pd_func);
        #endif

        if (pcidev[devind].pd_busnr != busnr)
        {
            #if 0
                printf("wrong bus\n");
            #endif
            continue;
        }

        vid= pcidev[devind].pd_vid;
        did= pcidev[devind].pd_did;
        for (i= 0; pci_pcibridge[i].vid != 0; i++)
        {
            if (pci_pcibridge[i].vid != vid)
                continue;
            if (pci_pcibridge[i].did != did)
                continue;
            break;
        }
        type= pci_pcibridge[i].type;
        if (pci_pcibridge[i].vid == 0)
        {
            headt= pci_attr_r8(devind, PCI_HEADT);
            type= 0;

```

```

        if ((headt & PHT_MASK) == PHT_BRIDGE)
            type= PCI_PPB_STD;
        else if ((headt & PHT_MASK) == PHT_CARDBUS)
            type= PCI_PPB_CB;
        else
        {
            printf("not a bridge\n");

            continue; /* Not a bridge */
        }

        baseclass= pci_attr_r8(devind, PCI_BCR);
        subclass= pci_attr_r8(devind, PCI_SCR);
        infclass= pci_attr_r8(devind, PCI_PIFR);
        t3= ((baseclass << 16) | (subclass << 8) | infclass);
        if (type == PCI_PPB_STD &&
            t3 != PCI_T3_PCI2PCI &&
            t3 != PCI_T3_PCI2PCI_SUBTR)
        {
            printf(
"Unknown PCI class %02x:%02x:%02x for PCI-to-PCI bridge, device %04X/%04X\n",
                    baseclass, subclass, infclass,
                    vid, did);

            continue;
        }
        if (type == PCI_PPB_CB &&
            t3 != PCI_T3_CARDBUS)
        {
            printf(
"Unknown PCI class %02x:%02x:%02x for Cardbus bridge, device %04X/%04X\n",
                    baseclass, subclass, infclass,
                    vid, did);

            continue;
        }
    }

    if (debug)
    {
        printf("%u.%u.%u: PCI-to-PCI bridge: %04X/%04X\n",
            pcidev[devind].pd_busr,
            pcidev[devind].pd_dev,
            pcidev[devind].pd_func, vid, did);
    }

    /* Assume that the BIOS initialized the secondary bus
     * number.
     */
    sbusn= pci_attr_r8(devind, PPB_SECBN);

    if (DEBUG)
        printf("sbusn = %d\n", sbusn);
    printf("subordn = %d\n", pci_attr_r8(devind, PPB_SUBORDBN));

    if (nr_pcibus >= NR_PCIBUS)
        panic("PCI", "too many PCI busses", nr_pcibus);
    ind= nr_pcibus;
    nr_pcibus++;
    pcibus[ind].pb_type= PBT_PCIBRIDGE;
    pcibus[ind].pb_needinit= 1;
    pcibus[ind].pb_isabridge_dev= -1;
    pcibus[ind].pb_isabridge_type= 0;
    pcibus[ind].pb_devind= devind;
    pcibus[ind].pb_busr= sbusn;
    pcibus[ind].pb_rreg8= pcibus[busind].pb_rreg8;
    pcibus[ind].pb_rreg16= pcibus[busind].pb_rreg16;
    pcibus[ind].pb_rreg32= pcibus[busind].pb_rreg32;
    pcibus[ind].pb_wreg8= pcibus[busind].pb_wreg8;
    pcibus[ind].pb_wreg16= pcibus[busind].pb_wreg16;
    pcibus[ind].pb_wreg32= pcibus[busind].pb_wreg32;
    switch(type)
    {
        case PCI_PPB_STD:
            pcibus[ind].pb_rsts= pcibr_std_rsts;

```

```

        pcibus[ind].pb_wsts= pcibr_std_wsts;
        break;
    case PCI_PPB_CB:
        pcibus[ind].pb_type= PBT_CARDBUS;
        pcibus[ind].pb_rsts= pcibr_cb_rsts;
        pcibus[ind].pb_wsts= pcibr_cb_wsts;
        break;
    case PCI_AGPB_VIA:
        pcibus[ind].pb_rsts= pcibr_via_rsts;
        pcibus[ind].pb_wsts= pcibr_via_wsts;
        break;
    default:
        panic("PCI", "unknown PCI-PCI bridge type", type);
    }
    if (sbusn == 0)
    {
        printf("Secondary bus number not initialized\n");
        continue;
    }
    pcibus[ind].pb_needinit= 0;

    probe_bus(ind);

    /* Look for PCI bridges */
    do_pcibridge(ind);
}

/*=====
 *                               get_busind                               *
 *=====*/
PRIVATE int get_busind(busnr)
int busnr;
{
    int i;

    for (i= 0; i<nr_pcibus; i++)
    {
        if (pcibus[i].pb_busnr == busnr)
            return i;
    }
    panic("pci", "get_busind: can't find bus", busnr);
}

/*=====
 *                               do_piix                               *
 *=====*/
PRIVATE int do_piix(devind)
int devind;
{
    int i, s, dev, func, irqrc, irq;
    u32_t elcr1, elcr2, elcr;

#ifdef DEBUG
    printf("in piix\n");
#endif
    dev= pcidev[devind].pd_dev;
    func= pcidev[devind].pd_func;
#ifdef USER_SPACE
    if (OK != (s=sys_inb(PIIX_ELCR1, &elcr1)))
        printf("Warning, sys_inb failed: %d\n", s);
    if (OK != (s=sys_inb(PIIX_ELCR2, &elcr2)))
        printf("Warning, sys_inb failed: %d\n", s);
#else
    elcr1= inb(PIIX_ELCR1);
    elcr2= inb(PIIX_ELCR2);
#endif
    elcr= elcr1 | (elcr2 << 8);
    for (i= 0; i<4; i++)
    {
        irqrc= pci_attr_r8(devind, PIIIX_PIRQCA+i);
        if (irqrc & PIIIX_IRQ_DI)
        {
            if (debug)

```

```

        printf("INT%c: disabled\n", 'A'+i);
    }
    else
    {
        irq= irqrc & PIIX_IRQ_MASK;
        if (debug)
            printf("INT%c: %d\n", 'A'+i, irq);
        if (!(elcr & (1 << irq)))
        {
            if (debug)
            {
                printf(
                    "(warning) IRQ %d is not level triggered\n",
                    irq);
            }
        }
        irq_mode_pci(irq);
    }
}
return 0;
}

/*=====
*                               do_amd_isabr                               *
*=====*/
PRIVATE int do_amd_isabr(devind)
int devind;
{
    int i, busnr, dev, func, xdevind, irq, edge;
    u8_t levmask;
    u16_t pciirq;

    /* Find required function */
    func= AMD_ISABR_FUNC;
    busnr= pcidev[devind].pd_busnr;
    dev= pcidev[devind].pd_dev;

    /* Fake a device with the required function */
    if (nr_pcidev >= NR_PCIDEV)
        panic("PCI", "too many PCI devices", nr_pcidev);
    xdevind= nr_pcidev;
    pcidev[xdevind].pd_busnr= busnr;
    pcidev[xdevind].pd_dev= dev;
    pcidev[xdevind].pd_func= func;
    pcidev[xdevind].pd_inuse= 1;
    nr_pcidev++;

    levmask= pci_attr_r8(xdevind, AMD_ISABR_PCIIRQ_LEV);
    pciirq= pci_attr_r16(xdevind, AMD_ISABR_PCIIRQ_ROUTE);
    for (i= 0; i<4; i++)
    {
        edge= (levmask >> i) & 1;
        irq= (pciirq >> (4*i)) & 0xf;
        if (!irq)
        {
            if (debug)
                printf("INT%c: disabled\n", 'A'+i);
        }
        else
        {
            if (debug)
                printf("INT%c: %d\n", 'A'+i, irq);
            if (edge && debug)
            {
                printf(
                    "(warning) IRQ %d is not level triggered\n",
                    irq);
            }
            irq_mode_pci(irq);
        }
    }
    nr_pcidev--;
    return 0;
}

```

```

/*=====
 *
 *                               do_sis_isabr
 *=====*/
PRIVATE int do_sis_isabr(devind)
int devind;
{
    int i, dev, func, irq;

    dev= pcidev[devind].pd_dev;
    func= pcidev[devind].pd_func;
    irq= 0; /* lint */
    for (i= 0; i<4; i++)
    {
        irq= pci_attr_r8(devind, SIS_ISABR_IRQ_A+i);
        if (irq & SIS_IRQ_DISABLED)
        {
            if (debug)
                printf("INT%c: disabled\n", 'A'+i);
        }
        else
        {
            irq &= SIS_IRQ_MASK;
            if (debug)
                printf("INT%c: %d\n", 'A'+i, irq);
            irq_mode_pci(irq);
        }
    }
    return 0;
}

/*=====
 *
 *                               do_via_isabr
 *=====*/
PRIVATE int do_via_isabr(devind)
int devind;
{
    int i, dev, func, irq, edge;
    u8_t levmask;

    dev= pcidev[devind].pd_dev;
    func= pcidev[devind].pd_func;
    levmask= pci_attr_r8(devind, VIA_ISABR_EL);
    irq= 0; /* lint */
    edge= 0; /* lint */
    for (i= 0; i<4; i++)
    {
        switch(i)
        {
            case 0:
                edge= (levmask & VIA_ISABR_EL_INTA);
                irq= pci_attr_r8(devind, VIA_ISABR_IRQ_R2) >> 4;
                break;
            case 1:
                edge= (levmask & VIA_ISABR_EL_INTB);
                irq= pci_attr_r8(devind, VIA_ISABR_IRQ_R2);
                break;
            case 2:
                edge= (levmask & VIA_ISABR_EL_INTC);
                irq= pci_attr_r8(devind, VIA_ISABR_IRQ_R3) >> 4;
                break;
            case 3:
                edge= (levmask & VIA_ISABR_EL_INTD);
                irq= pci_attr_r8(devind, VIA_ISABR_IRQ_R1) >> 4;
                break;
            default:
                assert(0);
        }
        irq &= 0xf;
        if (!irq)
        {
            if (debug)
                printf("INT%c: disabled\n", 'A'+i);
        }
    }
}

```

```

        else
        {
            if (debug)
                printf("INT%c: %d\n", 'A'+i, irq);
            if (edge && debug)
            {
                printf(
                    "(warning) IRQ %d is not level triggered\n",
                    irq);
            }
            irq_mode_pci(irq);
        }
    }
    return 0;
}

/*=====
 *                               report_vga                               *
 *=====*/
PRIVATE void report_vga(devind)
int devind;
{
    /* Report the amount of video memory. This is needed by the X11R6
     * postinstall script to chmem the X server. Hopefully this can be
     * removed when we get virtual memory.
     */
    size_t amount, size;
    int i;

    amount= 0;
    for (i= 0; i<pcidev[devind].pd_bar_nr; i++)
    {
        if (pcidev[devind].pd_bar[i].pb_flags & PBF_IO)
            continue;
        size= pcidev[devind].pd_bar[i].pb_size;
        if (size < amount)
            continue;
        amount= size;
    }
    if (size != 0)
    {
        printf("PCI: video memory for device at %d.%d.%d: %d bytes\n",
            pcidev[devind].pd_busr,
            pcidev[devind].pd_dev,
            pcidev[devind].pd_func,
            amount);
    }
}

/*=====
 *                               pci_vid_name                               *
 *=====*/
PRIVATE char *pci_vid_name(vid)
ul6_t vid;
{
    int i;

    for (i= 0; pci_vendor_table[i].name; i++)
    {
        if (pci_vendor_table[i].vid == vid)
            return pci_vendor_table[i].name;
    }
    return "unknown";
}

/*=====
 *                               pci_baseclass_name                               *
 *=====*/
PRIVATE char *pci_baseclass_name(baseclass)
u8_t baseclass;
{
    int i;

```



```

    for (i= 0; pci_baseclass_table[i].name; i++)
    {
        if (pci_baseclass_table[i].baseclass == baseclass)
            return pci_baseclass_table[i].name;
    }
    return NULL;
}

/*=====
 *                               pci_subclass_name                               *
 *=====*/
PRIVATE char *pci_subclass_name(baseclass, subclass, infclass)
u8_t baseclass;
u8_t subclass;
u8_t infclass;
{
    int i;

    for (i= 0; pci_subclass_table[i].name; i++)
    {
        if (pci_subclass_table[i].baseclass != baseclass)
            continue;
        if (pci_subclass_table[i].subclass != subclass)
            continue;
        if (pci_subclass_table[i].infclass != infclass &&
            pci_subclass_table[i].infclass != (u16_t)-1)
        {
            continue;
        }
        return pci_subclass_table[i].name;
    }
    return NULL;
}

/*=====
 *                               ntostr                               *
 *=====*/
PRIVATE void ntostr(n, str, end)
unsigned n;
char **str;
char *end;
{
    char tmpstr[20];
    int i;

    if (n == 0)
    {
        tmpstr[0]= '0';
        i= 1;
    }
    else
    {
        for (i= 0; n; i++)
        {
            tmpstr[i]= '0' + (n%10);
            n /= 10;
        }
    }
    for (; i>0; i--)
    {
        if (*str == end)
        {
            break;
        }
        **str= tmpstr[i-1];
        (*str)++;
    }
    if (*str == end)
        end[-1]= '\0';
    else
        **str= '\0';
}

```

```

/*=====
 *                               pci_attr_rsts                               *
 *=====*/
PRIVATE ul6_t pci_attr_rsts(devind)
int devind;
{
    int busnr, busind;

    busnr= pcidev[devind].pd_busnr;
    busind= get_busind(busnr);
    return pcibus[busind].pb_rsts(busind);
}

/*=====
 *                               pcibr_std_rsts                               *
 *=====*/
PRIVATE ul6_t pcibr_std_rsts(busind)
int busind;
{
    int devind;

    devind= pcibus[busind].pb_devind;
    return pci_attr_r16(devind, PPB_SSTS);
}

/*=====
 *                               pcibr_std_wsts                               *
 *=====*/
PRIVATE void pcibr_std_wsts(busind, value)
int busind;
ul6_t value;
{
    int devind;
    devind= pcibus[busind].pb_devind;

#if 0
    printf("pcibr_std_wsts(%d, 0x%X), devind= %d\n",
           busind, value, devind);
#endif
    pci_attr_w16(devind, PPB_SSTS, value);
}

/*=====
 *                               pcibr_cb_rsts                               *
 *=====*/
PRIVATE ul6_t pcibr_cb_rsts(busind)
int busind;
{
    int devind;
    devind= pcibus[busind].pb_devind;

    return pci_attr_r16(devind, CBB_SSTS);
}

/*=====
 *                               pcibr_cb_wsts                               *
 *=====*/
PRIVATE void pcibr_cb_wsts(busind, value)
int busind;
ul6_t value;
{
    int devind;
    devind= pcibus[busind].pb_devind;

#if 0
    printf("pcibr_cb_wsts(%d, 0x%X), devind= %d\n",
           busind, value, devind);
#endif
    pci_attr_w16(devind, CBB_SSTS, value);
}

/*=====
 *                               pcibr_via_rsts                               *
 *=====*/

```

```

/*=====*/
PRIVATE ul6_t pcibr_via_rsts(busind)
int busind;
{
    int devind;
    devind= pcibus[busind].pb_devind;

    return 0;
}

/*=====
*                               pcibr_via_wsts                               *
*=====*/
PRIVATE void pcibr_via_wsts(busind, value)
int busind;
ul6_t value;
{
    int devind;
    devind= pcibus[busind].pb_devind;

#if 0
    printf("pcibr_via_wsts(%d, 0x%X), devind= %d (not implemented)\n",
           busind, value, devind);
#endif
}

/*=====
*                               pci_attr_wsts                               *
*=====*/
PRIVATE void pci_attr_wsts(devind, value)
int devind;
ul6_t value;
{
    int busnr, busind;

    busnr= pcidev[devind].pd_busnr;
    busind= get_busind(busnr);
    pcibus[busind].pb_wsts(busind, value);
}

/*=====
*                               pcii_rreg8                               *
*=====*/
PRIVATE u8_t pcii_rreg8(busind, devind, port)
int busind;
int devind;
int port;
{
    u8_t v;
    int s;

    v= PCII_RREG8(pcibus[busind].pb_busnr,
                  pcidev[devind].pd_dev, pcidev[devind].pd_func,
                  port);
#if USER_SPACE
    if (OK != (s=sys_outl(PCII_CONFADD, PCII_UNSEL)))
        printf("PCI: warning, sys_outl failed: %d\n", s);
#else
    outl(PCII_CONFADD, PCII_UNSEL);
#endif
#if 0
    printf("pcii_rreg8(%d, %d, 0x%X): %d.%d.%d= 0x%X\n",
           busind, devind, port,
           pcibus[busind].pb_bus, pcidev[devind].pd_dev,
           pcidev[devind].pd_func, v);
#endif
    return v;
}

/*=====
*                               pcii_rreg16                               *
*=====*/
PRIVATE ul6_t pcii_rreg16(busind, devind, port)

```

```

int busind;
int devind;
int port;
{
    ul6_t v;
    int s;

    v= PCII_RREG16_(pcibus[busind].pb_busnr,
        pcidev[devind].pd_dev, pcidev[devind].pd_func,
        port);
#if USER_SPACE
    if (OK != (s=sys_outl(PCII_CONFADD, PCII_UNSEL)))
        printf("PCI: warning, sys_outl failed: %d\n" );
#else
    outl(PCII_CONFADD, PCII_UNSEL);
#endif
#if 0
    printf("pcii_rreg16(%d, %d, 0x%X): %d.%d.%d= 0x%X\n",
        busind, devind, port,
        pcibus[busind].pb_bus, pcidev[devind].pd_dev,
        pcidev[devind].pd_func, v);
#endif
    return v;
}

/*=====
*                                     pcii_rreg32                                     *
*=====*/
PRIVATE u32_t pcii_rreg32(busind, devind, port)
int busind;
int devind;
int port;
{
    u32_t v;
    int s;

    v= PCII_RREG32_(pcibus[busind].pb_busnr,
        pcidev[devind].pd_dev, pcidev[devind].pd_func,
        port);
#if USER_SPACE
    if (OK != (s=sys_outl(PCII_CONFADD, PCII_UNSEL)))
        printf("PCI: warning, sys_outl failed: %d\n", s);
#else
    outl(PCII_CONFADD, PCII_UNSEL);
#endif
#if 0
    printf("pcii_rreg32(%d, %d, 0x%X): %d.%d.%d= 0x%X\n",
        busind, devind, port,
        pcibus[busind].pb_bus, pcidev[devind].pd_dev,
        pcidev[devind].pd_func, v);
#endif
    return v;
}

/*=====
*                                     pcii_wreg8                                     *
*=====*/
PRIVATE void pcii_wreg8(busind, devind, port, value)
int busind;
int devind;
int port;
u8_t value;
{
    int s;
#if 0
    printf("pcii_wreg8(%d, %d, 0x%X, 0x%X): %d.%d.%d\n",
        busind, devind, port, value,
        pcibus[busind].pb_bus, pcidev[devind].pd_dev,
        pcidev[devind].pd_func);
#endif
    PCII_WREG8_(pcibus[busind].pb_busnr,
        pcidev[devind].pd_dev, pcidev[devind].pd_func,
        port, value);
#if USER_SPACE

```

```

        if (OK != (s=sys_outl(PCII_CONFADD, PCII_UNSEL)))
            printf("PCI: warning, sys_outl failed: %d\n", s);
#else
    outl(PCII_CONFADD, PCII_UNSEL);
#endif
}

/*=====
 *                               pcii_wreg16                               *
 *=====*/
PRIVATE void pcii_wreg16(busind, devind, port, value)
int busind;
int devind;
int port;
ul6_t value;
{
    int s;
#if 0
    printf("pcii_wreg16(%d, %d, 0x%X, 0x%X): %d.%d.%d\n",
        busind, devind, port, value,
        pcibus[busind].pb_bus, pcidev[devind].pd_dev,
        pcidev[devind].pd_func);
#endif
    PCII_WREG16_(pcibus[busind].pb_busnr,
        pcidev[devind].pd_dev, pcidev[devind].pd_func,
        port, value);
#if USER_SPACE
    if (OK != (s=sys_outl(PCII_CONFADD, PCII_UNSEL)))
        printf("PCI: warning, sys_outl failed: %d\n", s);
#else
    outl(PCII_CONFADD, PCII_UNSEL);
#endif
}

/*=====
 *                               pcii_wreg32                               *
 *=====*/
PRIVATE void pcii_wreg32(busind, devind, port, value)
int busind;
int devind;
int port;
u32_t value;
{
    int s;
#if 0
    printf("pcii_wreg32(%d, %d, 0x%X, 0x%X): %d.%d.%d\n",
        busind, devind, port, value,
        pcibus[busind].pb_bus, pcidev[devind].pd_dev,
        pcidev[devind].pd_func);
#endif
    PCII_WREG32_(pcibus[busind].pb_busnr,
        pcidev[devind].pd_dev, pcidev[devind].pd_func,
        port, value);
#if USER_SPACE
    if (OK != (s=sys_outl(PCII_CONFADD, PCII_UNSEL)))
        printf("PCI: warning, sys_outl failed: %d\n", s);
#else
    outl(PCII_CONFADD, PCII_UNSEL);
#endif
}

/*=====
 *                               pcii_rsts                               *
 *=====*/
PRIVATE ul6_t pcii_rsts(busind)
int busind;
{
    ul6_t v;
    int s;

    v= PCII_RREG16_(pcibus[busind].pb_busnr, 0, 0, PCI_SR);
#if USER_SPACE
    if (OK != (s=sys_outl(PCII_CONFADD, PCII_UNSEL)))
        printf("PCI: warning, sys_outl failed: %d\n", s);

```

```

#else
    outl(PCII_CONFADD, PCII_UNSEL);
#endif
    return v;
}

/*=====
 *                               pcii_wsts                               *
 *=====*/
PRIVATE void pcii_wsts(busind, value)
int busind;
ul6_t value;
{
    int s;
    PCII_WREG16(pcibus[busind].pb_busnr, 0, 0, PCI_SR, value);
    #if USER_SPACE
        if (OK != (s=sys_outl(PCII_CONFADD, PCII_UNSEL)))
            printf("PCI: warning, sys_outl failed: %d\n", s);
    #else
        outl(PCII_CONFADD, PCII_UNSEL);
    #endif
}

/*=====
 *                               print_capabilities                       *
 *=====*/
PRIVATE void print_capabilities(devind)
int devind;
{
    u8_t status, capptr, type, next;
    char *str;

    /* Check capabilities bit in the device status register */
    status= pci_attr_rl6(devind, PCI_SR);
    if (!(status & PSR_CAPPTR))
        return;

    capptr= (pci_attr_r8(devind, PCI_CAPPTR) & PCI_CP_MASK);
    while (capptr != 0)
    {
        type = pci_attr_r8(devind, capptr+CAP_TYPE);
        next= (pci_attr_r8(devind, capptr+CAP_NEXT) & PCI_CP_MASK);
        switch(type)
        {
            case 1: str= "PCI Power Management"; break;
            case 2: str= "AGP"; break;
            case 3: str= "Vital Product Data"; break;
            case 4: str= "Slot Identification"; break;
            case 5: str= "Message Signaled Interrupts"; break;
            case 6: str= "CompactPCI Hot Swap"; break;
            case 8: str= "AMD HyperTransport"; break;
            case 0xf: str= "AMD I/O MMU"; break;
            default: str= "(unknown type)"; break;
        }

        printf(" @0x%x: capability type 0x%x: %s\n",
            capptr, type, str);
        capptr= next;
    }
}

/*
 * $PchId: pci.c,v 1.7 2003/08/07 09:06:51 philip Exp $
 */

```

```
/*
pci.h

Created:      Jan 2000 by Philip Homburg <philip@cs.vu.nl>
*/

/* temporary functions: to be replaced later (see pci_intel.h) */
_PROTOTYPE( unsigned pci_inb, (U16_t port) );
_PROTOTYPE( unsigned pci_inw, (U16_t port) );
_PROTOTYPE( unsigned pci_inl, (U16_t port) );

_PROTOTYPE( void pci_outb, (U16_t port, U8_t value) );
_PROTOTYPE( void pci_outw, (U16_t port, U16_t value) );
_PROTOTYPE( void pci_outl, (U16_t port, U32_t value) );

struct pci_vendor
{
    u16_t vid;
    char *name;
};

struct pci_device
{
    u16_t vid;
    u16_t did;
    char *name;
};

struct pci_baseclass
{
    u8_t baseclass;
    char *name;
};

struct pci_subclass
{
    u8_t baseclass;
    u8_t subclass;
    u16_t infclass;
    char *name;
};

struct pci_intel_ctrl
{
    u16_t vid;
    u16_t did;
};

struct pci_isabridge
{
    u16_t vid;
    u16_t did;
    int checkclass;
    int type;
};

struct pci_pcibridge
{
    u16_t vid;
    u16_t did;
    int type;
};

#define PCI_IB_PIIIX      1      /* Intel PIIIX compatible ISA bridge */
#define PCI_IB_VIA        2      /* VIA compatible ISA bridge */
#define PCI_IB_AMD        3      /* AMD compatible ISA bridge */
#define PCI_IB_SIS        4      /* SIS compatible ISA bridge */

#define PCI_PPB_STD       1      /* Standard PCI-to-PCI bridge */
#define PCI_PPB_CB        2      /* Cardbus bridge */
/* Still needed? */
#define PCI_AGPB_VIA      3      /* VIA compatible AGP bridge */

extern struct pci_vendor pci_vendor_table[];
```

```
extern struct pci_device pci_device_table[];
extern struct pci_baseclass pci_baseclass_table[];
extern struct pci_subclass pci_subclass_table[];
#if 0
extern struct pci_intel_ctrl pci_intel_ctrl[];
#endif
extern struct pci_isabridge pci_isabridge[];
extern struct pci_pcibridge pci_pcibridge[];

/* Utility functions */
_PROTOTYPE( void pci_reserve3, (int devind, int proc, char name[M3_STRING]));
_PROTOTYPE( void pci_release, (char name[M3_STRING]) );

/*
 * $PchId: pci.h,v 1.4 2001/12/06 20:21:22 philip Exp $
 */
```



```
/*
pci_amd.h

Created:      Nov 2001 by Philip Homburg <philip@cs.vu.nl>
*/

#define AMD_ISABR_FUNC 3          /* Registers are in function 3 */
#define AMD_ISABR_PCIIRQ_LEV    0x54
#define AMD_PCILEV_INTA         0x1
#define AMD_PCILEV_INTB         0x2
#define AMD_PCILEV_INTC         4x2
#define AMD_PCILEV_INTD         4x8
#define AMD_ISABR_PCIIRQ_ROUTE  0x56
#define AMD_PCIIRQ_INTA_MASK    0x000F
#define AMD_PCIIRQ_INTB_MASK    0x00F0
#define AMD_PCIIRQ_INTC_MASK    0x0F00
#define AMD_PCIIRQ_INTD_MASK    0xF000

/*
 * $PchId: pci_amd.h,v 1.1 2001/11/09 19:57:37 philip Exp $
 */
```

```

/*
pci_intel.h

Created:      Jan 2000 by Philip Homburg <philip@cs.vu.nl>
*/

#define PCII_CONFADD      0xCF8
#define PCIIC_CODE        0x80000000
#define PCIIC_BUSNUM_MASK  0xff0000
#define PCIIC_BUSNUM_SHIFT 16
#define PCIIC_DEVNUM_MASK  0xf800
#define PCIIC_DEVNUM_SHIFT 11
#define PCIIC_FUNCNUM_MASK 0x700
#define PCIIC_FUNCNUM_SHIFT 8
#define PCIIC_REGNUM_MASK  0xfc
#define PCIIC_REGNUM_SHIFT 2
#define PCII_CONFDATA     0xCFC

#define PCII_SELREG_(bus, dev, func, reg) \
    (PCIIC_CODE | \
      (((bus) << PCIIC_BUSNUM_SHIFT) & PCIIC_BUSNUM_MASK) | \
      (((dev) << PCIIC_DEVNUM_SHIFT) & PCIIC_DEVNUM_MASK) | \
      (((func) << PCIIC_FUNCNUM_SHIFT) & PCIIC_FUNCNUM_MASK) | \
      (((reg)/4) << PCIIC_REGNUM_SHIFT) & PCIIC_REGNUM_MASK))

#define PCII_UNSEL      (0)

#define PCII_RREG8_(bus, dev, func, reg) \
    (pci_outl(PCII_CONFADD, PCII_SELREG_(bus, dev, func, reg)), \
     pci_inb(PCII_CONFDATA+((reg)&3)))

#define PCII_RREG16_(bus, dev, func, reg) \
    (PCII_RREG8_(bus, dev, func, reg) | \
     (PCII_RREG8_(bus, dev, func, reg+1) << 8))

#define PCII_RREG32_(bus, dev, func, reg) \
    (PCII_RREG16_(bus, dev, func, reg) | \
     (PCII_RREG16_(bus, dev, func, reg+2) << 16))

#define PCII_WREG8_(bus, dev, func, reg, val) \
    (pci_outl(PCII_CONFADD, PCII_SELREG_(bus, dev, func, reg)), \
     pci_outb(PCII_CONFDATA+((reg)&3), (val)))

#define PCII_WREG16_(bus, dev, func, reg, val) \
    (PCII_WREG8_(bus, dev, func, reg, (val)), \
     (PCII_WREG8_(bus, dev, func, reg+1, (val) >> 8)))

#define PCII_WREG32_(bus, dev, func, reg, val) \
    (PCII_WREG16_(bus, dev, func, reg, (val)), \
     (PCII_WREG16_(bus, dev, func, reg+2, (val) >> 16)))

/* PIIX configuration registers */
#define PIIX_PIRQCA      0x60
#define PIIX_IRQ_DI      0x80
#define PIIX_IRQ_MASK    0x0F

/* PIIX extensions to the PIC */
#define PIIX_ELCR1      0x4D0
#define PIIX_ELCR2      0x4D1

/*
 * $PchId: pci_intel.h,v 1.1 2000/08/12 11:20:17 philip Exp $
 */

```

```
/*
pci_sis.h

Created:      Nov 2001 by Philip Homburg <philip@cs.vu.nl>
*/

/* Constants are taken from pci-irq.c in the Linux kernel source */
#define SIS_ISABR_IRQ_A 0x41    /* IRQA routing */
#define SIS_ISABR_IRQ_B 0x42    /* IRQB routing */
#define SIS_ISABR_IRQ_C 0x43    /* IRQC routing */
#define SIS_ISABR_IRQ_D 0x44    /* IRQD routing */
#define SIS_IRQ_DISABLED 0x80
#define SIS_IRQ_MASK 0x0F

/*
 * $PchId: pci_sis.h,v 1.1 2001/12/06 20:22:52 philip Exp $
 */
```

```
/*
pci_table.c
```

Tables with PCI vendor and device identifiers

Created: Jan 2000 by Philip Homburg <philip@cs.vu.nl>

See the Linux PCI ID Repository <<http://pciids.sourceforge.net/>>.

```
*/
/* Changes from original Minix 2.0.4 version (2003-09-05):
 * 2003-11-30 (kjb) Minix 2.0.4 FIX.TAZ add D-Link RTL8139 (0x1186, 0x1300)
 * 2004-08-08 (asw) add Intel 82371AB (0x8086, 0x7100)
 */
```

```
#include "../drivers.h"
#include "pci.h"
#if __minix_vmd
#include "config.h"
#endif
```

```
struct pci_vendor pci_vendor_table[] =
{
    { 0x1000, "NCR" },
    { 0x1002, "ATI Technologies" },
    { 0x100B, "National Semiconductor Corporation" },
    { 0x1013, "Cirrus Logic" },
    { 0x1022, "Advanced Micro Devices" },
    { 0x102B, "Matrox Graphics, Inc." },
    { 0x1039, "Silicon Integrated Systems (SiS)" },
    { 0x104C, "Texas Instruments" },
    { 0x105A, "Promise Technology" },
    { 0x10B7, "3Com Corporation" },
    { 0x10B9, "AcerLabs (ALI)" },
    { 0x10C8, "Neomagic Corporation" },
    { 0x10DE, "nVidia Corporation" },
    { 0x10EC, "Realtek" },
    { 0x1106, "VIA" },
    { 0x110A, "Siemens Nixdorf AG" },
    { 0x125D, "ESS Technology" },
    { 0x1274, "Ensoniq" },
    { 0x5333, "S3" },
    { 0x8086, "Intel" },
    { 0x9004, "Adaptec" },
    { 0x9005, "Adaptec" },
    { 0x0000, NULL }
};

struct pci_device pci_device_table[] =
{
    { 0x1000, 0x0001, "NCR 53C810" },
    { 0x1000, 0x000F, "NCR 53C875" },
    { 0x1002, 0x4752, "ATI Rage XL PCI" },
    { 0x100B, 0xD001, "Nat. Semi. 87410" },
    { 0x1013, 0x00B8, "Cirrus Logic GD 5446" },
    { 0x1013, 0x6003, "Cirrus Logic CS4614/22/24 CrystalClear" },
    { 0x1022, 0x1100, "K8 HyperTransport Tech. Conf." },
    { 0x1022, 0x1101, "K8 [Athlon64/Opteron] Address Map" },
    { 0x1022, 0x1102, "K8 [Athlon64/Opteron] DRAM Controller" },
    { 0x1022, 0x1103, "K8 [Athlon64/Opteron] Misc. Control" },
    { 0x1022, 0x2000, "AMD Lance/PCI" },
    { 0x1022, 0x700C, "AMD-762 CPU to PCI Bridge (SMP chipset)" },
    { 0x1022, 0x700D, "AMD-762 CPU to PCI Bridge (AGP 4x)" },
    { 0x1022, 0x7410, "AMD-766 PCI to ISA/LPC Bridge" },
    { 0x1022, 0x7411, "AMD-766 EIDE Controller" },
    { 0x102B, 0x051B, "Matrox MGA 2164W [Millennium II]" },
    { 0x102B, 0x0525, "Matrox MGA G400 AGP" },
    { 0x1039, 0x0008, "SiS 85C503/5513" },
    { 0x1039, 0x0200, "SiS 5597/5598 VGA" },
    { 0x1039, 0x0406, "SiS 85C501/2" },
    { 0x1039, 0x5597, "SiS 5582" },
    { 0x104C, 0xAC1C, "TI PCI1225" },
    { 0x105A, 0x0D30, "Promise Technology 20265" },
    { 0x10B7, 0x9058, "3Com 3c905B-Combo" },

```

```
0x10B7, 0x9805, "3Com 3c980-TX Python-T" },
0x10B9, 0x1533, "ALI M1533 ISA-bridge [Aladdin IV]" },
0x10B9, 0x1541, "ALI M1541" },
0x10B9, 0x5229, "ALI M5229 (IDE)" },
0x10B9, 0x5243, "ALI M5243" },
0x10B9, 0x7101, "ALI M7101 PMU" },
0x10C8, 0x0005, "Neomagic NM2200 Magic Graph 256AV" },
0x10C8, 0x8005, "Neomagic NM2200 Magic Graph 256AV Audio" },
0x10DE, 0x0020, "nVidia Riva TnT [NV04]" },
0x10DE, 0x0110, "nVidia GeForce2 MX [NV11]" },
0x10EC, 0x8029, "Realtek RTL8029" },
0x10EC, 0x8139, "Realtek RTL8139" },
0x1106, 0x0305, "VIA VT8363/8365 [KT133/KM133]" },
0x1106, 0x0571, "VIA IDE controller" },
0x1106, 0x0686, "VIA VT82C686 (Apollo South Bridge)" },
0x1106, 0x1204, "K8M800 Host Bridge" },
0x1106, 0x2204, "K8M800 Host Bridge" },
0x1106, 0x3038, "VT83C572 PCI USB Controller" },
0x1106, 0x3057, "VT82C686A ACPI Power Management Controller" },
0x1106, 0x3058, "VIA AC97 Audio Controller" },
0x1106, 0x3059, "VIA AC97 Audio Controller" },
0x1106, 0x3065, "VT6102 [Rhine-II]" },
0x1106, 0x3074, "VIA VT8233" },
0x1106, 0x3099, "VIA VT8367 [KT266]" },
0x1106, 0x3104, "VIA USB 2.0" },
0x1106, 0x3108, "VIA S3 Unichrome Pro VGA Adapter" },
0x1106, 0x3149, "VIA VT6420 SATA RAID Controller" },
0x1106, 0x3204, "K8M800 Host Bridge" },
0x1106, 0x3227, "VT8237 ISA bridge" },
0x1106, 0x4204, "K8M800 Host Bridge" },
0x1106, 0x8305, "VIA VT8365 [KM133 AGP]" },
0x1106, 0xB099, "VIA VT8367 [KT266 AGP]" },
0x1106, 0xB188, "VT8237 PCI bridge" },
0x110A, 0x0005, "Siemens Nixdorf Tulip Cntrl., Power Management" },
0x1186, 0x1300, "D-Link RTL8139" },
0x125D, 0x1969, "ESS ES1969 Solo-1 Audiodrive" },
0x1274, 0x1371, "Ensoniq ES1371 [AudioPCI-97]" },
0x1274, 0x5000, "Ensoniq ES1370" },
0x1274, 0x5880, "Ensoniq CT5880 [AudioPCI]" },
0x5333, 0x8811, "S3 86c764/765 [Trio32/64/64V+]" },
0x5333, 0x883d, "S3 Virge/VX" },
0x5333, 0x88d0, "S3 Vision 964 vers 0" },
0x5333, 0x8a01, "S3 Virge/DX or /GX" },
0x8086, 0x1004, "Intel 82543GC Gigabit Ethernet Controller" },
0x8086, 0x1029, "Intel EtherExpressPro100 ID1029" },
0x8086, 0x1030, "Intel Corporation 82559 InBusiness 10/100" },
0x8086, 0x1209, "Intel EtherExpressPro100 82559ER" },
0x8086, 0x1229, "Intel EtherExpressPro100 82557/8/9" },
0x8086, 0x122D, "Intel 82437FX" },
0x8086, 0x122E, "Intel 82371FB (PIIX)" },
0x8086, 0x1230, "Intel 82371FB (IDE)" },
0x8086, 0x1237, "Intel 82441FX (440FX)" },
0x8086, 0x1250, "Intel 82439HX" },
0x8086, 0x1A30, "Intel 82845B/A MCH" },
0x8086, 0x1A31, "Intel 82845B/A PCI Bridge to AGP port" },
0x8086, 0x2440, "Intel 82801B PCI to ISA bridge" },
0x8086, 0x2449, "Intel EtherExpressPro100 82562EM" },
0x8086, 0x244e, "Intel 82801 PCI Bridge" },
0x8086, 0x2560, "Intel 82845G/GL[Brookdale-G]/GE/PE" },
0x8086, 0x2561, "Intel 82845G/GL/GE/PE Host-to-AGP Bridge" },
0x8086, 0x7000, "Intel 82371SB" },
0x8086, 0x7010, "Intel 82371SB (IDE)" },
0x8086, 0x7020, "Intel 82371SB (USB)" },
0x8086, 0x7030, "Intel 82437VX" },
0x8086, 0x7100, "Intel 82371AB" },
0x8086, 0x7100, "Intel 82371AB" },
0x8086, 0x7110, "Intel 82371AB (PIIX4)" },
0x8086, 0x7111, "Intel 82371AB (IDE)" },
0x8086, 0x7112, "Intel 82371AB (USB)" },
0x8086, 0x7113, "Intel 82371AB (Power)" },
0x8086, 0x7124, "Intel 82801AA" },
0x8086, 0x7190, "Intel 82443BX" },
0x8086, 0x7191, "Intel 82443BX (AGP bridge)" },
0x8086, 0x7192, "Intel 82443BX (Host-to-PCI bridge)" },
```

```

    { 0x9004, 0x8178, "Adaptec AHA-2940U/2940UW Ultra/Ultra-Wide SCSI Ctrlr" },
    { 0x9005, 0x0080, "Adaptec AIC-7892A Ultra160/m PCI SCSI Controller" },
    { 0x0000, 0x0000, NULL }
};

```

```

struct pci_baseclass pci_baseclass_table[] =
{
    { 0x00, "No device class" },
    { 0x01, "Mass storage controller" },
    { 0x02, "Network controller" },
    { 0x03, "Display controller" },
    { 0x04, "Multimedia device" },
    { 0x05, "Memory controller" },
    { 0x06, "Bridge device" },
    { 0x07, "Simple comm. controller" },
    { 0x08, "Base system peripheral" },
    { 0x09, "Input device" },
    { 0x0A, "Docking station" },
    { 0x0B, "Processor" },
    { 0x0C, "Serial bus controller" },
    { 0x0D, "Wireless controller" },
    { 0x0E, "Intelligent I/O controller" },
    { 0x0F, "Satellite comm. controller" },
    { 0x10, "Encryption/decryption controller" },
    { 0x11, "Data acquisition controller" },
    { 0xFF, "Misc. device" },

    { 0x00, NULL }
};

```

*/\* -1 in the infclass field is a wildcard for infclass \*/*

```

struct pci_subclass pci_subclass_table[] =
{
    { 0x00, 0x01, 0x00, "VGA-compatible device" },

    { 0x01, 0x00, 0x00, "SCSI bus controller" },
    { 0x01, 0x01, -1, "IDE controller" },
    { 0x01, 0x02, 0x00, "Floppy disk controller" },
    { 0x01, 0x03, 0x00, "IPI controller" },
    { 0x01, 0x04, 0x00, "RAID controller" },
    { 0x01, 0x80, 0x00, "Other mass storage controller" },

    { 0x02, 0x00, 0x00, "Ethernet controller" },
    { 0x02, 0x01, 0x00, "Token Ring controller" },
    { 0x02, 0x02, 0x00, "FDDI controller" },
    { 0x02, 0x03, 0x00, "ATM controller" },
    { 0x02, 0x04, 0x00, "ISDN controller" },
    { 0x02, 0x80, 0x00, "Other network controller" },

    { 0x03, 0x00, 0x00, "VGA-compatible controller" },
    { 0x03, 0x00, 0x01, "8514-compatible controller" },
    { 0x03, 0x01, 0x00, "XGA controller" },
    { 0x03, 0x02, 0x00, "3D controller" },
    { 0x03, 0x80, 0x00, "Other display controller" },

    { 0x04, 0x00, 0x00, "Video device" },
    { 0x04, 0x01, 0x00, "Audio device" },
    { 0x04, 0x02, 0x00, "Computer telephony device" },
    { 0x04, 0x80, 0x00, "Other multimedia device" },

    { 0x06, 0x00, 0x00, "Host bridge" },
    { 0x06, 0x01, 0x00, "ISA bridge" },
    { 0x06, 0x02, 0x00, "EISA bridge" },
    { 0x06, 0x03, 0x00, "MCA bridge" },
    { 0x06, 0x04, 0x00, "PCI-to-PCI bridge" },
    { 0x06, 0x04, 0x01, "Subtractive decode PCI-to-PCI bridge" },
    { 0x06, 0x05, 0x00, "PCMCIA bridge" },
    { 0x06, 0x06, 0x00, "NuBus bridge" },
    { 0x06, 0x07, 0x00, "CardBus bridge" },
    { 0x06, 0x08, -1, "RACEway bridge" },
    { 0x06, 0x09, -1, "Semi-transparent PCI-to-PCI bridge" },
    { 0x06, 0x80, 0x00, "Other bridge device" },

    { 0x0C, 0x00, 0x00, "IEEE 1394 (FireWire)" },

```

```

    { 0x0C, 0x00, 0x10, "IEEE 1394 (OpenHCI)" },
    { 0x0C, 0x01, 0x00, "ACCESS bus" },
    { 0x0C, 0x02, 0x00, "SSA" },
    { 0x0C, 0x03, 0x00, "USB (with UHC)" },
    { 0x0C, 0x03, 0x10, "USB (with OHC)" },
    { 0x0C, 0x03, 0x80, "USB (other host inf.)" },
    { 0x0C, 0x03, 0xFE, "USB device" },
    { 0x0C, 0x04, 0x00, "Fibre Channel" },
    { 0x0C, 0x05, 0x00, "SMBus" },

    { 0x00, 0x00, 0x00, NULL }
};

#ifdef 0
struct pci_intel_ctrl pci_intel_ctrl[] =
{
    { 0x1022, 0x700C, }, /* AMD-762 */
    { 0x1039, 0x0406, }, /* SiS 85C501/2 */
    { 0x1039, 0x5597, }, /* SiS 5582 */
    { 0x10B9, 0x1541, }, /* ALI M1541 */
    { 0x1106, 0x0305, }, /* VIA VT8363/8365 */
    { 0x1106, 0x3099, }, /* VIA VT8367 [KT266] */
    { 0x1106, 0x3188, }, /* VIA */
    { 0x1106, 0x0282, }, /* VIA */
    { 0x1106, 0x0204, }, /* VIA VT8367 [KT266] */
    { 0x8086, 0x122D, }, /* Intel 82437FX */
    { 0x8086, 0x1237, }, /* Intel 82441FX */
    { 0x8086, 0x1250, }, /* Intel 82439HX */
    { 0x8086, 0x1A30, }, /* Intel 82845 MCH */
    { 0x8086, 0x2560, }, /* Intel 82845G/GL[Brookdale-G]/GE/PE */
    { 0x8086, 0x7030, }, /* Intel 82437VX (asw 2005-03-02) */
    { 0x8086, 0x7100, }, /* Intel 82371AB (asw 2004-07-31) */
    { 0x8086, 0x7124, }, /* Intel 82801AA (asw 2004-11-09) */
    { 0x8086, 0x7190, }, /* Intel 82443BX - AGP enabled */
    { 0x8086, 0x7192, }, /* Intel 82443BX - AGP disabled */
    { 0x0000, 0x0000, },
};
#endif

struct pci_isabridge pci_isabridge[] =
{
    { 0x1022, 0x7410, 1, PCI_IB_AMD, }, /* AMD-766 */
    { 0x1039, 0x0008, 1, PCI_IB_SIS, }, /* SiS 85C503/5513 */
    { 0x10B9, 0x1533, 1, PCI_IB_PIIX, }, /* ALI M1533 */
    { 0x1106, 0x0686, 1, PCI_IB_VIA, }, /* VIA VT82C686 */
    { 0x1106, 0x3074, 1, PCI_IB_VIA, }, /* VIA VT8233 */
    { 0x1106, 0x3227, 1, PCI_IB_VIA, }, /* VIA */
    { 0x8086, 0x122E, 1, PCI_IB_PIIX, }, /* Intel 82371FB */
    { 0x8086, 0x2440, 1, PCI_IB_PIIX, }, /* Intel 82801B */
    { 0x8086, 0x7000, 1, PCI_IB_PIIX, }, /* Intel 82371SB */
    { 0x8086, 0x7030, 1, PCI_IB_PIIX, }, /* Intel 82437VX (asw 2005-03-02) */
    /*
    { 0x8086, 0x7100, 1, PCI_IB_PIIX, }, /* Intel 82371AB (asw 2004-07-31) */
    /*
    { 0x8086, 0x7110, 1, PCI_IB_PIIX, }, /* Intel PIIX4 */
    { 0x8086, 0x7124, 1, PCI_IB_PIIX, }, /* Intel 82801AA (asw 2004-11-09) */
    /*
    { 0x0000, 0x0000, 0, 0, },
    */
};

struct pci_pcibridge pci_pcibridge[] =
{
#ifdef 0
    { 0x8086, 0x1A31, PCI_PCIB_INTEL, }, /* Intel 82845B/A AGP Bridge */
    { 0x8086, 0x2448, PCI_PCIB_INTEL, }, /* Intel 82801 Mobile */
    { 0x8086, 0x244e, PCI_PCIB_INTEL, }, /* Intel 82801 PCI Bridge */
    { 0x8086, 0x2561, PCI_PCIB_INTEL, }, /* Intel 82845 AGP Bridge */
    { 0x8086, 0x7191, PCI_PCIB_INTEL, }, /* Intel 82443BX (AGP bridge) */
    { 0x1022, 0x700D, PCI_PCIB_INTEL, }, /* AMD-762 (AGP 4x) */
    { 0x10B9, 0x5243, PCI_PCIB_INTEL, }, /* ALI M5243 */
    { 0x1106, 0x8305, PCI_AGPB_VIA, }, /* VIA VT8365 [KM133 AGP] */
    { 0x1106, 0xB188, PCI_AGPB_VIA, }, /* VT8237 PCI bridge */
#endif
    { 0x0000, 0x0000, 0, },
};
#endif

```

```
};  
  
/*  
 * $PchId: pci_table.c,v 1.7 2003/09/05 10:53:22 philip Exp $  
 */
```



```
/*
pci_via.h

Created:      Jun 2001 by Philip Homburg <philip@cs.vu.nl>
*/

#define VIA_ISABR_EL      0x54      /* Edge or level triggered */
#define VIA_ISABR_EL_INTA      0x08      /* Edge (1) or level (0) */
#define VIA_ISABR_EL_INTB      0x04
#define VIA_ISABR_EL_INTC      0x02
#define VIA_ISABR_EL_INTD      0x01

#define VIA_ISABR_IRQ_R1 0x55      /* IRQ routing 1 */
#define VIA_ISABR_IRQ_INTD      0xf0      /* routing for INTD */
#define VIA_ISABR_IRQ_INT0      0x0f      /* routing for INT0 */
#define VIA_ISABR_IRQ_R2 0x56      /* IRQ routing 2 */
#define VIA_ISABR_IRQ_INTA      0xf0      /* routing for INTA */
#define VIA_ISABR_IRQ_INTB      0x0f      /* routing for INTB */
#define VIA_ISABR_IRQ_R3 0x57      /* IRQ routing 3 */
#define VIA_ISABR_IRQ_INTC      0xf0      /* routing for INTC */
#define VIA_ISABR_IRQ_INT1      0x0f      /* routing for INT1 */
#define VIA_ISABR_IRQ_R4 0x58      /* IRQ routing 4 */
#define VIA_ISABR_IRQ_INT2      0x0f      /* routing for INT2 */

/*
 * $PchId: pci_via.h,v 1.1 2001/06/20 15:50:25 philip Exp $
 */
```

```
# Makefile for Centronics printer driver (PRINTER)
DRIVER = printer

# directories
u = /usr
i = $u/include
s = $i/sys
m = $i/minix
b = $i/ibm
d = ..

# programs, flags, etc.
CC =      exec cc
CFLAGS = -I$i
LDFLAGS = -i
LIBS = -lsys -lsysutil

OBJ = printer.o

# build local binary
all build:      $(DRIVER)
$(DRIVER):      $(OBJ)
                $(CC) -o $@ $(LDFLAGS) $(OBJ) $(LIBS)
#              install -S 64w $(DRIVER)

# install with other drivers
install:      /usr/sbin/$(DRIVER)
/usr/sbin/$(DRIVER):  $(DRIVER)
                install -o root -c $? $@
#              install -o root -cs $? $@

# clean up local files
clean:
                rm -f *.o *.bak $(DRIVER)

depend:
                /usr/bin/mkdep "$(CC) -E $(CPPFLAGS)" *.c > .depend

# Include generated dependencies.
include .depend
```

```

/* This file contains the printer driver. It is a fairly simple driver,
 * supporting only one printer. Characters that are written to the driver
 * are written to the printer without any changes at all.
 *
 * Changes:
 *     May 07, 2004    fix: wait until printer is ready (Jorrit N. Herder)
 *     May 06, 2004    printer driver moved to user-space (Jorrit N. Herder)
 *
 * The valid messages and their parameters are:
 *
 *     DEV_OPEN:  initializes the printer
 *     DEV_CLOSE: does nothing
 *     HARD_INT:  interrupt handler has finished current chunk of output
 *     DEV_WRITE: a process wants to write on a terminal
 *     CANCEL:    terminate a previous incomplete system call immediately
 *
 *      m_type      TTY_LINE   IO_ENDPT   COUNT   ADDRESS
 *  +-----+-----+-----+-----+-----+
 *  | DEV_OPEN      |           |           |         |
 *  +-----+-----+-----+-----+-----+
 *  | DEV_CLOSE      |           | proc nr  |         |
 *  +-----+-----+-----+-----+-----+
 *  | HARD_INT       |           |           |         |
 *  +-----+-----+-----+-----+-----+
 *  | SYS_EVENT      |           |           |         |
 *  +-----+-----+-----+-----+-----+
 *  | DEV_WRITE      | minor dev | proc nr  | count   | buf ptr |
 *  +-----+-----+-----+-----+-----+
 *  | CANCEL         | minor dev | proc nr  |         |         |
 *  +-----+-----+-----+-----+-----+
 *
 * Note: since only 1 printer is supported, minor dev is not used at present.
 */

```

```
#include "../drivers.h"
```

```

/* Control bits (in port_base + 2). "+" means positive logic and "-" means
 * negative logic. Most of the signals are negative logic on the pins but
 * many are converted to positive logic in the ports. Some manuals are
 * misleading because they only document the pin logic.
 *
 *     +0x01    Pin 1   -Strobe
 *     +0x02    Pin 14  -Auto Feed
 *     -0x04    Pin 16  -Initialize Printer
 *     +0x08    Pin 17  -Select Printer
 *     +0x10    IRQ7 Enable
 *
 * Auto Feed and Select Printer are always enabled. Strobe is enabled briefly
 * when characters are output. Initialize Printer is enabled briefly when
 * the task is started. IRQ7 is enabled when the first character is output
 * and left enabled until output is completed (or later after certain
 * abnormal completions).
 */
#define ASSERT_STROBE    0x1D    /* strobe a character to the interface */
#define NEGATE_STROBE    0x1C    /* enable interrupt on interface */
#define PR_SELECT        0x0C    /* select printer bit */
#define INIT_PRINTER     0x08    /* init printer bits */

/* Status bits (in port_base + 2).
 *
 *     -0x08    Pin 15  -Error
 *     +0x10    Pin 13  +Select Status
 *     +0x20    Pin 12  +Out of Paper
 *     -0x40    Pin 10  -Acknowledge
 *     -0x80    Pin 11  +Busy
 */
#define BUSY_STATUS      0x10    /* printer gives this status when busy */
#define NO_PAPER         0x20    /* status bit saying that paper is out */
#define NORMAL_STATUS    0x90    /* printer gives this status when idle */
#define ON_LINE          0x10    /* status bit saying that printer is online */
#define STATUS_MASK      0xB0    /* mask to filter out status bits */

#define MAX_ONLINE_RETRIES 120    /* about 60s: waits 0.5s after each retry */

```

```

/* Centronics interface timing that must be met by software (in microsec).
 *
 * Strobe length:      0.5u to 100u (not sure about the upper limit).
 * Data set up:       0.5u before strobe.
 * Data hold:         0.5u after strobe.
 * Init pulse length: over 200u (not sure).
 *
 * The strobe length is about 50u with the code here and function calls for
 * sys_outb() - not much to spare. The 0.5u minimums will not be violated
 * with the sys_outb() messages exchanged.
 */

PRIVATE int caller;          /* process to tell when printing done (FS) */
PRIVATE int revive_pending;  /* set to true if revive is pending */
PRIVATE int revive_status;   /* revive status */
PRIVATE int done_status;     /* status of last output completion */
PRIVATE int oleft;           /* bytes of output left in obuf */
PRIVATE char obuf[128];      /* output buffer */
PRIVATE char *optr;          /* ptr to next char in obuf to print */
PRIVATE int orig_count;      /* original byte count */
PRIVATE int port_base;       /* I/O port for printer */
PRIVATE int proc_nr;         /* user requesting the printing */
PRIVATE int user_left;       /* bytes of output left in user buf */
PRIVATE vir_bytes user_vir;  /* address of remainder of user buf */
PRIVATE int writing;          /* nonzero while write is in progress */
PRIVATE int irq_hook_id;     /* id of irq hook at kernel */

extern int errno;            /* error number */

FORWARD _PROTOTYPE( void do_cancel, (message *m_ptr) );
FORWARD _PROTOTYPE( void output_done, (void) );
FORWARD _PROTOTYPE( void do_write, (message *m_ptr) );
FORWARD _PROTOTYPE( void do_status, (message *m_ptr) );
FORWARD _PROTOTYPE( void prepare_output, (void) );
FORWARD _PROTOTYPE( void do_initialize, (void) );
FORWARD _PROTOTYPE( void reply, (int code,int replyee,int proc,int status));
FORWARD _PROTOTYPE( void do_printer_output, (void) );
FORWARD _PROTOTYPE( void do_signal, (message *m_ptr) );

/*=====
 *                               printer_task                               *
 *=====*/
PUBLIC void main(void)
{
    /* Main routine of the printer task. */
    message pr_mess;          /* buffer for all incoming messages */
    struct sigaction sa;
    int s;

    /* Install signal handlers. Ask PM to transform signal into message. */
    sa.sa_handler = SIG_MESS;
    sigemptyset(&sa.sa_mask);
    sa.sa_flags = 0;
    if (sigaction(SIGTERM,&sa,NULL)<0) panic("PRN","sigaction failed", errno);

    while (TRUE) {
        receive(ANY, &pr_mess);
        switch(pr_mess.m_type) {
            case DEV_OPEN:
                do_initialize();          /* initialize */
                /* fall through */
            case DEV_CLOSE:
                reply(TASK_REPLY, pr_mess.m_source, pr_mess.IO_ENDPT, OK);
                break;
            case DEV_WRITE:
                do_write(&pr_mess);      break;
            case DEV_STATUS:
                do_status(&pr_mess);     break;
            case CANCEL:
                do_cancel(&pr_mess);     break;
            case HARD_INT:
                do_printer_output();     break;
            case SYS_SIG:
                do_signal(&pr_mess);     break;
            case DEV_PING:
                notify(pr_mess.m_source); break;
            case PROC_EVENT:
                break;
            default:
                reply(TASK_REPLY, pr_mess.m_source, pr_mess.IO_ENDPT, EINVAL);
        }
    }
}

```

```

    }
}

/*=====
 *                               do_signal                               *
 *=====*/
PRIVATE void do_signal(m_ptr)
message *m_ptr;                /* signal message */
{
    int sig;
    sigset_t sigset = m_ptr->NOTIFY_ARG;

    /* Expect a SIGTERM signal when this server must shutdown. */
    if (sigismember(&sigset, SIGTERM)) {
        exit(0);
    }
    /* Ignore all other signals. */
}

/*=====
 *                               do_write                               *
 *=====*/
PRIVATE void do_write(m_ptr)
register message *m_ptr;        /* pointer to the newly arrived message */
{
    /* The printer is used by sending DEV_WRITE messages to it. Process one. */

    register int r = SUSPEND;
    int retries;
    unsigned long status;

    /* Reject command if last write is not yet finished, the count is not
     * positive, or the user address is bad.
     */
    if (writing)                r = EIO;
    else if (m_ptr->COUNT <= 0) r = EINVAL;

    /* Reply to FS, no matter what happened, possible SUSPEND caller. */
    reply(TASK_REPLY, m_ptr->m_source, m_ptr->IO_ENDPT, r);

    /* If no errors occurred, continue printing with SUSPENDED caller.
     * First wait until the printer is online to prevent stupid errors.
     */
    if (SUSPEND == r) {
        caller = m_ptr->m_source;
        proc_nr = m_ptr->IO_ENDPT;
        user_left = m_ptr->COUNT;
        orig_count = m_ptr->COUNT;
        user_vir = (vir_bytes) m_ptr->ADDRESS;
        writing = TRUE;

        retries = MAX_ONLINE_RETRIES + 1;
        while (--retries > 0) {
            sys_inb(port_base + 1, &status);
            if ((status & ON_LINE)) { /* printer online! */
                prepare_output();
                do_printer_output();
                return;
            }
            tickdelay(30); /* wait before retry */
        }
        /* If we reach this point, the printer was not online in time. */
        done_status = status;
        output_done();
    }
}

/*=====
 *                               output_done                               *
 *=====*/
PRIVATE void output_done()
{

```

```

/* Previous chunk of printing is finished. Continue if OK and more.
 * Otherwise, reply to caller (FS).
 */
    register int status;

    if (!writing) return; /* probably leftover interrupt */
    if (done_status != OK) { /* printer error occurred */
        status = EIO;
        if ((done_status & ON_LINE) == 0) {
            printf("Printer is not on line\n");
        } else if ((done_status & NO_PAPER)) {
            printf("Printer is out of paper\n");
            status = EAGAIN;
        } else {
            printf("Printer error, status is 0x%02X\n", done_status);
        }
        /* Some characters have been printed, tell how many. */
        if (status == EAGAIN && user_left < orig_count) {
            status = orig_count - user_left;
        }
        oleft = 0; /* cancel further output */
    }
    else if (user_left != 0) { /* not yet done, continue! */
        prepare_output();
        return;
    }
    else { /* done! report back to FS */
        status = orig_count;
    }
    revive_pending = TRUE;
    revive_status = status;
    notify(caller);
}

/*=====
 * do_status
 *=====*/
PRIVATE void do_status(m_ptr)
register message *m_ptr; /* pointer to the newly arrived message */
{
    if (revive_pending) {
        m_ptr->m_type = DEV_REVIVE; /* build message */
        m_ptr->REP_ENDPT = proc_nr;
        m_ptr->REP_STATUS = revive_status;

        writing = FALSE; /* unmark event */
        revive_pending = FALSE; /* unmark event */
    } else {
        m_ptr->m_type = DEV_NO_STATUS;
    }
    send(m_ptr->m_source, m_ptr); /* send the message */
}

/*=====
 * do_cancel
 *=====*/
PRIVATE void do_cancel(m_ptr)
register message *m_ptr; /* pointer to the newly arrived message */
{
    /* Cancel a print request that has already started. Usually this means that
     * the process doing the printing has been killed by a signal. It is not
     * clear if there are race conditions. Try not to cancel the wrong process,
     * but rely on FS to handle the EINTR reply and de-suspension properly.
     */

    if (writing && m_ptr->IO_ENDPT == proc_nr) {
        oleft = 0; /* cancel output by interrupt handler */
        writing = FALSE;
        revive_pending = FALSE;
    }
    reply(TASK_REPLY, m_ptr->m_source, m_ptr->IO_ENDPT, EINTR);
}

/*=====

```

```

*                                     reply                                     *
*=====*/
PRIVATE void reply(code, replyee, process, status)
int code;                               /* TASK_REPLY or REVIVE */
int replyee;                           /* destination for message (normally FS) */
int process;                           /* which user requested the printing */
int status;                            /* number of chars printed or error code */
{
/* Send a reply telling FS that printing has started or stopped. */

    message pr_mess;

    pr_mess.m_type = code;               /* TASK_REPLY or REVIVE */
    pr_mess.REP_STATUS = status;         /* count or EIO */
    pr_mess.REP_ENDPT = process;         /* which user does this pertain to */
    send(replyee, &pr_mess);            /* send the message */
}

/*=====*
*                                     do_initialize                             *
*=====*/
PRIVATE void do_initialize()
{
/* Set global variables and initialize the printer. */
    static int initialized = FALSE;
    if (initialized) return;
    initialized = TRUE;

    /* Get the base port for first printer. */
    sys_vircopy(SELF, BIOS_SEG, LPT1_IO_PORT_ADDR,
                SELF, D, (vir_bytes) &port_base, LPT1_IO_PORT_SIZE);
    sys_outb(port_base + 2, INIT_PRINTER);
    tickdelay(1);                       /* easily satisfies Centronics minimum */
                                        /* was 2 millisecs; now is ~17 millisecs */

    sys_outb(port_base + 2, PR_SELECT);
    irq_hook_id = 0;
    sys_irqsetpolicy(PRINTER_IRQ, 0, &irq_hook_id);
    sys_irqenable(&irq_hook_id);
}

/*=====*
*                                     prepare_output                           *
*=====*/
PRIVATE void prepare_output()
{
/* Start next chunk of printer output. Fetch the data from user space. */

    register int chunk;

    if ( (chunk = user_left) > sizeof obuf) chunk = sizeof obuf;
    if (OK!=sys_datacopy(proc_nr, user_vir, SELF, (vir_bytes) obuf, chunk)) {
        done_status = EFAULT;
        output_done();
        return;
    }
    optr = obuf;
    oleft = chunk;
}

/*=====*
*                                     do_printer_output                         *
*=====*/
PRIVATE void do_printer_output()
{
/* This function does the actual output to the printer. This is called on
* a HARD_INT message sent from the generic interrupt handler that 'forwards'
* interrupts to this driver. The generic handler did not reenale the
* printer IRQ yet!
*/

    unsigned long status;
    pvb_pair_t char_out[3];

```

```
if (oleft == 0) {
    /* Nothing more to print. Turn off printer interrupts in case they
     * are level-sensitive as on the PS/2. This should be safe even
     * when the printer is busy with a previous character, because the
     * interrupt status does not affect the printer.
     */
    sys_outb(port_base + 2, PR_SELECT);
    sys_irgenable(&irq_hook_id);
    return;
}

do {
    /* Loop to handle fast (buffered) printers. It is important that
     * processor interrupts are not disabled here, just printer interrupts.
     */
    (void) sys_inb(port_base + 1, &status);
    if ((status & STATUS_MASK) == BUSY_STATUS) {
        /* Still busy with last output. This normally happens
         * immediately after doing output to an unbuffered or slow
         * printer. It may happen after a call from prepare_output or
         * pr_restart, since they are not synchronized with printer
         * interrupts. It may happen after a spurious interrupt.
         */
        sys_irgenable(&irq_hook_id);
        return;
    }
    if ((status & STATUS_MASK) == NORMAL_STATUS) {
        /* Everything is all right. Output another character. */
        pv_set(char_out[0], port_base, *optr++);
        pv_set(char_out[1], port_base+2, ASSERT_STROBE);
        pv_set(char_out[2], port_base+2, NEGATE_STROBE);
        sys_voutb(char_out, 3); /* request series of port outb */

        user_vir++;
        user_left--;
    } else {
        /* Error. This would be better ignored (treat as busy). */
        done_status = status;
        output_done();
        sys_irgenable(&irq_hook_id);
        return;
    }
}
while (--oleft != 0);

/* Finished printing chunk OK. */
done_status = OK;
output_done();
sys_irgenable(&irq_hook_id);
}
```



```
# Makefile for random driver (RANDOM)
DRIVER = random

# directories
u = /usr
i = $u/include
s = $i/sys
m = $i/minix
b = $i/ibm
d = ..

# programs, flags, etc.
MAKE = exec make
CC = exec cc
CFLAGS = -I$i
LDFLAGS = -i
LIBS = -lsys -lsysutil

OBJ = main.o random.o sha2.o aes/rijndael_api.o aes/rijndael_alg.o
LIBDRIVER = $d/libdriver/driver.o

# build local binary
all build: $(DRIVER)
$(DRIVER): $(OBJ) $(LIBDRIVER)
    $(CC) -o $@ $(LDFLAGS) $(OBJ) $(LIBDRIVER) $(LIBS)
    install -S 1024w $(DRIVER)

$(LIBDRIVER):
    cd $d/libdriver && $(MAKE)

aes/rijndael_api.o:
    $(CC) -c -o $@ aes/rijndael_api.c

aes/rijndael_alg.o:
    $(CC) -c -o $@ aes/rijndael_alg.c

# install with other drivers
install: /usr/sbin/$(DRIVER)
/usr/sbin/$(DRIVER): $(DRIVER)
    install -o root -cs $? $@

# clean up local files
clean:
    rm -f $(DRIVER) *.o *.bak

depend:
    /usr/bin/mkdep "$(CC) -E $(CPPFLAGS)" *.c ../libdriver/*.c aes/*.c > .depend

# Include generated dependencies.
include .depend
```

```

/* This file contains the device dependent part of the drivers for the
 * following special files:
 *    /dev/random      - random number generator
 */

#include "../drivers.h"
#include "../libdriver/driver.h"
#include <sys/ioc_memory.h>
#include "../kernel/const.h"
#include "../kernel/config.h"
#include "../kernel/type.h"

#include "assert.h"
#include "random.h"

#define NR_DEVS          1          /* number of minor devices */
#define RANDOM_DEV      0          /* minor device for /dev/random */

#define KRANDOM_PERIOD   1          /* ticks between krandom calls */

PRIVATE struct device m_geom[NR_DEVS]; /* base and size of each device */
PRIVATE int m_device;                /* current device */

extern int errno;                    /* error number for PM calls */

FORWARD _PROTOTYPE( char *r_name, (void) );
FORWARD _PROTOTYPE( struct device *r_prepare, (int device) );
FORWARD _PROTOTYPE( int r_transfer, (int proc_nr, int opcode, off_t position,
                                     iovec_t *iov, unsigned nr_req) );
FORWARD _PROTOTYPE( int r_do_open, (struct driver *dp, message *m_ptr) );
FORWARD _PROTOTYPE( void r_init, (void) );
FORWARD _PROTOTYPE( int r_ioctl, (struct driver *dp, message *m_ptr) );
FORWARD _PROTOTYPE( void r_geometry, (struct partition *entry) );
FORWARD _PROTOTYPE( void r_random, (struct driver *dp, message *m_ptr) );

/* Entry points to this driver. */
PRIVATE struct driver r_dtab = {
    r_name,          /* current device's name */
    r_do_open,       /* open or mount */
    do_nop,          /* nothing on a close */
    r_ioctl,         /* specify ram disk geometry */
    r_prepare,       /* prepare for I/O on a given minor device */
    r_transfer,      /* do the I/O */
    nop_cleanup,     /* no need to clean up */
    r_geometry,      /* device "geometry" */
    nop_signal,      /* system signals */
    r_random,        /* get randomness from kernel (alarm) */
    nop_cancel,
    nop_select,
    NULL,
    NULL
};

/* Buffer for the /dev/random number generator. */
#define RANDOM_BUF_SIZE 1024
PRIVATE char random_buf[RANDOM_BUF_SIZE];

/*=====
 *                               main                               *
 *=====*/
PUBLIC int main(void)
{
    r_init();          /* initialize the memory driver */
    driver_task(&r_dtab); /* start driver's main loop */
    return(OK);
}

/*=====
 *                               r_name                               *
 *=====*/
PRIVATE char *r_name()
{
    /* Return a name for the current device. */
    static char name[] = "random";

```

```

    return name;
}

/*=====
 *                               r_prepare                               *
 *=====*/
PRIVATE struct device *r_prepare(device)
int device;
{
    /* Prepare for I/O on a device: check if the minor device number is ok. */

    if (device < 0 || device >= NR_DEVS) return(NIL_DEV);
    m_device = device;

    return(&m_geom[device]);
}

/*=====
 *                               r_transfer                               *
 *=====*/
PRIVATE int r_transfer(proc_nr, opcode, position, iov, nr_req)
int proc_nr;          /* process doing the request */
int opcode;           /* DEV_GATHER or DEV_SCATTER */
off_t position;       /* offset on device to read or write */
iovec_t *iov;         /* pointer to read or write request vector */
unsigned nr_req;      /* length of request vector */
{
    /* Read or write one the driver's minor devices. */
    unsigned count, left, chunk;
    vir_bytes user_vir;
    struct device *dv;
    unsigned long dv_size;

    /* Get minor device number and check for /dev/null. */
    dv = &m_geom[m_device];
    dv_size = cv64ul(dv->dv_size);

    while (nr_req > 0) {

        /* How much to transfer and where to / from. */
        count = iov->iov_size;
        user_vir = iov->iov_addr;

        switch (m_device) {

            /* Random number generator. Character instead of block device. */
            case RANDOM_DEV:
                if (opcode == DEV_GATHER && !random_isseeded())
                    return(EAGAIN);
                left = count;
                while (left > 0) {
                    chunk = (left > RANDOM_BUF_SIZE) ? RANDOM_BUF_SIZE : left;
                    if (opcode == DEV_GATHER) {
                        random_getbytes(random_buf, chunk);
                        sys_vircopy(SEL, D, (vir_bytes) random_buf,
                                proc_nr, D, user_vir, chunk);
                    } else if (opcode == DEV_SCATTER) {
                        sys_vircopy(proc_nr, D, user_vir,
                                SEL, D, (vir_bytes) random_buf, chunk);
                        random_putbytes(random_buf, chunk);
                    }
                    user_vir += chunk;
                    left -= chunk;
                }
                break;

            /* Unknown (illegal) minor device. */
            default:
                return(EINVAL);
        }

        /* Book the number of bytes transferred. */
        position += count;
        iov->iov_addr += count;
    }
}

```

```

    if ((iov->iov_size -= count) == 0) { iov++; nr_req--; }
}
return(OK);
}

/*=====
 *                               r_do_open                               *
 *=====*/
PRIVATE int r_do_open(dp, m_ptr)
struct driver *dp;
message *m_ptr;
{
    /* Check device number on open.
    */
    if (r_prepare(m_ptr->DEVICE) == NIL_DEV) return(ENXIO);

    return(OK);
}

/*=====
 *                               r_init                               *
 *=====*/
PRIVATE void r_init()
{
    /* Initialize this task. All minor devices are initialized one by one. */
    random_init();
    r_random(NULL, NULL); /* also set periodic timer */
}

/*=====
 *                               r_ioctl                               *
 *=====*/
PRIVATE int r_ioctl(dp, m_ptr)
struct driver *dp; /* pointer to driver structure */
message *m_ptr; /* pointer to control message */
{
    struct device *dv;
    if ((dv = r_prepare(m_ptr->DEVICE)) == NIL_DEV) return(ENXIO);

    switch (m_ptr->REQUEST) {

        default:
            return(do_diocntl(&r_dtab, m_ptr));
    }
    return(OK);
}

/*=====
 *                               r_random                               *
 *=====*/
PRIVATE void r_random(dp, m_ptr)
struct driver *dp; /* pointer to driver structure */
message *m_ptr; /* pointer to alarm message */
{
    /* Fetch random information from the kernel to update /dev/random. */
    int i, s, r_next, r_size, r_high;
    struct randomness krandom;

    if (OK != (s=sys_getrandomness(&krandom)))
        report("RANDOM", "sys_getrandomness failed", s);

    for (i= 0; i<RANDOM_SOURCES; i++)
    {
        r_next= krandom.bin[i].r_next;
        r_size= krandom.bin[i].r_size;
        r_high= r_next+r_size;
        if (r_high <= RANDOM_ELEMENTS)
        {
            random_update(i, &krandom.bin[i].r_buf[r_next], r_size);
        }
        else
        {
            assert(r_next < RANDOM_ELEMENTS);

```

```
        random_update(i, &krandom.bin[i].r_buf[r_next],
                      RANDOM_ELEMENTS-r_next);
        random_update(i, &krandom.bin[i].r_buf[0],
                      r_high-RANDOM_ELEMENTS);
    }
}

/* Schedule new alarm for next m_random call. */
if (OK != (s=sys_setalarm(KRANDOM_PERIOD, 0)))
    report("RANDOM", "sys_setalarm failed", s);
}

/*=====
 *                               r_geometry                               *
 *=====*/
PRIVATE void r_geometry(entry)
struct partition *entry;
{
    /* Memory devices don't have a geometry, but the outside world insists. */
    entry->cylinders = div64u(m_geom[m_device].dv_size, SECTOR_SIZE) / (64 * 32);
    entry->heads = 64;
    entry->sectors = 32;
}
```

```
/*
random.c

Random number generator.

The random number generator collects data from the kernel and compressed
that data into a seed for a psuedo random number generator.
*/

#include "../drivers.h"
#include "../kernel/const.h"
#include "assert.h"

#include "random.h"
#include "sha2.h"
#include "aes/rijndael.h"

#define N_DERIV 16
#define NR_POOLS 32
#define MIN_SAMPLES 256 /* Number of samples needed in pool 0 for a
                          * re-seed.
                          */

PRIVATE unsigned long deriv[RANDOM_SOURCES][N_DERIV];
PRIVATE int pool_ind[RANDOM_SOURCES];
PRIVATE SHA256_CTX pool_ctx[NR_POOLS];
PRIVATE unsigned samples= 0;
PRIVATE int got_seeded= 0;
PRIVATE u8_t random_key[2*AES_BLOCKSIZE];
PRIVATE u32_t count_lo, count_hi;
PRIVATE u32_t reseed_count;

FORWARD _PROTOTYPE( void add_sample, (int source, unsigned long sample) );
FORWARD _PROTOTYPE( void data_block, (rd_keyinstance *keyp,
                                      void *data) );
FORWARD _PROTOTYPE( void reseed, (void) );

PUBLIC void random_init()
{
    int i, j;

    assert(&deriv[RANDOM_SOURCES-1][N_DERIV-1] ==
           &deriv[0][0] + RANDOM_SOURCES*N_DERIV -1);

    for (i= 0; i<RANDOM_SOURCES; i++)
    {
        for (j= 0; j<N_DERIV; j++)
            deriv[i][j]= 0;
        pool_ind[i]= 0;
    }
    for (i= 0; i<NR_POOLS; i++)
        SHA256_Init(&pool_ctx[i]);
    count_lo= 0;
    count_hi= 0;
    reseed_count= 0;
}

PUBLIC int random_isseeded()
{
    if (got_seeded)
        return 1;
    return 0;
}

PUBLIC void random_update(source, buf, count)
int source;
unsigned short *buf;
int count;
{
    int i;

#if 0
    printf("random_update: got %d samples for source %d\n", count, source);
#endif
}
```

```
    if (source < 0 || source >= RANDOM_SOURCES)
        panic("memory", "random_update: bad source", source);
    for (i= 0; i<count; i++)
        add_sample(source, buf[i]);
    reseed();
}

PUBLIC void random_getbytes(buf, size)
void *buf;
size_t size;
{
    int n, r;
    u8_t *cp;
    rd_keyinstance key;
    u8_t output[AES_BLOCKSIZE];

    r= rijndael_makekey(&key, sizeof(random_key), random_key);
    assert(r == 0);

    cp= buf;
    while (size > 0)
    {
        n= AES_BLOCKSIZE;
        if (n > size)
        {
            n= size;
            data_block(&key, output);
            memcpy(cp, output, n);
        }
        else
            data_block(&key, cp);
        cp += n;
        size -= n;
    }

    /* Generate new key */
    assert(sizeof(random_key) == 2*AES_BLOCKSIZE);
    data_block(&key, random_key);
    data_block(&key, random_key+AES_BLOCKSIZE);
}

PUBLIC void random_putbytes(buf, size)
void *buf;
size_t size;
{
    /* Add bits to pool zero */
    SHA256_Update(&pool_ctx[0], buf, size);

    /* Assume that these bits are truly random. Increment samples
     * with the number of bits.
     */
    samples += size*8;

    reseed();
}

PRIVATE void add_sample(source, sample)
int source;
unsigned long sample;
{
    int i, pool_nr;
    unsigned long d, v, di, min;

    /* Delete bad sample. Compute the Nth derivative. Delete the sample
     * if any derivative is too small.
     */
    min= (unsigned long)-1;
    v= sample;
    for (i= 0; i<N_DERIV; i++)
    {
        di= deriv[source][i];

        /* Compute the difference */
        if (v >= di)
```

```

                d= v-di;
            else
                d= di-v;
            deriv[source][i]= v;
            v= d;
            if (v <min)
                min= v;
        }
        if (min < 2)
        {
#ifdef 0
            printf("ignoring sample '%u' from source %d\n",
                sample, source);
#endif
            return;
        }
#ifdef 0
        printf("accepting sample '%u' from source %d\n", sample, source);
#endif

        pool_nr= pool_ind[source];
        assert(pool_nr >= 0 && pool_nr < NR_POOLS);

        SHA256_Update(&pool_ctx[pool_nr], (unsigned char *)&sample,
            sizeof(sample));
        if (pool_nr == 0)
            samples++;
        pool_nr++;
        if (pool_nr >= NR_POOLS)
            pool_nr= 0;
        pool_ind[source]= pool_nr;
    }

PRIVATE void data_block(keyp, data)
rd_keyinstance *keyp;
void *data;
{
    int r;
    u8_t input[AES_BLOCKSIZE];

    memset(input, '\0', sizeof(input));

    /* Do we want the output of the random numbers to be portable
     * across platforms (for example for RSA signatures)? At the moment
     * we don't do anything special. Encrypt the counter with the AES
     * key.
     */
    assert(sizeof(count_lo)+sizeof(count_hi) <= AES_BLOCKSIZE);
    memcpy(input, &count_lo, sizeof(count_lo));
    memcpy(input+sizeof(count_lo), &count_hi, sizeof(count_hi));
    r= rijndael_ecb_encrypt(keyp, input, data, AES_BLOCKSIZE, NULL);
    assert(r == AES_BLOCKSIZE);

    count_lo++;
    if (count_lo == 0)
        count_hi++;
}

PRIVATE void reseed()
{
    int i;
    SHA256_CTX ctx;
    u8_t digest[SHA256_DIGEST_LENGTH];

    if (samples < MIN_SAMPLES)
        return;

    reseed_count++;
    SHA256_Init(&ctx);
    if (got_seeded)
        SHA256_Update(&ctx, random_key, sizeof(random_key));
    SHA256_Final(digest, &pool_ctx[0]);
    SHA256_Update(&ctx, digest, sizeof(digest));
    SHA256_Init(&pool_ctx[0]);

```



```
for (i= 1; i<NR_POOLS; i++)
{
    if ((reseed_count & (1UL << (i-1))) != 0)
        break;
    SHA256_Final(digest, &pool_ctx[i]);
    SHA256_Update(&ctx, digest, sizeof(digest));
    SHA256_Init(&pool_ctx[i]);
}
SHA256_Final(digest, &ctx);
assert(sizeof(random_key) == sizeof(digest));
memcpy(random_key, &digest, sizeof(random_key));
samples= 0;

got_seeded= 1;
}
```

```
/*
random.h

Public interface to the random number generator
*/

_PROTOTYPE( void random_init, (void) ) ;
_PROTOTYPE( int random_isseeded, (void) ) ;
_PROTOTYPE( void random_update, (int source, unsigned short *buf,
                                int count) ) ;
_PROTOTYPE( void random_getbytes, (void *buf, size_t size) ) ;
_PROTOTYPE( void random_putbytes, (void *buf, size_t size) ) ;
```

```
/*      $FreeBSD: src/sys/crypto/sha2/sha2.c,v 1.2.2.2 2002/03/05 08:36:47 ume Exp $      */
/*
/*      $KAME: sha2.c,v 1.8 2001/11/08 01:07:52 itojun Exp $      */

/*
 * sha2.c
 *
 * Version 1.0.0beta1
 *
 * Written by Aaron D. Gifford <me@aarongifford.com>
 *
 * Copyright 2000 Aaron D. Gifford. All rights reserved.
 *
 * Redistribution and use in source and binary forms, with or without
 * modification, are permitted provided that the following conditions
 * are met:
 * 1. Redistributions of source code must retain the above copyright
 * notice, this list of conditions and the following disclaimer.
 * 2. Redistributions in binary form must reproduce the above copyright
 * notice, this list of conditions and the following disclaimer in the
 * documentation and/or other materials provided with the distribution.
 * 3. Neither the name of the copyright holder nor the names of contributors
 * may be used to endorse or promote products derived from this software
 * without specific prior written permission.
 *
 * THIS SOFTWARE IS PROVIDED BY THE AUTHOR(S) AND CONTRIBUTOR(S) ``AS IS'' AND
 * ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE
 * IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE
 * ARE DISCLAIMED. IN NO EVENT SHALL THE AUTHOR(S) OR CONTRIBUTOR(S) BE LIABLE
 * FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL
 * DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS
 * OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION)
 * HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT
 * LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY
 * OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF
 * SUCH DAMAGE.
 */

#include <sys/types.h>
/* #include <sys/time.h> */
/* #include <sys/sysm.h> */
/* #include <machine/endian.h> */
#include "sha2.h"

/*
 * ASSERT NOTE:
 * Some sanity checking code is included using assert(). On my FreeBSD
 * system, this additional code can be removed by compiling with NDEBUG
 * defined. Check your own systems manpage on assert() to see how to
 * compile WITHOUT the sanity checking code on your system.
 *
 * UNROLLED TRANSFORM LOOP NOTE:
 * You can define SHA2_UNROLL_TRANSFORM to use the unrolled transform
 * loop version for the hash transform rounds (defined using macros
 * later in this file). Either define on the command line, for example:
 *
 * cc -DSHA2_UNROLL_TRANSFORM -o sha2 sha2.c sha2prog.c
 *
 * or define below:
 *
 * #define SHA2_UNROLL_TRANSFORM
 */

#if defined(__bsdi__) || defined(__FreeBSD__)
#define assert(x)
#endif

/** SHA-256/384/512 Machine Architecture Definitions *****/
/*
 * SHA2_BYTE_ORDER NOTE:
 *
 * Please make sure that your system defines SHA2_BYTE_ORDER. If your

```

```

* architecture is little-endian, make sure it also defines
* SHA2_LITTLE_ENDIAN and that the two (SHA2_BYTE_ORDER and SHA2_LITTLE_ENDIAN) are
* equivalent.
*
* If your system does not define the above, then you can do so by
* hand like this:
*
* #define SHA2_LITTLE_ENDIAN 1234
* #define SHA2_BIG_ENDIAN 4321
*
* And for little-endian machines, add:
*
* #define SHA2_BYTE_ORDER SHA2_LITTLE_ENDIAN
*
* Or for big-endian machines:
*
* #define SHA2_BYTE_ORDER SHA2_BIG_ENDIAN
*
* The FreeBSD machine this was written on defines BYTE_ORDER
* appropriately by including <sys/types.h> (which in turn includes
* <machine/endian.h> where the appropriate definitions are actually
* made).
*/
#if !defined(SHA2_BYTE_ORDER) || (SHA2_BYTE_ORDER != SHA2_LITTLE_ENDIAN && SHA2_BYTE_ORDE
R != SHA2_BIG_ENDIAN)
#error Define SHA2_BYTE_ORDER to be equal to either SHA2_LITTLE_ENDIAN or SHA2_BIG_ENDIAN
#endif

/*
* Define the followingsha2_* types to types of the correct length on
* the native architecture. Most BSD systems and Linux define u_intXX_t
* types. Machines with very recent ANSI C headers, can use the
* uintXX_t definintions from inttypes.h by defining SHA2_USE_INTTYPES_H
* during compile or in the sha.h header file.
*
* Machines that support neither u_intXX_t nor inttypes.h's uintXX_t
* will need to define these three typedefs below (and the appropriate
* ones in sha.h too) by hand according to their system architecture.
*
* Thank you, Jun-ichiro itojun Hagino, for suggesting using u_intXX_t
* types and pointing out recent ANSI C support for uintXX_t in inttypes.h.
*/
#if 0 /*def SHA2_USE_INTTYPES_H*/

typedef uint8_t sha2_byte; /* Exactly 1 byte */
typedef uint32_t sha2_word32; /* Exactly 4 bytes */
typedef uint64_t sha2_word64; /* Exactly 8 bytes */

#else /* SHA2_USE_INTTYPES_H */

typedef u_int8_t sha2_byte; /* Exactly 1 byte */
typedef u_int32_t sha2_word32; /* Exactly 4 bytes */
typedef u_int64_t sha2_word64; /* Exactly 8 bytes */

#endif /* SHA2_USE_INTTYPES_H */

/** SHA-256/384/512 Various Length Definitions *****/
/* NOTE: Most of these are in sha2.h */
#define SHA256_SHORT_BLOCK_LENGTH (SHA256_BLOCK_LENGTH - 8)
#define SHA384_SHORT_BLOCK_LENGTH (SHA384_BLOCK_LENGTH - 16)
#define SHA512_SHORT_BLOCK_LENGTH (SHA512_BLOCK_LENGTH - 16)

/** ENDIAN REVERSAL MACROS *****/
#if SHA2_BYTE_ORDER == SHA2_LITTLE_ENDIAN
#define REVERSE32(w,x) { \
    sha2_word32 tmp = (w); \
    tmp = (tmp >> 16) | (tmp << 16); \
    (x) = ((tmp & 0xff00ff00UL) >> 8) | ((tmp & 0x00ff00ffUL) << 8); \
}
#define REVERSE64(w,x) { \
    sha2_word64 tmp = (w); \
    tmp = (tmp >> 32) | (tmp << 32); \
    tmp = ((tmp & 0xff00ff00ff00ff00ULL) >> 8) | \
    ((tmp & 0x00ff00ff00ff00ffULL) << 8); \
}

```

```

        (x) = ((tmp & 0xffff0000ffff0000ULL) >> 16) | \
        ((tmp & 0x0000ffff0000ffffULL) << 16); \
    }
}
#if MINIX_64BIT
#undef REVERSE64
#define REVERSE64(w,x) { \
    u32_t hi, lo; \
    REVERSE32(ex64hi((w)), lo); \
    REVERSE32(ex64lo((w)), hi); \
    (x) = make64(lo, hi); \
}
#endif /* MINIX_64BIT */
#endif /* SHA2_BYTE_ORDER == SHA2_LITTLE_ENDIAN */

/*
 * Macro for incrementally adding the unsigned 64-bit integer n to the
 * unsigned 128-bit integer (represented using a two-element array of
 * 64-bit words):
 */
#define ADDINC128(w,n) { \
    (w)[0] += (sha2_word64)(n); \
    if ((w)[0] < (n)) { \
        (w)[1]++; \
    } \
}

/** THE SIX LOGICAL FUNCTIONS *****/
/*
 * Bit shifting and rotation (used by the six SHA-XYZ logical functions:
 *
 * NOTE: The naming of R and S appears backwards here (R is a SHIFT and
 * S is a ROTATION) because the SHA-256/384/512 description document
 * (see http://csrc.nist.gov/cryptval/shs/sha256-384-512.pdf) uses this
 * same "backwards" definition.
 */
/* Shift-right (used in SHA-256, SHA-384, and SHA-512): */
#define R(b,x) ((x) >> (b))
/* 32-bit Rotate-right (used in SHA-256): */
#define S32(b,x) (((x) >> (b)) | ((x) << (32 - (b))))
/* 64-bit Rotate-right (used in SHA-384 and SHA-512): */
#define S64(b,x) (((x) >> (b)) | ((x) << (64 - (b))))

/* Two of six logical functions used in SHA-256, SHA-384, and SHA-512: */
#define Ch(x,y,z) ((x) & (y)) ^ ((~(x)) & (z))
#define Maj(x,y,z) ((x) & (y)) ^ ((x) & (z)) ^ ((y) & (z))

/* Four of six logical functions used in SHA-256: */
#define Sigma0_256(x) (S32(2, (x)) ^ S32(13, (x)) ^ S32(22, (x)))
#define Sigma1_256(x) (S32(6, (x)) ^ S32(11, (x)) ^ S32(25, (x)))
#define sigma0_256(x) (S32(7, (x)) ^ S32(18, (x)) ^ R(3, (x)))
#define sigma1_256(x) (S32(17, (x)) ^ S32(19, (x)) ^ R(10, (x)))

/* Four of six logical functions used in SHA-384 and SHA-512: */
#define Sigma0_512(x) (S64(28, (x)) ^ S64(34, (x)) ^ S64(39, (x)))
#define Sigma1_512(x) (S64(14, (x)) ^ S64(18, (x)) ^ S64(41, (x)))
#define sigma0_512(x) (S64( 1, (x)) ^ S64( 8, (x)) ^ R( 7, (x)))
#define sigma1_512(x) (S64(19, (x)) ^ S64(61, (x)) ^ R( 6, (x)))

/** INTERNAL FUNCTION PROTOTYPES *****/
/* NOTE: These should not be accessed directly from outside this
 * library -- they are intended for private internal visibility/use
 * only.
 */
void SHA512_Last(SHA512_CTX*);
void SHA256_Transform(SHA256_CTX*, const sha2_word32*);
void SHA512_Transform(SHA512_CTX*, const sha2_word64*);

/** SHA-XYZ INITIAL HASH VALUES AND CONSTANTS *****/
/* Hash constant words K for SHA-256: */
const static sha2_word32 K256[64] = {
    0x428a2f98UL, 0x71374491UL, 0xb5c0fbcfUL, 0xe9b5dba5UL,
    0x3956c25bUL, 0x59f111f1UL, 0x923f82a4UL, 0xab1c5ed5UL,
    0xd807aa98UL, 0x12835b01UL, 0x243185beUL, 0x550c7dc3UL,
    0x72be5d74UL, 0x80deb1feUL, 0x9bdc06a7UL, 0xc19bf174UL,

```

```
    0xe49b69c1UL, 0xefbe4786UL, 0x0fc19dc6UL, 0x240ca1ccUL,
    0x2de92c6fUL, 0x4a7484aaUL, 0x5cb0a9dcUL, 0x76f988daUL,
    0x983e5152UL, 0xa831c66dUL, 0xb00327c8UL, 0xbf597fc7UL,
    0xc6e00bf3UL, 0xd5a79147UL, 0x06ca6351UL, 0x14292967UL,
    0x27b70a85UL, 0x2e1b2138UL, 0x4d2c6dfcUL, 0x53380d13UL,
    0x650a7354UL, 0x766a0abbUL, 0x81c2c92eUL, 0x92722c85UL,
    0xa2bfe8a1UL, 0xa81a664bUL, 0xc24b8b70UL, 0xc76c51a3UL,
    0xd192e819UL, 0xd6990624UL, 0xf40e3585UL, 0x106aa070UL,
    0x19a4c116UL, 0x1e376c08UL, 0x2748774cUL, 0x34b0bcb5UL,
    0x391c0cb3UL, 0x4ed8aa4aUL, 0x5b9cca4fUL, 0x682e6ff3UL,
    0x748f82eeUL, 0x78a5636fUL, 0x84c87814UL, 0x8cc70208UL,
    0x90beffffaUL, 0xa4506cebUL, 0xbef9a3f7UL, 0xc67178f2UL
};

/* Initial hash value H for SHA-256: */
const static sha2_word32 sha256_initial_hash_value[8] = {
    0x6a09e667UL,
    0xbb67ae85UL,
    0x3c6ef372UL,
    0xa54ff53aUL,
    0x510e527fUL,
    0x9b05688cUL,
    0x1f83d9abUL,
    0x5be0cd19UL
};

#if !NO_64BIT
/* Hash constant words K for SHA-384 and SHA-512: */
const static sha2_word64 K512[80] = {
    0x428a2f98d728ae22ULL, 0x7137449123ef65cdULL,
    0xb5c0fbcfec4d3b2fULL, 0xe9b5dba58189dbbcULL,
    0x3956c25bf348b538ULL, 0x59f111f1b605d019ULL,
    0x923f82a4af194f9bULL, 0xab1c5ed5da6d8118ULL,
    0xd807aa98a3030242ULL, 0x12835b0145706fbcULL,
    0x243185be4ee4b28cULL, 0x550c7dc3d5ffb4e2ULL,
    0x72be5d74f27b896fULL, 0x80deb1fe3b1696b1ULL,
    0x9bdc06a725c71235ULL, 0xc19bf174cf692694ULL,
    0xe49b69c19ef14ad2ULL, 0xefbe4786384f25e3ULL,
    0x0fc19dc68b8cd5b5ULL, 0x240ca1cc77ac9c65ULL,
    0x2de92c6f592b0275ULL, 0x4a7484aa6ea6e483ULL,
    0x5cb0a9dcbbd41fbd4ULL, 0x76f988da831153b5ULL,
    0x983e5152ee66dfabULL, 0xa831c66d2db43210ULL,
    0xb00327c898fb213fULL, 0xbf597fc7beef0ee4ULL,
    0xc6e00bf33da88fc2ULL, 0xd5a79147930aa725ULL,
    0x06ca6351e003826fULL, 0x142929670a0e6e70ULL,
    0x27b70a8546d22ffcULL, 0x2e1b21385c26c926ULL,
    0x4d2c6dfc5ac42aedULL, 0x53380d139d95b3dfULL,
    0x650a73548baf63deULL, 0x766a0abb3c77b2a8ULL,
    0x81c2c92e47edaee6ULL, 0x92722c851482353bULL,
    0xa2bfe8a14cf10364ULL, 0xa81a664bbc423001ULL,
    0xc24b8b70d0f89791ULL, 0xc76c51a30654be30ULL,
    0xd192e819d6ef5218ULL, 0xd69906245565a910ULL,
    0xf40e35855771202aULL, 0x106aa07032bdbl8ULL,
    0x19a4c116b8d2d0c8ULL, 0x1e376c085141ab53ULL,
    0x2748774cdf8eeb99ULL, 0x34b0bcb5e19b48a8ULL,
    0x391c0cb3c5c95a63ULL, 0x4ed8aa4ae3418acbULL,
    0x5b9cca4f7763e373ULL, 0x682e6ff3d6b2b8a3ULL,
    0x748f82ee5defb2fcULL, 0x78a5636f43172f60ULL,
    0x84c87814a1f0ab72ULL, 0x8cc702081a6439ecULL,
    0x90beffffa23631e28ULL, 0xa4506cebde82bde9ULL,
    0xbef9a3f7b2c67915ULL, 0xc67178f2e372532bULL,
    0xca273eceeaa26619cULL, 0xd186b8c721c0c207ULL,
    0xeda7dd6cde0ebl1eULL, 0xf57d4f7fee6ed178ULL,
    0x06f067aa72176fbaULL, 0xa637dc5a2c898a6ULL,
    0x113f9804bef90daeULL, 0x1b710b35131c471bULL,
    0x28db77f523047d84ULL, 0x32caab7b40c72493ULL,
    0x3c9ebe0a15c9bebcULL, 0x431d67c49c100d4cULL,
    0x4cc5d4becb3e42b6ULL, 0x597f299cfc657e2aULL,
    0x5fcb6fab3ad6faecULL, 0x6c44198c4a475817ULL
};

/* Initial hash value H for SHA-384 */
const static sha2_word64 sha384_initial_hash_value[8] = {
    0xcbbb9d5dc1059ed8ULL,
```

```
    0x629a292a367cd507ULL,
    0x9159015a3070dd17ULL,
    0x152fec8f70e5939ULL,
    0x67332667ffc00b31ULL,
    0x8eb44a8768581511ULL,
    0xdb0c2e0d64f98fa7ULL,
    0x47b5481dbefa4fa4ULL
};

/* Initial hash value H for SHA-512 */
const static sha2_word64 sha512_initial_hash_value[8] = {
    0x6a09e667f3bcc908ULL,
    0xbb67ae8584caa73bULL,
    0x3c6ef372fe94f82bULL,
    0xa54ff53a5f1d36f1ULL,
    0x510e527fade682d1ULL,
    0x9b05688c2b3e6c1fULL,
    0x1f83d9abfb41bd6bULL,
    0x5be0cd19137e2179ULL
};
#endif /* !NO_64BIT */

/*
 * Constant used by SHA256/384/512_End() functions for converting the
 * digest to a readable hexadecimal character string:
 */
static const char *sha2_hex_digits = "0123456789abcdef";

/** SHA-256: *****/
void SHA256_Init(SHA256_CTX* context) {
    if (context == (SHA256_CTX*)0) {
        return;
    }
    bcopy(sha256_initial_hash_value, context->state, SHA256_DIGEST_LENGTH);
    bzero(context->buffer, SHA256_BLOCK_LENGTH);
#if MINIX_64BIT
    context->bitcount = cvu64(0);
#else /* !MINIX_64BIT */
    context->bitcount = 0;
#endif /* MINIX_64BIT */
}

#ifdef SHA2_UNROLL_TRANSFORM

/* Unrolled SHA-256 round macros: */

#if SHA2_BYTE_ORDER == SHA2_LITTLE_ENDIAN

#define ROUND256_0_TO_15(a,b,c,d,e,f,g,h) \
    REVERSE32(*data++, W256[j]); \
    T1 = (h) + Sigma1_256(e) + Ch((e), (f), (g)) + \
        K256[j] + W256[j]; \
    (d) += T1; \
    (h) = T1 + Sigma0_256(a) + Maj((a), (b), (c)); \
    j++

#else /* SHA2_BYTE_ORDER == SHA2_LITTLE_ENDIAN */

#define ROUND256_0_TO_15(a,b,c,d,e,f,g,h) \
    T1 = (h) + Sigma1_256(e) + Ch((e), (f), (g)) + \
        K256[j] + (W256[j] = *data++); \
    (d) += T1; \
    (h) = T1 + Sigma0_256(a) + Maj((a), (b), (c)); \
    j++

#endif

#endif /* SHA2_BYTE_ORDER == SHA2_LITTLE_ENDIAN */

#define ROUND256(a,b,c,d,e,f,g,h) \
    s0 = W256[(j+1)&0x0f]; \
    s0 = sigma0_256(s0); \
    s1 = W256[(j+14)&0x0f]; \
    s1 = sigma1_256(s1); \
    T1 = (h) + Sigma1_256(e) + Ch((e), (f), (g)) + K256[j] + \
        (W256[j&0x0f] += s1 + W256[(j+9)&0x0f] + s0); \
```

```

    (d) += T1; \
    (h) = T1 + Sigma0_256(a) + Maj((a), (b), (c)); \
    j++
}

void SHA256_Transform(SHA256_CTX* context, const sha2_word32* data) {
    sha2_word32    a, b, c, d, e, f, g, h, s0, s1;
    sha2_word32    T1, *W256;
    int            j;

    W256 = (sha2_word32*)context->buffer;

    /* Initialize registers with the prev. intermediate value */
    a = context->state[0];
    b = context->state[1];
    c = context->state[2];
    d = context->state[3];
    e = context->state[4];
    f = context->state[5];
    g = context->state[6];
    h = context->state[7];

    j = 0;
    do {
        /* Rounds 0 to 15 (unrolled): */
        ROUND256_0_TO_15(a,b,c,d,e,f,g,h);
        ROUND256_0_TO_15(h,a,b,c,d,e,f,g);
        ROUND256_0_TO_15(g,h,a,b,c,d,e,f);
        ROUND256_0_TO_15(f,g,h,a,b,c,d,e);
        ROUND256_0_TO_15(e,f,g,h,a,b,c,d);
        ROUND256_0_TO_15(d,e,f,g,h,a,b,c);
        ROUND256_0_TO_15(c,d,e,f,g,h,a,b);
        ROUND256_0_TO_15(b,c,d,e,f,g,h,a);
    } while (j < 16);

    /* Now for the remaining rounds to 64: */
    do {
        ROUND256(a,b,c,d,e,f,g,h);
        ROUND256(h,a,b,c,d,e,f,g);
        ROUND256(g,h,a,b,c,d,e,f);
        ROUND256(f,g,h,a,b,c,d,e);
        ROUND256(e,f,g,h,a,b,c,d);
        ROUND256(d,e,f,g,h,a,b,c);
        ROUND256(c,d,e,f,g,h,a,b);
        ROUND256(b,c,d,e,f,g,h,a);
    } while (j < 64);

    /* Compute the current intermediate hash value */
    context->state[0] += a;
    context->state[1] += b;
    context->state[2] += c;
    context->state[3] += d;
    context->state[4] += e;
    context->state[5] += f;
    context->state[6] += g;
    context->state[7] += h;

    /* Clean up */
    a = b = c = d = e = f = g = h = T1 = 0;
}

#else /* SHA2_UNROLL_TRANSFORM */

void SHA256_Transform(SHA256_CTX* context, const sha2_word32* data) {
    sha2_word32    a, b, c, d, e, f, g, h, s0, s1;
    sha2_word32    T1, T2, *W256;
    int            j;

    W256 = (sha2_word32*)context->buffer;

    /* Initialize registers with the prev. intermediate value */
    a = context->state[0];
    b = context->state[1];
    c = context->state[2];
    d = context->state[3];

```



```

    e = context->state[4];
    f = context->state[5];
    g = context->state[6];
    h = context->state[7];

    j = 0;
    do {
#ife SHA2_BYTE_ORDER == SHA2_LITTLE_ENDIAN
        /* Copy data while converting to host byte order */
        REVERSE32(*data++, W256[j]);
        /* Apply the SHA-256 compression function to update a..h */
        T1 = h + Sigma1_256(e) + Ch(e, f, g) + K256[j] + W256[j];
#else /* SHA2_BYTE_ORDER == SHA2_LITTLE_ENDIAN */
        /* Apply the SHA-256 compression function to update a..h with copy */
        T1 = h + Sigma1_256(e) + Ch(e, f, g) + K256[j] + (W256[j] = *data++);
#endif /* SHA2_BYTE_ORDER == SHA2_LITTLE_ENDIAN */
        T2 = Sigma0_256(a) + Maj(a, b, c);
        h = g;
        g = f;
        f = e;
        e = d + T1;
        d = c;
        c = b;
        b = a;
        a = T1 + T2;

        j++;
    } while (j < 16);

    do {
        /* Part of the message block expansion: */
        s0 = W256[(j+1)&0x0f];
        s0 = sigma0_256(s0);
        s1 = W256[(j+14)&0x0f];
        s1 = sigma1_256(s1);

        /* Apply the SHA-256 compression function to update a..h */
        T1 = h + Sigma1_256(e) + Ch(e, f, g) + K256[j] +
            (W256[j&0x0f] += s1 + W256[(j+9)&0x0f] + s0);
        T2 = Sigma0_256(a) + Maj(a, b, c);
        h = g;
        g = f;
        f = e;
        e = d + T1;
        d = c;
        c = b;
        b = a;
        a = T1 + T2;

        j++;
    } while (j < 64);

    /* Compute the current intermediate hash value */
    context->state[0] += a;
    context->state[1] += b;
    context->state[2] += c;
    context->state[3] += d;
    context->state[4] += e;
    context->state[5] += f;
    context->state[6] += g;
    context->state[7] += h;

    /* Clean up */
    a = b = c = d = e = f = g = h = T1 = T2 = 0;
}

#endif /* SHA2_UNROLL_TRANSFORM */

void SHA256_Update(SHA256_CTX* context, const sha2_byte *data, size_t len) {
    unsigned int    freespace, usedspace;

    if (len == 0) {
        /* Calling with no data is valid - we do nothing */
        return;
    }

```

```

    }

    /* Sanity check: */
    assert(context != (SHA256_CTX*)0 && data != (sha2_byte*)0);

#if MINIX_64BIT
    usedspace= rem64u(context->bitcount, SHA256_BLOCK_LENGTH*8)/8;
#else /* !MINIX_64BIT */
    usedspace = (context->bitcount >> 3) % SHA256_BLOCK_LENGTH;
#endif /* MINIX_64BIT */
    if (usedspace > 0) {
        /* Calculate how much free space is available in the buffer */
        freespace = SHA256_BLOCK_LENGTH - usedspace;

        if (len >= freespace) {
            /* Fill the buffer completely and process it */
            bcopy(data, &context->buffer[usedspace], freespace);
#if MINIX_64BIT
            context->bitcount= add64u(context->bitcount,
                                     freespace << 3);
#else /* !MINIX_64BIT */
            context->bitcount += freespace << 3;
#endif /* MINIX_64BIT */
            len -= freespace;
            data += freespace;
            SHA256_Transform(context, (sha2_word32*)context->buffer);
        } else {
            /* The buffer is not yet full */
            bcopy(data, &context->buffer[usedspace], len);
#if MINIX_64BIT
            context->bitcount= add64u(context->bitcount, len << 3);
#else /* !MINIX_64BIT */
            context->bitcount += len << 3;
#endif /* MINIX_64BIT */
            /* Clean up: */
            usedspace = freespace = 0;
            return;
        }
    }
    while (len >= SHA256_BLOCK_LENGTH) {
        /* Process as many complete blocks as we can */
        SHA256_Transform(context, (const sha2_word32*)data);
#if MINIX_64BIT
        context->bitcount= add64u(context->bitcount,
                                SHA256_BLOCK_LENGTH << 3);
#else /* !MINIX_64BIT */
        context->bitcount += SHA256_BLOCK_LENGTH << 3;
#endif /* MINIX_64BIT */
        len -= SHA256_BLOCK_LENGTH;
        data += SHA256_BLOCK_LENGTH;
    }
    if (len > 0) {
        /* There's left-overs, so save 'em */
        bcopy(data, context->buffer, len);
#if MINIX_64BIT
        context->bitcount= add64u(context->bitcount, len << 3);
#else /* !MINIX_64BIT */
        context->bitcount += len << 3;
#endif /* MINIX_64BIT */
    }
    /* Clean up: */
    usedspace = freespace = 0;
}

void SHA256_Final(sha2_byte digest[], SHA256_CTX* context) {
    sha2_word32 *d = (sha2_word32*)digest;
    unsigned int usedspace;

    /* Sanity check: */
    assert(context != (SHA256_CTX*)0);

    /* If no digest buffer is passed, we don't bother doing this: */
    if (digest != (sha2_byte*)0) {
#if MINIX_64BIT

```

```

        usedspace= rem64u(context->bitcount, SHA256_BLOCK_LENGTH*8)/8;
#else /* !MINIX_64BIT */
        usedspace = (context->bitcount >> 3) % SHA256_BLOCK_LENGTH;
#endif /* MINIX_64BIT */
#if SHA2_BYTE_ORDER == SHA2_LITTLE_ENDIAN
    /* Convert FROM host byte order */
    REVERSE64(context->bitcount, context->bitcount);
#endif

    if (usedspace > 0) {
        /* Begin padding with a 1 bit: */
        context->buffer[usedspace++] = 0x80;

        if (usedspace <= SHA256_SHORT_BLOCK_LENGTH) {
            /* Set-up for the last transform: */
            bzero(&context->buffer[usedspace], SHA256_SHORT_BLOCK_LEN
GTH - usedspace);
        } else {
            if (usedspace < SHA256_BLOCK_LENGTH) {
                bzero(&context->buffer[usedspace], SHA256_BLOCK_L
ENGTH - usedspace);
            }
            /* Do second-to-last transform: */
            SHA256_Transform(context, (sha2_word32*)context->buffer);

            /* And set-up for the last transform: */
            bzero(context->buffer, SHA256_SHORT_BLOCK_LENGTH);
        }
    } else {
        /* Set-up for the last transform: */
        bzero(context->buffer, SHA256_SHORT_BLOCK_LENGTH);

        /* Begin padding with a 1 bit: */
        *context->buffer = 0x80;
    }
    /* Set the bit count: */
    *(sha2_word64*)&context->buffer[SHA256_SHORT_BLOCK_LENGTH] = context->bit
count;

    /* Final transform: */
    SHA256_Transform(context, (sha2_word32*)context->buffer);

#if SHA2_BYTE_ORDER == SHA2_LITTLE_ENDIAN
    {
        /* Convert TO host byte order */
        int j;
        for (j = 0; j < 8; j++) {
            REVERSE32(context->state[j], context->state[j]);
            *d++ = context->state[j];
        }
    }
#else
    bcopy(context->state, d, SHA256_DIGEST_LENGTH);
#endif
}

/* Clean up state data: */
bzero(context, sizeof(context));
usedspace = 0;
}

char *SHA256_End(SHA256_CTX* context, char buffer[]) {
    sha2_byte digest[SHA256_DIGEST_LENGTH], *d = digest;
    int i;

    /* Sanity check: */
    assert(context != (SHA256_CTX*)0);

    if (buffer != (char*)0) {
        SHA256_Final(digest, context);

        for (i = 0; i < SHA256_DIGEST_LENGTH; i++) {
            *buffer++ = sha2_hex_digits[(d & 0xf0) >> 4];
            *buffer++ = sha2_hex_digits[d & 0x0f];
            d++;
        }
    }
}

```

```

    }
    *buffer = (char)0;
} else {
    bzero(context, sizeof(context));
}
bzero(digest, SHA256_DIGEST_LENGTH);
return buffer;
}

char* SHA256_Data(const sha2_byte* data, size_t len, char digest[SHA256_DIGEST_STRING_LENGTH]) {
    SHA256_CTX context;

    SHA256_Init(&context);
    SHA256_Update(&context, data, len);
    return SHA256_End(&context, digest);
}

#if !NO_64BIT

/** SHA-512: *****/
void SHA512_Init(SHA512_CTX* context) {
    if (context == (SHA512_CTX*)0) {
        return;
    }
    bcopy(sha512_initial_hash_value, context->state, SHA512_DIGEST_LENGTH);
    bzero(context->buffer, SHA512_BLOCK_LENGTH);
    context->bitcount[0] = context->bitcount[1] = 0;
}

#ifdef SHA2_UNROLL_TRANSFORM

/* Unrolled SHA-512 round macros: */
#if SHA2_BYTE_ORDER == SHA2_LITTLE_ENDIAN

#define ROUND512_0_TO_15(a,b,c,d,e,f,g,h) \
    REVERSE64(*data++, W512[j]); \
    T1 = (h) + Sigma1_512(e) + Ch((e), (f), (g)) + \
        K512[j] + W512[j]; \
    (d) += T1; \
    (h) = T1 + Sigma0_512(a) + Maj((a), (b), (c)); \
    j++

#else /* SHA2_BYTE_ORDER == SHA2_LITTLE_ENDIAN */

#define ROUND512_0_TO_15(a,b,c,d,e,f,g,h) \
    T1 = (h) + Sigma1_512(e) + Ch((e), (f), (g)) + \
        K512[j] + (W512[j] = *data++); \
    (d) += T1; \
    (h) = T1 + Sigma0_512(a) + Maj((a), (b), (c)); \
    j++

#endif

#define ROUND512(a,b,c,d,e,f,g,h) \
    s0 = W512[(j+1)&0x0f]; \
    s0 = sigma0_512(s0); \
    s1 = W512[(j+14)&0x0f]; \
    s1 = sigma1_512(s1); \
    T1 = (h) + Sigma1_512(e) + Ch((e), (f), (g)) + K512[j] + \
        (W512[j&0x0f] += s1 + W512[(j+9)&0x0f] + s0); \
    (d) += T1; \
    (h) = T1 + Sigma0_512(a) + Maj((a), (b), (c)); \
    j++

void SHA512_Transform(SHA512_CTX* context, const sha2_word64* data) {
    sha2_word64 a, b, c, d, e, f, g, h, s0, s1;
    sha2_word64 T1, *W512 = (sha2_word64*)context->buffer;
    int j;

    /* Initialize registers with the prev. intermediate value */
    a = context->state[0];
    b = context->state[1];
    c = context->state[2];

```

```

d = context->state[3];
e = context->state[4];
f = context->state[5];
g = context->state[6];
h = context->state[7];

j = 0;
do {
    ROUND512_0_TO_15(a,b,c,d,e,f,g,h);
    ROUND512_0_TO_15(h,a,b,c,d,e,f,g);
    ROUND512_0_TO_15(g,h,a,b,c,d,e,f);
    ROUND512_0_TO_15(f,g,h,a,b,c,d,e);
    ROUND512_0_TO_15(e,f,g,h,a,b,c,d);
    ROUND512_0_TO_15(d,e,f,g,h,a,b,c);
    ROUND512_0_TO_15(c,d,e,f,g,h,a,b);
    ROUND512_0_TO_15(b,c,d,e,f,g,h,a);
} while (j < 16);

/* Now for the remaining rounds up to 79: */
do {
    ROUND512(a,b,c,d,e,f,g,h);
    ROUND512(h,a,b,c,d,e,f,g);
    ROUND512(g,h,a,b,c,d,e,f);
    ROUND512(f,g,h,a,b,c,d,e);
    ROUND512(e,f,g,h,a,b,c,d);
    ROUND512(d,e,f,g,h,a,b,c);
    ROUND512(c,d,e,f,g,h,a,b);
    ROUND512(b,c,d,e,f,g,h,a);
} while (j < 80);

/* Compute the current intermediate hash value */
context->state[0] += a;
context->state[1] += b;
context->state[2] += c;
context->state[3] += d;
context->state[4] += e;
context->state[5] += f;
context->state[6] += g;
context->state[7] += h;

/* Clean up */
a = b = c = d = e = f = g = h = T1 = 0;
}

#else /* SHA2_UNROLL_TRANSFORM */

void SHA512_Transform(SHA512_CTX* context, const sha2_word64* data) {
    sha2_word64    a, b, c, d, e, f, g, h, s0, s1;
    sha2_word64    T1, T2, *W512 = (sha2_word64*)context->buffer;
    int            j;

    /* Initialize registers with the prev. intermediate value */
    a = context->state[0];
    b = context->state[1];
    c = context->state[2];
    d = context->state[3];
    e = context->state[4];
    f = context->state[5];
    g = context->state[6];
    h = context->state[7];

    j = 0;
    do {
#ifdef SHA2_BYTE_ORDER == SHA2_LITTLE_ENDIAN
        /* Convert TO host byte order */
        REVERSE64(*data++, W512[j]);
        /* Apply the SHA-512 compression function to update a..h */
        T1 = h + Sigma1_512(e) + Ch(e, f, g) + K512[j] + W512[j];
#else /* SHA2_BYTE_ORDER == SHA2_LITTLE_ENDIAN */
        /* Apply the SHA-512 compression function to update a..h with copy */
        T1 = h + Sigma1_512(e) + Ch(e, f, g) + K512[j] + (W512[j] = *data++);
#endif /* SHA2_BYTE_ORDER == SHA2_LITTLE_ENDIAN */
        T2 = Sigma0_512(a) + Maj(a, b, c);
        h = g;

```

```

        g = f;
        f = e;
        e = d + T1;
        d = c;
        c = b;
        b = a;
        a = T1 + T2;

        j++;
    } while (j < 16);

do {
    /* Part of the message block expansion: */
    s0 = W512[(j+1)&0x0f];
    s0 = sigma0_512(s0);
    s1 = W512[(j+14)&0x0f];
    s1 = sigma1_512(s1);

    /* Apply the SHA-512 compression function to update a..h */
    T1 = h + Sigma1_512(e) + Ch(e, f, g) + K512[j] +
        (W512[j&0x0f] += s1 + W512[(j+9)&0x0f] + s0);
    T2 = Sigma0_512(a) + Maj(a, b, c);
    h = g;
    g = f;
    f = e;
    e = d + T1;
    d = c;
    c = b;
    b = a;
    a = T1 + T2;

    j++;
} while (j < 80);

/* Compute the current intermediate hash value */
context->state[0] += a;
context->state[1] += b;
context->state[2] += c;
context->state[3] += d;
context->state[4] += e;
context->state[5] += f;
context->state[6] += g;
context->state[7] += h;

/* Clean up */
a = b = c = d = e = f = g = h = T1 = T2 = 0;
}

#endif /* SHA2_UNROLL_TRANSFORM */

void SHA512_Update(SHA512_CTX* context, const sha2_byte *data, size_t len) {
    unsigned int    freespace, usedspace;

    if (len == 0) {
        /* Calling with no data is valid - we do nothing */
        return;
    }

    /* Sanity check: */
    assert(context != (SHA512_CTX*)0 && data != (sha2_byte*)0);

    usedspace = (context->bitcount[0] >> 3) % SHA512_BLOCK_LENGTH;
    if (usedspace > 0) {
        /* Calculate how much free space is available in the buffer */
        freespace = SHA512_BLOCK_LENGTH - usedspace;

        if (len >= freespace) {
            /* Fill the buffer completely and process it */
            bcopy(data, &context->buffer[usedspace], freespace);
            ADDINC128(context->bitcount, freespace << 3);
            len -= freespace;
            data += freespace;
            SHA512_Transform(context, (sha2_word64*)context->buffer);
        } else {

```

```

        /* The buffer is not yet full */
        bcopy(data, &context->buffer[usedspace], len);
        ADDINC128(context->bitcount, len << 3);
        /* Clean up: */
        usedspace = freespace = 0;
        return;
    }
}
while (len >= SHA512_BLOCK_LENGTH) {
    /* Process as many complete blocks as we can */
    SHA512_Transform(context, (const sha2_word64*)data);
    ADDINC128(context->bitcount, SHA512_BLOCK_LENGTH << 3);
    len -= SHA512_BLOCK_LENGTH;
    data += SHA512_BLOCK_LENGTH;
}
if (len > 0) {
    /* There's left-overs, so save 'em */
    bcopy(data, context->buffer, len);
    ADDINC128(context->bitcount, len << 3);
}
/* Clean up: */
usedspace = freespace = 0;
}

void SHA512_Last(SHA512_CTX* context) {
    unsigned int    usedspace;

    usedspace = (context->bitcount[0] >> 3) % SHA512_BLOCK_LENGTH;
#ifdef SHA2_BYTE_ORDER == SHA2_LITTLE_ENDIAN
    /* Convert FROM host byte order */
    REVERSE64(context->bitcount[0], context->bitcount[0]);
    REVERSE64(context->bitcount[1], context->bitcount[1]);
#endif
    if (usedspace > 0) {
        /* Begin padding with a 1 bit: */
        context->buffer[usedspace++] = 0x80;

        if (usedspace <= SHA512_SHORT_BLOCK_LENGTH) {
            /* Set-up for the last transform: */
            bzero(&context->buffer[usedspace], SHA512_SHORT_BLOCK_LENGTH - us
edgespace);
        } else {
            if (usedspace < SHA512_BLOCK_LENGTH) {
                bzero(&context->buffer[usedspace], SHA512_BLOCK_LENGTH -
usedspace);
            }
            /* Do second-to-last transform: */
            SHA512_Transform(context, (sha2_word64*)context->buffer);

            /* And set-up for the last transform: */
            bzero(context->buffer, SHA512_BLOCK_LENGTH - 2);
        }
    } else {
        /* Prepare for final transform: */
        bzero(context->buffer, SHA512_SHORT_BLOCK_LENGTH);

        /* Begin padding with a 1 bit: */
        *context->buffer = 0x80;
    }
    /* Store the length of input data (in bits): */
    *(sha2_word64*)&context->buffer[SHA512_SHORT_BLOCK_LENGTH] = context->bitcount[1]
;
    *(sha2_word64*)&context->buffer[SHA512_SHORT_BLOCK_LENGTH+8] = context->bitcount[
0];

    /* Final transform: */
    SHA512_Transform(context, (sha2_word64*)context->buffer);
}

void SHA512_Final(sha2_byte digest[], SHA512_CTX* context) {
    sha2_word64    *d = (sha2_word64*)digest;

    /* Sanity check: */
    assert(context != (SHA512_CTX*)0);

```

```

    /* If no digest buffer is passed, we don't bother doing this: */
    if (digest != (sha2_byte*)0) {
        SHA512_Last(context);

        /* Save the hash data for output: */
#ifdef SHA2_BYTE_ORDER == SHA2_LITTLE_ENDIAN
        {
            /* Convert TO host byte order */
            int j;
            for (j = 0; j < 8; j++) {
                REVERSE64(context->state[j], context->state[j]);
                *d++ = context->state[j];
            }
        }
#else
        bcopy(context->state, d, SHA512_DIGEST_LENGTH);
#endif
    }

    /* Zero out state data */
    bzero(context, sizeof(context));
}

char *SHA512_End(SHA512_CTX* context, char buffer[]) {
    sha2_byte digest[SHA512_DIGEST_LENGTH], *d = digest;
    int i;

    /* Sanity check: */
    assert(context != (SHA512_CTX*)0);

    if (buffer != (char*)0) {
        SHA512_Final(digest, context);

        for (i = 0; i < SHA512_DIGEST_LENGTH; i++) {
            *buffer++ = sha2_hex_digits[(*d & 0xf0) >> 4];
            *buffer++ = sha2_hex_digits[*d & 0x0f];
            d++;
        }
        *buffer = (char)0;
    } else {
        bzero(context, sizeof(context));
    }
    bzero(digest, SHA512_DIGEST_LENGTH);
    return buffer;
}

char* SHA512_Data(const sha2_byte* data, size_t len, char digest[SHA512_DIGEST_STRING_LENGTH]) {
    SHA512_CTX context;

    SHA512_Init(&context);
    SHA512_Update(&context, data, len);
    return SHA512_End(&context, digest);
}

/** SHA-384: *****/
void SHA384_Init(SHA384_CTX* context) {
    if (context == (SHA384_CTX*)0) {
        return;
    }
    bcopy(sha384_initial_hash_value, context->state, SHA512_DIGEST_LENGTH);
    bzero(context->buffer, SHA384_BLOCK_LENGTH);
    context->bitcount[0] = context->bitcount[1] = 0;
}

void SHA384_Update(SHA384_CTX* context, const sha2_byte* data, size_t len) {
    SHA512_Update((SHA512_CTX*)context, data, len);
}

void SHA384_Final(sha2_byte digest[], SHA384_CTX* context) {
    sha2_word64 *d = (sha2_word64*)digest;

    /* Sanity check: */

```



```

    assert(context != (SHA384_CTX*)0);

    /* If no digest buffer is passed, we don't bother doing this: */
    if (digest != (sha2_byte*)0) {
        SHA512_Last((SHA512_CTX*)context);

        /* Save the hash data for output: */
#ifdef SHA2_BYTE_ORDER == SHA2_LITTLE_ENDIAN
        {
            /* Convert TO host byte order */
            int j;
            for (j = 0; j < 6; j++) {
                REVERSE64(context->state[j], context->state[j]);
                *d++ = context->state[j];
            }
        }
#else
        bcopy(context->state, d, SHA384_DIGEST_LENGTH);
#endif
    }

    /* Zero out state data */
    bzero(context, sizeof(context));
}

char *SHA384_End(SHA384_CTX* context, char buffer[]) {
    sha2_byte digest[SHA384_DIGEST_LENGTH], *d = digest;
    int i;

    /* Sanity check: */
    assert(context != (SHA384_CTX*)0);

    if (buffer != (char*)0) {
        SHA384_Final(digest, context);

        for (i = 0; i < SHA384_DIGEST_LENGTH; i++) {
            *buffer++ = sha2_hex_digits[(*d & 0xf0) >> 4];
            *buffer++ = sha2_hex_digits[*d & 0x0f];
            d++;
        }
        *buffer = (char)0;
    } else {
        bzero(context, sizeof(context));
    }
    bzero(digest, SHA384_DIGEST_LENGTH);
    return buffer;
}

char* SHA384_Data(const sha2_byte* data, size_t len, char digest[SHA384_DIGEST_STRING_LENGTH]) {
    SHA384_CTX context;

    SHA384_Init(&context);
    SHA384_Update(&context, data, len);
    return SHA384_End(&context, digest);
}

#endif /* !NO_64BIT */

/*
 * $PchId: sha2.c,v 1.1 2005/06/28 14:29:23 philip Exp $
 */

```

```
/*      $FreeBSD: src/sys/crypto/sha2/sha2.h,v 1.1.2.1 2001/07/03 11:01:36 ume Exp $      */
/*
/*      $KAME: sha2.h,v 1.3 2001/03/12 08:27:48 itojun Exp $      */
/*
* sha2.h
*
* Version 1.0.0beta1
*
* Written by Aaron D. Gifford <me@aarongifford.com>
*
* Copyright 2000 Aaron D. Gifford. All rights reserved.
*
* Redistribution and use in source and binary forms, with or without
* modification, are permitted provided that the following conditions
* are met:
* 1. Redistributions of source code must retain the above copyright
* notice, this list of conditions and the following disclaimer.
* 2. Redistributions in binary form must reproduce the above copyright
* notice, this list of conditions and the following disclaimer in the
* documentation and/or other materials provided with the distribution.
* 3. Neither the name of the copyright holder nor the names of contributors
* may be used to endorse or promote products derived from this software
* without specific prior written permission.
*
* THIS SOFTWARE IS PROVIDED BY THE AUTHOR(S) AND CONTRIBUTOR(S) ``AS IS'' AND
* ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE
* IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE
* ARE DISCLAIMED. IN NO EVENT SHALL THE AUTHOR(S) OR CONTRIBUTOR(S) BE LIABLE
* FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL
* DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS
* OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION)
* HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT
* LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY
* OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF
* SUCH DAMAGE.
*/

#ifdef __SHA2_H__
#define __SHA2_H__

#ifdef __cplusplus
extern "C" {
#endif

/** SHA-256/384/512 Various Length Definitions *****/
#define SHA256_BLOCK_LENGTH 64
#define SHA256_DIGEST_LENGTH 32
#define SHA256_DIGEST_STRING_LENGTH (SHA256_DIGEST_LENGTH * 2 + 1)
#define SHA384_BLOCK_LENGTH 128
#define SHA384_DIGEST_LENGTH 48
#define SHA384_DIGEST_STRING_LENGTH (SHA384_DIGEST_LENGTH * 2 + 1)
#define SHA512_BLOCK_LENGTH 128
#define SHA512_DIGEST_LENGTH 64
#define SHA512_DIGEST_STRING_LENGTH (SHA512_DIGEST_LENGTH * 2 + 1)

#ifdef __minix
#include <assert.h>
#include <string.h>
#include <sys/types.h>
#include <minix/u64.h>

typedef u8_t u_int8_t; /* 1-byte (8-bits) */
typedef u32_t u_int32_t; /* 4-bytes (32-bits) */
typedef u64_t u_int64_t; /* 8-bytes (64-bits) */

#ifdef __P
#define __P(x) x
#endif

#define NO_64BIT 1
#define MINIX_64BIT 1
```

```

#define SHA2_BYTE_ORDER      0x04030201
#define SHA2_LITTLE_ENDIAN   0x04030201
#define SHA2_BIG_ENDIAN      0x01020204
#define bcopy(s,d,l)         (memmove((d),(s),(l)))
#define bzero(d,l)           (memset((d), '0', (l)))
#endif

/** SHA-256/384/512 Context Structures *****/
/* NOTE: If your architecture does not define either u_intXX_t types or
 * uintXX_t (from inttypes.h), you may need to define things by hand
 * for your system:
 */
#if 0
typedef unsigned char u_int8_t;      /* 1-byte (8-bits) */
typedef unsigned int u_int32_t;      /* 4-bytes (32-bits) */
typedef unsigned long long u_int64_t; /* 8-bytes (64-bits) */
#endif
/*
 * Most BSD systems already define u_intXX_t types, as does Linux.
 * Some systems, however, like Compaq's Tru64 Unix instead can use
 * uintXX_t types defined by very recent ANSI C standards and included
 * in the file:
 *
 * #include <inttypes.h>
 *
 * If you choose to use <inttypes.h> then please define:
 *
 * #define SHA2_USE_INTTYPES_H
 *
 * Or on the command line during compile:
 *
 * cc -DSHA2_USE_INTTYPES_H ...
 */
#if 0 /*def SHA2_USE_INTTYPES_H*/

typedef struct _SHA256_CTX {
    uint32_t      state[8];
    uint64_t      bitcount;
    uint8_t buffer[SHA256_BLOCK_LENGTH];
} SHA256_CTX;
typedef struct _SHA512_CTX {
    uint64_t      state[8];
    uint64_t      bitcount[2];
    uint8_t buffer[SHA512_BLOCK_LENGTH];
} SHA512_CTX;

#else /* SHA2_USE_INTTYPES_H */

typedef struct _SHA256_CTX {
    u_int32_t      state[8];
    u_int64_t      bitcount;
    u_int8_t      buffer[SHA256_BLOCK_LENGTH];
} SHA256_CTX;
typedef struct _SHA512_CTX {
    u_int64_t      state[8];
    u_int64_t      bitcount[2];
    u_int8_t      buffer[SHA512_BLOCK_LENGTH];
} SHA512_CTX;

#endif /* SHA2_USE_INTTYPES_H */

typedef SHA512_CTX SHA384_CTX;

/** SHA-256/384/512 Function Prototypes *****/

void SHA256_Init __P((SHA256_CTX *));
void SHA256_Update __P((SHA256_CTX*, const u_int8_t*, size_t));
void SHA256_Final __P((u_int8_t[SHA256_DIGEST_LENGTH], SHA256_CTX*));
char* SHA256_End __P((SHA256_CTX*, char[SHA256_DIGEST_STRING_LENGTH]));
char* SHA256_Data __P((const u_int8_t*, size_t, char[SHA256_DIGEST_STRING_LENGTH]));

void SHA384_Init __P((SHA384_CTX*));
void SHA384_Update __P((SHA384_CTX*, const u_int8_t*, size_t));
void SHA384_Final __P((u_int8_t[SHA384_DIGEST_LENGTH], SHA384_CTX*));

```

```
char* SHA384_End __P((SHA384_CTX*, char[SHA384_DIGEST_STRING_LENGTH]));
char* SHA384_Data __P((const u_int8_t*, size_t, char[SHA384_DIGEST_STRING_LENGTH]));

void SHA512_Init __P((SHA512_CTX*));
void SHA512_Update __P((SHA512_CTX*, const u_int8_t*, size_t));
void SHA512_Final __P((u_int8_t[SHA512_DIGEST_LENGTH], SHA512_CTX*));
char* SHA512_End __P((SHA512_CTX*, char[SHA512_DIGEST_STRING_LENGTH]));
char* SHA512_Data __P((const u_int8_t*, size_t, char[SHA512_DIGEST_STRING_LENGTH]));

#ifdef __cplusplus
}
#endif /* __cplusplus */

#endif /* __SHA2_H__ */

/*
 * $PchId: sha2.h,v 1.1 2005/06/28 14:29:33 philip Exp $
 */
```

```
word8 S[256] = {
  99, 124, 119, 123, 242, 107, 111, 197, 48, 1, 103, 43, 254, 215, 171, 118,
  202, 130, 201, 125, 250, 89, 71, 240, 173, 212, 162, 175, 156, 164, 114, 192,
  183, 253, 147, 38, 54, 63, 247, 204, 52, 165, 229, 241, 113, 216, 49, 21,
  4, 199, 35, 195, 24, 150, 5, 154, 7, 18, 128, 226, 235, 39, 178, 117,
  9, 131, 44, 26, 27, 110, 90, 160, 82, 59, 214, 179, 41, 227, 47, 132,
  83, 209, 0, 237, 32, 252, 177, 91, 106, 203, 190, 57, 74, 76, 88, 207,
  208, 239, 170, 251, 67, 77, 51, 133, 69, 249, 2, 127, 80, 60, 159, 168,
  81, 163, 64, 143, 146, 157, 56, 245, 188, 182, 218, 33, 16, 255, 243, 210,
  205, 12, 19, 236, 95, 151, 68, 23, 196, 167, 126, 61, 100, 93, 25, 115,
  96, 129, 79, 220, 34, 42, 144, 136, 70, 238, 184, 20, 222, 94, 11, 219,
  224, 50, 58, 10, 73, 6, 36, 92, 194, 211, 172, 98, 145, 149, 228, 121,
  231, 200, 55, 109, 141, 213, 78, 169, 108, 86, 244, 234, 101, 122, 174, 8,
  186, 120, 37, 46, 28, 166, 180, 198, 232, 221, 116, 31, 75, 189, 139, 138,
  112, 62, 181, 102, 72, 3, 246, 14, 97, 53, 87, 185, 134, 193, 29, 158,
  225, 248, 152, 17, 105, 217, 142, 148, 155, 30, 135, 233, 206, 85, 40, 223,
  140, 161, 137, 13, 191, 230, 66, 104, 65, 153, 45, 15, 176, 84, 187, 22
};

#ifdef INTERMEDIATE_VALUE_KAT
word8 Si[256] = {
  82, 9, 106, 213, 48, 54, 165, 56, 191, 64, 163, 158, 129, 243, 215, 251,
  124, 227, 57, 130, 155, 47, 255, 135, 52, 142, 67, 68, 196, 222, 233, 203,
  84, 123, 148, 50, 166, 194, 35, 61, 238, 76, 149, 11, 66, 250, 195, 78,
  8, 46, 161, 102, 40, 217, 36, 178, 118, 91, 162, 73, 109, 139, 209, 37,
  114, 248, 246, 100, 134, 104, 152, 22, 212, 164, 92, 204, 93, 101, 182, 146,
  108, 112, 72, 80, 253, 237, 185, 218, 94, 21, 70, 87, 167, 141, 157, 132,
  144, 216, 171, 0, 140, 188, 211, 10, 247, 228, 88, 5, 184, 179, 69, 6,
  208, 44, 30, 143, 202, 63, 15, 2, 193, 175, 189, 3, 1, 19, 138, 107,
  58, 145, 17, 65, 79, 103, 220, 234, 151, 242, 207, 206, 240, 180, 230, 115,
  150, 172, 116, 34, 231, 173, 53, 133, 226, 249, 55, 232, 28, 117, 223, 110,
  71, 241, 26, 113, 29, 41, 197, 137, 111, 183, 98, 14, 170, 24, 190, 27,
  252, 86, 62, 75, 198, 210, 121, 32, 154, 219, 192, 254, 120, 205, 90, 244,
  31, 221, 168, 51, 136, 7, 199, 49, 177, 18, 16, 89, 39, 128, 236, 95,
  96, 81, 127, 169, 25, 181, 74, 13, 45, 229, 122, 159, 147, 201, 156, 239,
  160, 224, 59, 77, 174, 42, 245, 176, 200, 235, 187, 60, 131, 83, 153, 97,
  23, 43, 4, 126, 186, 119, 214, 38, 225, 105, 20, 99, 85, 33, 12, 125
};
#endif /* INTERMEDIATE_VALUE_KAT */

word8 Tl[256][4] = {
  {0xc6,0x63,0x63,0xa5}, {0xf8,0x7c,0x7c,0x84}, {0xee,0x77,0x77,0x99}, {0xf6,0x7b,0x7b,0x8d},
  {0xff,0xf2,0xf2,0x0d}, {0xd6,0x6b,0x6b,0xbd}, {0xde,0x6f,0x6f,0xb1}, {0x91,0xc5,0xc5,0x54},
  {0x60,0x30,0x30,0x50}, {0x02,0x01,0x01,0x03}, {0xce,0x67,0x67,0xa9}, {0x56,0x2b,0x2b,0x7d},
  {0xe7,0xfe,0xfe,0x19}, {0xb5,0xd7,0xd7,0x62}, {0x4d,0xab,0xab,0xe6}, {0xec,0x76,0x76,0x9a},
  {0x8f,0xca,0xca,0x45}, {0x1f,0x82,0x82,0x9d}, {0x89,0xc9,0xc9,0x40}, {0xfa,0x7d,0x7d,0x87},
  {0xef,0xfa,0xfa,0x15}, {0xb2,0x59,0x59,0xeb}, {0x8e,0x47,0x47,0xc9}, {0xfb,0xf0,0xf0,0x0b},
  {0x41,0xad,0xad,0xec}, {0xb3,0xd4,0xd4,0x67}, {0x5f,0xa2,0xa2,0xfd}, {0x45,0xaf,0xaf,0xea},
  {0x23,0x9c,0x9c,0xbf}, {0x53,0xa4,0xa4,0xf7}, {0xe4,0x72,0x72,0x96}, {0x9b,0xc0,0xc0,0x5b},
  {0x75,0xb7,0xb7,0xc2}, {0xe1,0xfd,0xfd,0x1c}, {0x3d,0x93,0x93,0xae}, {0x4c,0x26,0x26,0x6a},
  {0x6c,0x36,0x36,0x5a}, {0x7e,0x3f,0x3f,0x41}, {0xf5,0xf7,0xf7,0x02}, {0x83,0xcc,0xcc,0x4f},
  {0x68,0x34,0x34,0x5c}, {0x51,0xa5,0xa5,0xf4}, {0xd1,0xe5,0xe5,0x34}, {0xf9,0xf1,0xf1,0x08},
  {0xe2,0x71,0x71,0x93}, {0xab,0xd8,0xd8,0x73}, {0x62,0x31,0x31,0x53}, {0x2a,0x15,0x15,0x3f},
  {0x08,0x04,0x04,0x0c}, {0x95,0xc7,0xc7,0x52}, {0x46,0x23,0x23,0x65}, {0x9d,0xc3,0xc3,0x5e},
  {0x30,0x18,0x18,0x28}, {0x37,0x96,0x96,0xa1}, {0x0a,0x05,0x05,0x0f}, {0x2f,0x9a,0x9a,0xb5},
  {0x0e,0x07,0x07,0x09}, {0x24,0x12,0x12,0x36}, {0x1b,0x80,0x80,0x9b}, {0xdf,0xe2,0xe2,0x3d},
  {0xcd,0xeb,0xeb,0x26}, {0x4e,0x27,0x27,0x69}, {0x7f,0xb2,0xb2,0xcd}, {0xea,0x75,0x75,0x9f},
  {0x12,0x09,0x09,0x1b}, {0x1d,0x83,0x83,0x9e}, {0x58,0x2c,0x2c,0x74}, {0x34,0x1a,0x1a,0x2e}
```

, {0x36,0x1b,0x1b,0x2d}, {0xdc,0x6e,0x6e,0xb2}, {0xb4,0x5a,0x5a,0xee}, {0x5b,0xa0,0xa0,0xfb},  
, {0xa4,0x52,0x52,0xf6}, {0x76,0x3b,0x3b,0x4d}, {0xb7,0xd6,0xd6,0x61}, {0x7d,0xb3,0xb3,0xce},  
, {0x52,0x29,0x29,0x7b}, {0xdd,0xe3,0xe3,0x3e}, {0x5e,0x2f,0x2f,0x71}, {0x13,0x84,0x84,0x97},  
, {0xa6,0x53,0x53,0xf5}, {0xb9,0xd1,0xd1,0x68}, {0x00,0x00,0x00,0x00}, {0xc1,0xed,0xed,0x2c},  
, {0x40,0x20,0x20,0x60}, {0xe3,0xfc,0xfc,0x1f}, {0x79,0xb1,0xb1,0xc8}, {0xb6,0x5b,0x5b,0xed},  
, {0xd4,0x6a,0x6a,0xbe}, {0x8d,0xcb,0xcb,0x46}, {0x67,0xbe,0xbe,0xd9}, {0x72,0x39,0x39,0x4b},  
, {0x94,0x4a,0x4a,0xde}, {0x98,0x4c,0x4c,0xd4}, {0xb0,0x58,0x58,0xe8}, {0x85,0xcf,0xcf,0x4a},  
, {0xbb,0xd0,0xd0,0x6b}, {0xc5,0xef,0xef,0x2a}, {0x4f,0xaa,0xaa,0xe5}, {0xed,0xfb,0xfb,0x16},  
, {0x86,0x43,0x43,0xc5}, {0x9a,0x4d,0x4d,0xd7}, {0x66,0x33,0x33,0x55}, {0x11,0x85,0x85,0x94},  
, {0x8a,0x45,0x45,0xcf}, {0xe9,0xf9,0xf9,0x10}, {0x04,0x02,0x02,0x06}, {0xfe,0x7f,0x7f,0x81},  
, {0xa0,0x50,0x50,0xf0}, {0x78,0x3c,0x3c,0x44}, {0x25,0x9f,0x9f,0xba}, {0x4b,0xa8,0xa8,0xe3},  
, {0xa2,0x51,0x51,0xf3}, {0x5d,0xa3,0xa3,0xfe}, {0x80,0x40,0x40,0xc0}, {0x05,0x8f,0x8f,0x8a},  
, {0x3f,0x92,0x92,0xad}, {0x21,0x9d,0x9d,0xbc}, {0x70,0x38,0x38,0x48}, {0xf1,0xf5,0xf5,0x04},  
, {0x63,0xbc,0xbc,0xdf}, {0x77,0xb6,0xb6,0xc1}, {0xaf,0xda,0xda,0x75}, {0x42,0x21,0x21,0x63},  
, {0x20,0x10,0x10,0x30}, {0xe5,0xff,0xff,0x1a}, {0xfd,0xf3,0xf3,0x0e}, {0xbf,0xd2,0xd2,0x6d},  
, {0x81,0xcd,0xcd,0x4c}, {0x18,0x0c,0x0c,0x14}, {0x26,0x13,0x13,0x35}, {0xc3,0xec,0xec,0x2f},  
, {0xbe,0x5f,0x5f,0xe1}, {0x35,0x97,0x97,0xa2}, {0x88,0x44,0x44,0xcc}, {0x2e,0x17,0x17,0x39},  
, {0x93,0xc4,0xc4,0x57}, {0x55,0xa7,0xa7,0xf2}, {0xfc,0x7e,0x7e,0x82}, {0x7a,0x3d,0x3d,0x47},  
, {0xc8,0x64,0x64,0xac}, {0xba,0x5d,0x5d,0xe7}, {0x32,0x19,0x19,0x2b}, {0xe6,0x73,0x73,0x95},  
, {0xc0,0x60,0x60,0xa0}, {0x19,0x81,0x81,0x98}, {0x9e,0x4f,0x4f,0xd1}, {0xa3,0xdc,0xdc,0x7f},  
, {0x44,0x22,0x22,0x66}, {0x54,0x2a,0x2a,0x7e}, {0x3b,0x90,0x90,0xab}, {0x0b,0x88,0x88,0x83},  
, {0x8c,0x46,0x46,0xca}, {0xc7,0xee,0xee,0x29}, {0x6b,0xb8,0xb8,0xd3}, {0x28,0x14,0x14,0x3c},  
, {0xa7,0xde,0xde,0x79}, {0xbc,0x5e,0x5e,0xe2}, {0x16,0x0b,0x0b,0x1d}, {0xad,0xdb,0xdb,0x76},  
, {0xdb,0xe0,0xe0,0x3b}, {0x64,0x32,0x32,0x56}, {0x74,0x3a,0x3a,0x4e}, {0x14,0x0a,0x0a,0x1e},  
, {0x92,0x49,0x49,0xdb}, {0x0c,0x06,0x06,0x0a}, {0x48,0x24,0x24,0x6c}, {0xb8,0x5c,0x5c,0xe4},  
, {0x9f,0xc2,0xc2,0x5d}, {0xbd,0xd3,0xd3,0x6e}, {0x43,0xac,0xac,0xef}, {0xc4,0x62,0x62,0xa6},  
, {0x39,0x91,0x91,0xa8}, {0x31,0x95,0x95,0xa4}, {0xd3,0xe4,0xe4,0x37}, {0xf2,0x79,0x79,0x8b},  
, {0xd5,0xe7,0xe7,0x32}, {0x8b,0xc8,0xc8,0x43}, {0x6e,0x37,0x37,0x59}, {0xda,0x6d,0x6d,0xb7},  
, {0x01,0x8d,0x8d,0x8c}, {0xb1,0xd5,0xd5,0x64}, {0x9c,0x4e,0x4e,0xd2}, {0x49,0xa9,0xa9,0xe0},  
, {0xd8,0x6c,0x6c,0xb4}, {0xac,0x56,0x56,0xfa}, {0xf3,0xf4,0xf4,0x07}, {0xcf,0xea,0xea,0x25},  
, {0xca,0x65,0x65,0xaf}, {0xf4,0x7a,0x7a,0x8e}, {0x47,0xae,0xae,0xe9}, {0x10,0x08,0x08,0x18},  
, {0x6f,0xba,0xba,0xd5}, {0xf0,0x78,0x78,0x88}, {0x4a,0x25,0x25,0x6f}, {0x5c,0x2e,0x2e,0x72},  
, {0x38,0x1c,0x1c,0x24}, {0x57,0xa6,0xa6,0xf1}, {0x73,0xb4,0xb4,0xc7}, {0x97,0xc6,0xc6,0x51},  
, {0xcb,0xe8,0xe8,0x23}, {0xa1,0xdd,0xdd,0x7c}, {0xe8,0x74,0x74,0x9c}, {0x3e,0x1f,0x1f,0x21},  
, {0x96,0x4b,0x4b,0xdd}, {0x61,0xbd,0xbd,0xdc}, {0x0d,0x8b,0x8b,0x86}, {0x0f,0x8a,0x8a,0x85},  
, {0xe0,0x70,0x70,0x90}, {0x7c,0x3e,0x3e,0x42}, {0x71,0xb5,0xb5,0xc4}, {0xcc,0x66,0x66,0xaa},  
, {0x90,0x48,0x48,0xd8}, {0x06,0x03,0x03,0x05}, {0xf7,0xf6,0xf6,0x01}, {0x1c,0x0e,0x0e,0x12}

```
{0xc2,0x61,0x61,0xa3}, {0x6a,0x35,0x35,0x5f}, {0xae,0x57,0x57,0xf9}, {0x69,0xb9,0xb9,0xd0},
{0x17,0x86,0x86,0x91}, {0x99,0xc1,0xc1,0x58}, {0x3a,0x1d,0x1d,0x27}, {0x27,0x9e,0x9e,0xb9},
{0xd9,0xe1,0xe1,0x38}, {0xeb,0xf8,0xf8,0x13}, {0x2b,0x98,0x98,0xb3}, {0x22,0x11,0x11,0x33},
{0xd2,0x69,0x69,0xbb}, {0xa9,0xd9,0xd9,0x70}, {0x07,0x8e,0x8e,0x89}, {0x33,0x94,0x94,0xa7},
{0x2d,0x9b,0x9b,0xb6}, {0x3c,0x1e,0x1e,0x22}, {0x15,0x87,0x87,0x92}, {0xc9,0xe9,0xe9,0x20},
{0x87,0xce,0xce,0x49}, {0xaa,0x55,0x55,0xff}, {0x50,0x28,0x28,0x78}, {0xa5,0xdf,0xdf,0x7a},
{0x03,0x8c,0x8c,0x8f}, {0x59,0xa1,0xa1,0xf8}, {0x09,0x89,0x89,0x80}, {0x1a,0x0d,0x0d,0x17},
{0x65,0xbf,0xbf,0xda}, {0xd7,0xe6,0xe6,0x31}, {0x84,0x42,0x42,0xc6}, {0xd0,0x68,0x68,0xb8},
{0x82,0x41,0x41,0xc3}, {0x29,0x99,0x99,0xb0}, {0x5a,0x2d,0x2d,0x77}, {0x1e,0x0f,0x0f,0x11},
{0x7b,0xb0,0xb0,0xcb}, {0xa8,0x54,0x54,0xfc}, {0x6d,0xbb,0xbb,0xd6}, {0x2c,0x16,0x16,0x3a},
;
word8 T2[256][4] = {
{0xa5,0xc6,0x63,0x63}, {0x84,0xf8,0x7c,0x7c}, {0x99,0xee,0x77,0x77}, {0x8d,0xf6,0x7b,0x7b},
{0x0d,0xff,0xf2,0xf2}, {0xbd,0xd6,0x6b,0x6b}, {0xb1,0xde,0x6f,0x6f}, {0x54,0x91,0xc5,0xc5},
{0x50,0x60,0x30,0x30}, {0x03,0x02,0x01,0x01}, {0xa9,0xce,0x67,0x67}, {0x7d,0x56,0x2b,0x2b},
{0x19,0xe7,0xfe,0xfe}, {0x62,0xb5,0xd7,0xd7}, {0xe6,0x4d,0xab,0xab}, {0x9a,0xec,0x76,0x76},
{0x45,0x8f,0xca,0xca}, {0x9d,0x1f,0x82,0x82}, {0x40,0x89,0xc9,0xc9}, {0x87,0xfa,0x7d,0x7d},
{0x15,0xef,0xfa,0xfa}, {0xeb,0xb2,0x59,0x59}, {0xc9,0x8e,0x47,0x47}, {0x0b,0xfb,0xf0,0xf0},
{0xec,0x41,0xad,0xad}, {0x67,0xb3,0xd4,0xd4}, {0xfd,0x5f,0xa2,0xa2}, {0xea,0x45,0xaf,0xaf},
{0xbf,0x23,0x9c,0x9c}, {0xf7,0x53,0xa4,0xa4}, {0x96,0xe4,0x72,0x72}, {0x5b,0x9b,0xc0,0xc0},
{0xc2,0x75,0xb7,0xb7}, {0x1c,0xe1,0xfd,0xfd}, {0xae,0x3d,0x93,0x93}, {0x6a,0x4c,0x26,0x26},
{0x5a,0x6c,0x36,0x36}, {0x41,0x7e,0x3f,0x3f}, {0x02,0xf5,0xf7,0xf7}, {0x4f,0x83,0xcc,0xcc},
{0x5c,0x68,0x34,0x34}, {0xf4,0x51,0xa5,0xa5}, {0x34,0xd1,0xe5,0xe5}, {0x08,0xf9,0xf1,0xf1},
{0x93,0xe2,0x71,0x71}, {0x73,0xab,0xd8,0xd8}, {0x53,0x62,0x31,0x31}, {0x3f,0x2a,0x15,0x15},
{0x0c,0x08,0x04,0x04}, {0x52,0x95,0xc7,0xc7}, {0x65,0x46,0x23,0x23}, {0x5e,0x9d,0xc3,0xc3},
{0x28,0x30,0x18,0x18}, {0xa1,0x37,0x96,0x96}, {0x0f,0x0a,0x05,0x05}, {0xb5,0x2f,0x9a,0x9a},
{0x09,0x0e,0x07,0x07}, {0x36,0x24,0x12,0x12}, {0x9b,0x1b,0x80,0x80}, {0x3d,0xdf,0xe2,0xe2},
{0x26,0xcd,0xeb,0xeb}, {0x69,0x4e,0x27,0x27}, {0xcd,0x7f,0xb2,0xb2}, {0x9f,0xea,0x75,0x75},
{0x1b,0x12,0x09,0x09}, {0x9e,0x1d,0x83,0x83}, {0x74,0x58,0x2c,0x2c}, {0x2e,0x34,0x1a,0x1a},
{0x2d,0x36,0x1b,0x1b}, {0xb2,0xdc,0x6e,0x6e}, {0xee,0xb4,0x5a,0x5a}, {0xfb,0x5b,0xa0,0xa0},
{0xf6,0xa4,0x52,0x52}, {0x4d,0x76,0x3b,0x3b}, {0x61,0xb7,0xd6,0xd6}, {0xce,0x7d,0xb3,0xb3},
{0x7b,0x52,0x29,0x29}, {0x3e,0xdd,0xe3,0xe3}, {0x71,0x5e,0x2f,0x2f}, {0x97,0x13,0x84,0x84},
{0xf5,0xa6,0x53,0x53}, {0x68,0xb9,0xd1,0xd1}, {0x00,0x00,0x00,0x00}, {0x2c,0xc1,0xed,0xed},
{0x60,0x40,0x20,0x20}, {0x1f,0xe3,0xfc,0xfc}, {0xc8,0x79,0xb1,0xb1}, {0xed,0xb6,0x5b,0x5b},
{0xbe,0xd4,0x6a,0x6a}, {0x46,0x8d,0xcb,0xcb}, {0xd9,0x67,0xbe,0xbe}, {0x4b,0x72,0x39,0x39},
{0xde,0x94,0x4a,0x4a}, {0xd4,0x98,0x4c,0x4c}, {0xe8,0xb0,0x58,0x58}, {0x4a,0x85,0xcf,0xcf},
{0x6b,0xbb,0xd0,0xd0}, {0x2a,0xc5,0xef,0xef}, {0xe5,0x4f,0xaa,0xaa}, {0x16,0xed,0xfb,0xfb},
;
```

{0xc5,0x86,0x43,0x43}, {0xd7,0x9a,0x4d,0x4d}, {0x55,0x66,0x33,0x33}, {0x94,0x11,0x85,0x85},  
{0xcf,0x8a,0x45,0x45}, {0x10,0xe9,0xf9,0xf9}, {0x06,0x04,0x02,0x02}, {0x81,0xfe,0x7f,0x7f},  
{0xf0,0xa0,0x50,0x50}, {0x44,0x78,0x3c,0x3c}, {0xba,0x25,0x9f,0x9f}, {0xe3,0x4b,0xa8,0xa8},  
{0xf3,0xa2,0x51,0x51}, {0xfe,0x5d,0xa3,0xa3}, {0xc0,0x80,0x40,0x40}, {0x8a,0x05,0x8f,0x8f},  
{0xad,0x3f,0x92,0x92}, {0xbc,0x21,0x9d,0x9d}, {0x48,0x70,0x38,0x38}, {0x04,0xf1,0xf5,0xf5},  
{0xdf,0x63,0xbc,0xbc}, {0xc1,0x77,0xb6,0xb6}, {0x75,0xaf,0xda,0xda}, {0x63,0x42,0x21,0x21},  
{0x30,0x20,0x10,0x10}, {0x1a,0xe5,0xff,0xff}, {0x0e,0xfd,0xf3,0xf3}, {0x6d,0xbf,0xd2,0xd2},  
{0x4c,0x81,0xcd,0xcd}, {0x14,0x18,0x0c,0x0c}, {0x35,0x26,0x13,0x13}, {0x2f,0xc3,0xec,0xec},  
{0xe1,0xbe,0x5f,0x5f}, {0xa2,0x35,0x97,0x97}, {0xcc,0x88,0x44,0x44}, {0x39,0x2e,0x17,0x17},  
{0x57,0x93,0xc4,0xc4}, {0xf2,0x55,0xa7,0xa7}, {0x82,0xfc,0x7e,0x7e}, {0x47,0x7a,0x3d,0x3d},  
{0xac,0xc8,0x64,0x64}, {0xe7,0xba,0x5d,0x5d}, {0x2b,0x32,0x19,0x19}, {0x95,0xe6,0x73,0x73},  
{0xa0,0xc0,0x60,0x60}, {0x98,0x19,0x81,0x81}, {0xd1,0x9e,0x4f,0x4f}, {0x7f,0xa3,0xdc,0xdc},  
{0x66,0x44,0x22,0x22}, {0x7e,0x54,0x2a,0x2a}, {0xab,0x3b,0x90,0x90}, {0x83,0x0b,0x88,0x88},  
{0xca,0x8c,0x46,0x46}, {0x29,0xc7,0xee,0xee}, {0xd3,0x6b,0xb8,0xb8}, {0x3c,0x28,0x14,0x14},  
{0x79,0xa7,0xde,0xde}, {0xe2,0xbc,0x5e,0x5e}, {0x1d,0x16,0x0b,0x0b}, {0x76,0xad,0xdb,0xdb},  
{0x3b,0xdb,0xe0,0xe0}, {0x56,0x64,0x32,0x32}, {0x4e,0x74,0x3a,0x3a}, {0x1e,0x14,0x0a,0x0a},  
{0xdb,0x92,0x49,0x49}, {0x0a,0x0c,0x06,0x06}, {0x6c,0x48,0x24,0x24}, {0xe4,0xb8,0x5c,0x5c},  
{0x5d,0x9f,0xc2,0xc2}, {0x6e,0xbd,0xd3,0xd3}, {0xef,0x43,0xac,0xac}, {0xa6,0xc4,0x62,0x62},  
{0xa8,0x39,0x91,0x91}, {0xa4,0x31,0x95,0x95}, {0x37,0xd3,0xe4,0xe4}, {0x8b,0xf2,0x79,0x79},  
{0x32,0xd5,0xe7,0xe7}, {0x43,0x8b,0xc8,0xc8}, {0x59,0x6e,0x37,0x37}, {0xb7,0xda,0x6d,0x6d},  
{0x8c,0x01,0x8d,0x8d}, {0x64,0xb1,0xd5,0xd5}, {0xd2,0x9c,0x4e,0x4e}, {0xe0,0x49,0xa9,0xa9},  
{0xb4,0xd8,0x6c,0x6c}, {0xfa,0xac,0x56,0x56}, {0x07,0xf3,0xf4,0xf4}, {0x25,0xcf,0xea,0xea},  
{0xaf,0xca,0x65,0x65}, {0x8e,0xf4,0x7a,0x7a}, {0xe9,0x47,0xae,0xae}, {0x18,0x10,0x08,0x08},  
{0xd5,0x6f,0xba,0xba}, {0x88,0xf0,0x78,0x78}, {0x6f,0x4a,0x25,0x25}, {0x72,0x5c,0x2e,0x2e},  
{0x24,0x38,0x1c,0x1c}, {0xf1,0x57,0xa6,0xa6}, {0xc7,0x73,0xb4,0xb4}, {0x51,0x97,0xc6,0xc6},  
{0x23,0xcb,0xe8,0xe8}, {0x7c,0xa1,0xdd,0xdd}, {0x9c,0xe8,0x74,0x74}, {0x21,0x3e,0x1f,0x1f},  
{0xdd,0x96,0x4b,0x4b}, {0xdc,0x61,0xbd,0xbd}, {0x86,0x0d,0x8b,0x8b}, {0x85,0x0f,0x8a,0x8a},  
{0x90,0xe0,0x70,0x70}, {0x42,0x7c,0x3e,0x3e}, {0xc4,0x71,0xb5,0xb5}, {0xaa,0xcc,0x66,0x66},  
{0xd8,0x90,0x48,0x48}, {0x05,0x06,0x03,0x03}, {0x01,0xf7,0xf6,0xf6}, {0x12,0x1c,0x0e,0x0e},  
{0xa3,0xc2,0x61,0x61}, {0x5f,0x6a,0x35,0x35}, {0xf9,0xae,0x57,0x57}, {0xd0,0x69,0xb9,0xb9},  
{0x91,0x17,0x86,0x86}, {0x58,0x99,0xc1,0xc1}, {0x27,0x3a,0x1d,0x1d}, {0xb9,0x27,0x9e,0x9e},  
{0x38,0xd9,0xe1,0xe1}, {0x13,0xeb,0xf8,0xf8}, {0xb3,0x2b,0x98,0x98}, {0x33,0x22,0x11,0x11},  
{0xbb,0xd2,0x69,0x69}, {0x70,0xa9,0xd9,0xd9}, {0x89,0x07,0x8e,0x8e}, {0xa7,0x33,0x94,0x94},  
{0xb6,0x2d,0x9b,0x9b}, {0x22,0x3c,0x1e,0x1e}, {0x92,0x15,0x87,0x87}, {0x20,0xc9,0xe9,0xe9},  
{0x49,0x87,0xce,0xce}, {0xff,0xaa,0x55,0x55}, {0x78,0x50,0x28,0x28}, {0x7a,0xa5,0xdf,0xdf},  
{0x8f,0x03,0x8c,0x8c}, {0xf8,0x59,0xa1,0xa1}, {0x80,0x09,0x89,0x89}, {0x17,0x1a,0x0d,0x0d},  
{0xda,0x65,0xbf,0xbf}, {0x31,0xd7,0xe6,0xe6}, {0xc6,0x84,0x42,0x42}, {0xb8,0xd0,0x68,0x68}



```
{0xc3,0x82,0x41,0x41}, {0xb0,0x29,0x99,0x99}, {0x77,0x5a,0x2d,0x2d}, {0x11,0x1e,0x0f,0x0f}
{0xcb,0x7b,0xb0,0xb0}, {0xfc,0xa8,0x54,0x54}, {0xd6,0x6d,0xbb,0xbb}, {0x3a,0x2c,0x16,0x16}
;

word8 T3[256][4] = {
{0x63,0xa5,0xc6,0x63}, {0x7c,0x84,0xf8,0x7c}, {0x77,0x99,0xee,0x77}, {0x7b,0x8d,0xf6,0x7b}
{0xf2,0x0d,0xff,0xf2}, {0x6b,0xbd,0xd6,0x6b}, {0x6f,0xb1,0xde,0x6f}, {0xc5,0x54,0x91,0xc5}
{0x30,0x50,0x60,0x30}, {0x01,0x03,0x02,0x01}, {0x67,0xa9,0xce,0x67}, {0x2b,0x7d,0x56,0x2b}
{0xfe,0x19,0xe7,0xfe}, {0xd7,0x62,0xb5,0xd7}, {0xab,0xe6,0x4d,0xab}, {0x76,0x9a,0xec,0x76}
{0xca,0x45,0x8f,0xca}, {0x82,0x9d,0x1f,0x82}, {0xc9,0x40,0x89,0xc9}, {0x7d,0x87,0xfa,0x7d}
{0xfa,0x15,0xef,0xfa}, {0x59,0xeb,0xb2,0x59}, {0x47,0xc9,0x8e,0x47}, {0xf0,0x0b,0xfb,0xf0}
{0xad,0xec,0x41,0xad}, {0xd4,0x67,0xb3,0xd4}, {0xa2,0xfd,0x5f,0xa2}, {0xaf,0xea,0x45,0xaf}
{0x9c,0xbf,0x23,0x9c}, {0xa4,0xf7,0x53,0xa4}, {0x72,0x96,0xe4,0x72}, {0xc0,0x5b,0x9b,0xc0}
{0xb7,0xc2,0x75,0xb7}, {0xfd,0x1c,0xe1,0xfd}, {0x93,0xae,0x3d,0x93}, {0x26,0x6a,0x4c,0x26}
{0x36,0x5a,0x6c,0x36}, {0x3f,0x41,0x7e,0x3f}, {0xf7,0x02,0xf5,0xf7}, {0xcc,0x4f,0x83,0xcc}
{0x34,0x5c,0x68,0x34}, {0xa5,0xf4,0x51,0xa5}, {0xe5,0x34,0xd1,0xe5}, {0xf1,0x08,0xf9,0xf1}
{0x71,0x93,0xe2,0x71}, {0xd8,0x73,0xab,0xd8}, {0x31,0x53,0x62,0x31}, {0x15,0x3f,0x2a,0x15}
{0x04,0x0c,0x08,0x04}, {0xc7,0x52,0x95,0xc7}, {0x23,0x65,0x46,0x23}, {0xc3,0x5e,0x9d,0xc3}
{0x18,0x28,0x30,0x18}, {0x96,0xa1,0x37,0x96}, {0x05,0x0f,0x0a,0x05}, {0x9a,0xb5,0x2f,0x9a}
{0x07,0x09,0x0e,0x07}, {0x12,0x36,0x24,0x12}, {0x80,0x9b,0x1b,0x80}, {0xe2,0x3d,0xdf,0xe2}
{0xeb,0x26,0xcd,0xeb}, {0x27,0x69,0x4e,0x27}, {0xb2,0xcd,0x7f,0xb2}, {0x75,0x9f,0xea,0x75}
{0x09,0x1b,0x12,0x09}, {0x83,0x9e,0x1d,0x83}, {0x2c,0x74,0x58,0x2c}, {0x1a,0x2e,0x34,0x1a}
{0x1b,0x2d,0x36,0x1b}, {0x6e,0xb2,0xdc,0x6e}, {0x5a,0xee,0xb4,0x5a}, {0xa0,0xfb,0x5b,0xa0}
{0x52,0xf6,0xa4,0x52}, {0x3b,0x4d,0x76,0x3b}, {0xd6,0x61,0xb7,0xd6}, {0xb3,0xce,0x7d,0xb3}
{0x29,0x7b,0x52,0x29}, {0xe3,0x3e,0xdd,0xe3}, {0x2f,0x71,0x5e,0x2f}, {0x84,0x97,0x13,0x84}
{0x53,0xf5,0xa6,0x53}, {0xd1,0x68,0xb9,0xd1}, {0x00,0x00,0x00,0x00}, {0xed,0x2c,0xc1,0xed}
{0x20,0x60,0x40,0x20}, {0xfc,0x1f,0xe3,0xfc}, {0xb1,0xc8,0x79,0xb1}, {0x5b,0xed,0xb6,0x5b}
{0x6a,0xbe,0xd4,0x6a}, {0xcb,0x46,0x8d,0xcb}, {0xbe,0xd9,0x67,0xbe}, {0x39,0x4b,0x72,0x39}
{0x4a,0xde,0x94,0x4a}, {0x4c,0xd4,0x98,0x4c}, {0x58,0xe8,0xb0,0x58}, {0xcf,0x4a,0x85,0xcf}
{0xd0,0x6b,0xbb,0xd0}, {0xef,0x2a,0xc5,0xef}, {0xaa,0xe5,0x4f,0xaa}, {0xfb,0x16,0xed,0xfb}
{0x43,0xc5,0x86,0x43}, {0x4d,0xd7,0x9a,0x4d}, {0x33,0x55,0x66,0x33}, {0x85,0x94,0x11,0x85}
{0x45,0xcf,0x8a,0x45}, {0xf9,0x10,0xe9,0xf9}, {0x02,0x06,0x04,0x02}, {0x7f,0x81,0xfe,0x7f}
{0x50,0xf0,0xa0,0x50}, {0x3c,0x44,0x78,0x3c}, {0x9f,0xba,0x25,0x9f}, {0xa8,0xe3,0x4b,0xa8}
{0x51,0xf3,0xa2,0x51}, {0xa3,0xfe,0x5d,0xa3}, {0x40,0xc0,0x80,0x40}, {0x8f,0x8a,0x05,0x8f}
{0x92,0xad,0x3f,0x92}, {0x9d,0xbc,0x21,0x9d}, {0x38,0x48,0x70,0x38}, {0xf5,0x04,0xf1,0xf5}
{0xbc,0xdf,0x63,0xbc}, {0xb6,0xc1,0x77,0xb6}, {0xda,0x75,0xaf,0xda}, {0x21,0x63,0x42,0x21}
{0x10,0x30,0x20,0x10}, {0xff,0x1a,0xe5,0xff}, {0xf3,0x0e,0xfd,0xf3}, {0xd2,0x6d,0xbf,0xd2}
{0xcd,0x4c,0x81,0xcd}, {0x0c,0x14,0x18,0x0c}, {0x13,0x35,0x26,0x13}, {0xec,0x2f,0xc3,0xec}
{0x5f,0xe1,0xbe,0x5f}, {0x97,0xa2,0x35,0x97}, {0x44,0xcc,0x88,0x44}, {0x17,0x39,0x2e,0x17}
```

```
,
0xc4,0x57,0x93,0xc4}, {0xa7,0xf2,0x55,0xa7}, {0x7e,0x82,0xfc,0x7e}, {0x3d,0x47,0x7a,0x3d
0x64,0xac,0xc8,0x64}, {0x5d,0xe7,0xba,0x5d}, {0x19,0x2b,0x32,0x19}, {0x73,0x95,0xe6,0x73
0x60,0xa0,0xc0,0x60}, {0x81,0x98,0x19,0x81}, {0x4f,0xd1,0x9e,0x4f}, {0xdc,0x7f,0xa3,0xdc
0x22,0x66,0x44,0x22}, {0x2a,0x7e,0x54,0x2a}, {0x90,0xab,0x3b,0x90}, {0x88,0x83,0x0b,0x88
0x46,0xca,0x8c,0x46}, {0xee,0x29,0xc7,0xee}, {0xb8,0xd3,0x6b,0xb8}, {0x14,0x3c,0x28,0x14
0xde,0x79,0xa7,0xde}, {0x5e,0xe2,0xbc,0x5e}, {0x0b,0x1d,0x16,0x0b}, {0xdb,0x76,0xad,0xdb
0xe0,0x3b,0xdb,0xe0}, {0x32,0x56,0x64,0x32}, {0x3a,0x4e,0x74,0x3a}, {0x0a,0x1e,0x14,0x0a
0x49,0xdb,0x92,0x49}, {0x06,0x0a,0x0c,0x06}, {0x24,0x6c,0x48,0x24}, {0x5c,0xe4,0xb8,0x5c
0xc2,0x5d,0x9f,0xc2}, {0xd3,0x6e,0xbd,0xd3}, {0xac,0xef,0x43,0xac}, {0x62,0xa6,0xc4,0x62
0x91,0xa8,0x39,0x91}, {0x95,0xa4,0x31,0x95}, {0xe4,0x37,0xd3,0xe4}, {0x79,0x8b,0xf2,0x79
0xe7,0x32,0xd5,0xe7}, {0xc8,0x43,0x8b,0xc8}, {0x37,0x59,0x6e,0x37}, {0x6d,0xb7,0xda,0x6d
0x8d,0x8c,0x01,0x8d}, {0xd5,0x64,0xb1,0xd5}, {0x4e,0xd2,0x9c,0x4e}, {0xa9,0xe0,0x49,0xa9
0x6c,0xb4,0xd8,0x6c}, {0x56,0xfa,0xac,0x56}, {0xf4,0x07,0xf3,0xf4}, {0xea,0x25,0xcf,0xea
0x65,0xaf,0xca,0x65}, {0x7a,0x8e,0xf4,0x7a}, {0xae,0xe9,0x47,0xae}, {0x08,0x18,0x10,0x08
0xba,0xd5,0x6f,0xba}, {0x78,0x88,0xf0,0x78}, {0x25,0x6f,0x4a,0x25}, {0x2e,0x72,0x5c,0x2e
0x1c,0x24,0x38,0x1c}, {0xa6,0xf1,0x57,0xa6}, {0xb4,0xc7,0x73,0xb4}, {0xc6,0x51,0x97,0xc6
0xe8,0x23,0xcb,0xe8}, {0xdd,0x7c,0xa1,0xdd}, {0x74,0x9c,0xe8,0x74}, {0x1f,0x21,0x3e,0x1f
0x4b,0xdd,0x96,0x4b}, {0xbd,0xdc,0x61,0xbd}, {0x8b,0x86,0x0d,0x8b}, {0x8a,0x85,0x0f,0x8a
0x70,0x90,0xe0,0x70}, {0x3e,0x42,0x7c,0x3e}, {0xb5,0xc4,0x71,0xb5}, {0x66,0xaa,0xcc,0x66
0x48,0xd8,0x90,0x48}, {0x03,0x05,0x06,0x03}, {0xf6,0x01,0xf7,0xf6}, {0x0e,0x12,0x1c,0x0e
0x61,0xa3,0xc2,0x61}, {0x35,0x5f,0x6a,0x35}, {0x57,0xf9,0xae,0x57}, {0xb9,0xd0,0x69,0xb9
0x86,0x91,0x17,0x86}, {0xc1,0x58,0x99,0xc1}, {0x1d,0x27,0x3a,0x1d}, {0x9e,0xb9,0x27,0x9e
0xe1,0x38,0xd9,0xe1}, {0xf8,0x13,0xeb,0xf8}, {0x98,0xb3,0x2b,0x98}, {0x11,0x33,0x22,0x11
0x69,0xbb,0xd2,0x69}, {0xd9,0x70,0xa9,0xd9}, {0x8e,0x89,0x07,0x8e}, {0x94,0xa7,0x33,0x94
0x9b,0xb6,0x2d,0x9b}, {0x1e,0x22,0x3c,0x1e}, {0x87,0x92,0x15,0x87}, {0xe9,0x20,0xc9,0xe9
0xce,0x49,0x87,0xce}, {0x55,0xff,0xaa,0x55}, {0x28,0x78,0x50,0x28}, {0xdf,0x7a,0xa5,0xdf
0x8c,0x8f,0x03,0x8c}, {0xa1,0xf8,0x59,0xa1}, {0x89,0x80,0x09,0x89}, {0x0d,0x17,0x1a,0x0d
0xbf,0xda,0x65,0xbf}, {0xe6,0x31,0xd7,0xe6}, {0x42,0xc6,0x84,0x42}, {0x68,0xb8,0xd0,0x68
0x41,0xc3,0x82,0x41}, {0x99,0xb0,0x29,0x99}, {0x2d,0x77,0x5a,0x2d}, {0x0f,0x11,0x1e,0x0f
0xb0,0xcb,0x7b,0xb0}, {0x54,0xfc,0xa8,0x54}, {0xbb,0xd6,0x6d,0xbb}, {0x16,0x3a,0x2c,0x16
,
word8 T4[256][4] = {
0x63,0x63,0xa5,0xc6}, {0x7c,0x7c,0x84,0xf8}, {0x77,0x77,0x99,0xee}, {0x7b,0x7b,0x8d,0xf6
0xf2,0xf2,0x0d,0xff}, {0x6b,0x6b,0xbd,0xd6}, {0x6f,0x6f,0xb1,0xde}, {0xc5,0xc5,0x54,0x91
0x30,0x30,0x50,0x60}, {0x01,0x01,0x03,0x02}, {0x67,0x67,0xa9,0xce}, {0x2b,0x2b,0x7d,0x56
0xfe,0xfe,0x19,0xe7}, {0xd7,0xd7,0x62,0xb5}, {0xab,0xab,0xe6,0x4d}, {0x76,0x76,0x9a,0xec
0xca,0xca,0x45,0x8f}, {0x82,0x82,0x9d,0x1f}, {0xc9,0xc9,0x40,0x89}, {0x7d,0x7d,0x87,0xfa
,
```

{0xfa,0xfa,0x15,0xef}, {0x59,0x59,0xeb,0xb2}, {0x47,0x47,0xc9,0x8e}, {0xf0,0xf0,0x0b,0xfb},  
{0xad,0xad,0xec,0x41}, {0xd4,0xd4,0x67,0xb3}, {0xa2,0xa2,0xfd,0x5f}, {0xaf,0xaf,0xea,0x45},  
{0x9c,0x9c,0xbf,0x23}, {0xa4,0xa4,0xf7,0x53}, {0x72,0x72,0x96,0xe4}, {0xc0,0xc0,0x5b,0x9b},  
{0xb7,0xb7,0xc2,0x75}, {0xfd,0xfd,0x1c,0xe1}, {0x93,0x93,0xae,0x3d}, {0x26,0x26,0x6a,0x4c},  
{0x36,0x36,0x5a,0x6c}, {0x3f,0x3f,0x41,0x7e}, {0xf7,0xf7,0x02,0xf5}, {0xcc,0xcc,0x4f,0x83},  
{0x34,0x34,0x5c,0x68}, {0xa5,0xa5,0xf4,0x51}, {0xe5,0xe5,0x34,0xd1}, {0xf1,0xf1,0x08,0xf9},  
{0x71,0x71,0x93,0xe2}, {0xd8,0xd8,0x73,0xab}, {0x31,0x31,0x53,0x62}, {0x15,0x15,0x3f,0x2a},  
{0x04,0x04,0x0c,0x08}, {0xc7,0xc7,0x52,0x95}, {0x23,0x23,0x65,0x46}, {0xc3,0xc3,0x5e,0x9d},  
{0x18,0x18,0x28,0x30}, {0x96,0x96,0xa1,0x37}, {0x05,0x05,0x0f,0x0a}, {0x9a,0x9a,0xb5,0x2f},  
{0x07,0x07,0x09,0x0e}, {0x12,0x12,0x36,0x24}, {0x80,0x80,0x9b,0x1b}, {0xe2,0xe2,0x3d,0xdf},  
{0xeb,0xeb,0x26,0xcd}, {0x27,0x27,0x69,0x4e}, {0xb2,0xb2,0xcd,0x7f}, {0x75,0x75,0x9f,0xea},  
{0x09,0x09,0x1b,0x12}, {0x83,0x83,0x9e,0x1d}, {0x2c,0x2c,0x74,0x58}, {0x1a,0x1a,0x2e,0x34},  
{0x1b,0x1b,0x2d,0x36}, {0x6e,0x6e,0xb2,0xdc}, {0x5a,0x5a,0xee,0xb4}, {0xa0,0xa0,0xfb,0x5b},  
{0x52,0x52,0xf6,0xa4}, {0x3b,0x3b,0x4d,0x76}, {0xd6,0xd6,0x61,0xb7}, {0xb3,0xb3,0xce,0x7d},  
{0x29,0x29,0x7b,0x52}, {0xe3,0xe3,0x3e,0xdd}, {0x2f,0x2f,0x71,0x5e}, {0x84,0x84,0x97,0x13},  
{0x53,0x53,0xf5,0xa6}, {0xd1,0xd1,0x68,0xb9}, {0x00,0x00,0x00,0x00}, {0xed,0xed,0x2c,0xc1},  
{0x20,0x20,0x60,0x40}, {0xfc,0xfc,0x1f,0xe3}, {0xb1,0xb1,0xc8,0x79}, {0x5b,0x5b,0xed,0xb6},  
{0x6a,0x6a,0xbe,0xd4}, {0xcb,0xcb,0x46,0x8d}, {0xbe,0xbe,0xd9,0x67}, {0x39,0x39,0x4b,0x72},  
{0x4a,0x4a,0xde,0x94}, {0x4c,0x4c,0xd4,0x98}, {0x58,0x58,0xe8,0xb0}, {0xcf,0xcf,0x4a,0x85},  
{0xd0,0xd0,0x6b,0xbb}, {0xef,0xef,0x2a,0xc5}, {0xaa,0xaa,0xe5,0x4f}, {0xfb,0xfb,0x16,0xed},  
{0x43,0x43,0xc5,0x86}, {0x4d,0x4d,0xd7,0x9a}, {0x33,0x33,0x55,0x66}, {0x85,0x85,0x94,0x11},  
{0x45,0x45,0xcf,0x8a}, {0xf9,0xf9,0x10,0xe9}, {0x02,0x02,0x06,0x04}, {0x7f,0x7f,0x81,0xfe},  
{0x50,0x50,0xf0,0xa0}, {0x3c,0x3c,0x44,0x78}, {0x9f,0x9f,0xba,0x25}, {0xa8,0xa8,0xe3,0x4b},  
{0x51,0x51,0xf3,0xa2}, {0xa3,0xa3,0xfe,0x5d}, {0x40,0x40,0xc0,0x80}, {0x8f,0x8f,0x8a,0x05},  
{0x92,0x92,0xad,0x3f}, {0x9d,0x9d,0xbc,0x21}, {0x38,0x38,0x48,0x70}, {0xf5,0xf5,0x04,0xf1},  
{0xbc,0xbc,0xdf,0x63}, {0xb6,0xb6,0xc1,0x77}, {0xda,0xda,0x75,0xaf}, {0x21,0x21,0x63,0x42},  
{0x10,0x10,0x30,0x20}, {0xff,0xff,0x1a,0xe5}, {0xf3,0xf3,0x0e,0xfd}, {0xd2,0xd2,0x6d,0xbf},  
{0xcd,0xcd,0x4c,0x81}, {0x0c,0x0c,0x14,0x18}, {0x13,0x13,0x35,0x26}, {0xec,0xec,0x2f,0xc3},  
{0x5f,0x5f,0xe1,0xbe}, {0x97,0x97,0xa2,0x35}, {0x44,0x44,0xcc,0x88}, {0x17,0x17,0x39,0x2e},  
{0xc4,0xc4,0x57,0x93}, {0xa7,0xa7,0xf2,0x55}, {0x7e,0x7e,0x82,0xfc}, {0x3d,0x3d,0x47,0x7a},  
{0x64,0x64,0xac,0xc8}, {0x5d,0x5d,0xe7,0xba}, {0x19,0x19,0x2b,0x32}, {0x73,0x73,0x95,0xe6},  
{0x60,0x60,0xa0,0xc0}, {0x81,0x81,0x98,0x19}, {0x4f,0x4f,0xd1,0x9e}, {0xdc,0xdc,0x7f,0xa3},  
{0x22,0x22,0x66,0x44}, {0x2a,0x2a,0x7e,0x54}, {0x90,0x90,0xab,0x3b}, {0x88,0x88,0x83,0x0b},  
{0x46,0x46,0xca,0x8c}, {0xee,0xee,0x29,0xc7}, {0xb8,0xb8,0xd3,0x6b}, {0x14,0x14,0x3c,0x28},  
{0xde,0xde,0x79,0xa7}, {0x5e,0x5e,0xe2,0xbc}, {0x0b,0x0b,0x1d,0x16}, {0xdb,0xdb,0x76,0xad},  
{0xe0,0xe0,0x3b,0xdb}, {0x32,0x32,0x56,0x64}, {0x3a,0x3a,0x4e,0x74}, {0x0a,0x0a,0x1e,0x14},  
{0x49,0x49,0xdb,0x92}, {0x06,0x06,0x0a,0x0c}, {0x24,0x24,0x6c,0x48}, {0x5c,0x5c,0xe4,0xb8}

```
{0xc2,0xc2,0x5d,0x9f}, {0xd3,0xd3,0x6e,0xbd}, {0xac,0xac,0xef,0x43}, {0x62,0x62,0xa6,0xc4}
{0x91,0x91,0xa8,0x39}, {0x95,0x95,0xa4,0x31}, {0xe4,0xe4,0x37,0xd3}, {0x79,0x79,0x8b,0xf2}
{0xe7,0xe7,0x32,0xd5}, {0xc8,0xc8,0x43,0x8b}, {0x37,0x37,0x59,0x6e}, {0x6d,0x6d,0xb7,0xda}
{0x8d,0x8d,0x8c,0x01}, {0xd5,0xd5,0x64,0xb1}, {0x4e,0x4e,0xd2,0x9c}, {0xa9,0xa9,0xe0,0x49}
{0x6c,0x6c,0xb4,0xd8}, {0x56,0x56,0xfa,0xac}, {0xf4,0xf4,0x07,0xf3}, {0xea,0xea,0x25,0xcf}
{0x65,0x65,0xaf,0xca}, {0x7a,0x7a,0x8e,0xf4}, {0xae,0xae,0xe9,0x47}, {0x08,0x08,0x18,0x10}
{0xba,0xba,0xd5,0x6f}, {0x78,0x78,0x88,0xf0}, {0x25,0x25,0x6f,0x4a}, {0x2e,0x2e,0x72,0x5c}
{0x1c,0x1c,0x24,0x38}, {0xa6,0xa6,0xf1,0x57}, {0xb4,0xb4,0xc7,0x73}, {0xc6,0xc6,0x51,0x97}
{0xe8,0xe8,0x23,0xcb}, {0xdd,0xdd,0x7c,0xa1}, {0x74,0x74,0x9c,0xe8}, {0x1f,0x1f,0x21,0x3e}
{0x4b,0x4b,0xdd,0x96}, {0xbd,0xbd,0xdc,0x61}, {0x8b,0x8b,0x86,0x0d}, {0x8a,0x8a,0x85,0x0f}
{0x70,0x70,0x90,0xe0}, {0x3e,0x3e,0x42,0x7c}, {0xb5,0xb5,0xc4,0x71}, {0x66,0x66,0xaa,0xcc}
{0x48,0x48,0xd8,0x90}, {0x03,0x03,0x05,0x06}, {0xf6,0xf6,0x01,0xf7}, {0x0e,0x0e,0x12,0x1c}
{0x61,0x61,0xa3,0xc2}, {0x35,0x35,0x5f,0x6a}, {0x57,0x57,0xf9,0xae}, {0xb9,0xb9,0xd0,0x69}
{0x86,0x86,0x91,0x17}, {0xc1,0xc1,0x58,0x99}, {0x1d,0x1d,0x27,0x3a}, {0x9e,0x9e,0xb9,0x27}
{0xe1,0xe1,0x38,0xd9}, {0xf8,0xf8,0x13,0xeb}, {0x98,0x98,0xb3,0x2b}, {0x11,0x11,0x33,0x22}
{0x69,0x69,0xbb,0xd2}, {0xd9,0xd9,0x70,0xa9}, {0x8e,0x8e,0x89,0x07}, {0x94,0x94,0xa7,0x33}
{0x9b,0x9b,0xb6,0x2d}, {0x1e,0x1e,0x22,0x3c}, {0x87,0x87,0x92,0x15}, {0xe9,0xe9,0x20,0xc9}
{0xce,0xce,0x49,0x87}, {0x55,0x55,0xff,0xaa}, {0x28,0x28,0x78,0x50}, {0xdf,0xdf,0x7a,0xa5}
{0x8c,0x8c,0x8f,0x03}, {0xa1,0xa1,0xf8,0x59}, {0x89,0x89,0x80,0x09}, {0x0d,0x0d,0x17,0x1a}
{0xbf,0xbf,0xda,0x65}, {0xe6,0xe6,0x31,0xd7}, {0x42,0x42,0xc6,0x84}, {0x68,0x68,0xb8,0xd0}
{0x41,0x41,0xc3,0x82}, {0x99,0x99,0xb0,0x29}, {0x2d,0x2d,0x77,0x5a}, {0x0f,0x0f,0x11,0x1e}
{0xb0,0xb0,0xcb,0x7b}, {0x54,0x54,0xfc,0xa8}, {0xbb,0xbb,0xd6,0x6d}, {0x16,0x16,0x3a,0x2c}
;
```

```
word8 T5[256][4] = {
{0x51,0xf4,0xa7,0x50}, {0x7e,0x41,0x65,0x53}, {0x1a,0x17,0xa4,0xc3}, {0x3a,0x27,0x5e,0x96}
{0x3b,0xab,0x6b,0xcb}, {0x1f,0x9d,0x45,0xf1}, {0xac,0xfa,0x58,0xab}, {0x4b,0xe3,0x03,0x93}
{0x20,0x30,0xfa,0x55}, {0xad,0x76,0x6d,0xf6}, {0x88,0xcc,0x76,0x91}, {0xf5,0x02,0x4c,0x25}
{0x4f,0xe5,0xd7,0xfc}, {0xc5,0x2a,0xcb,0xd7}, {0x26,0x35,0x44,0x80}, {0xb5,0x62,0xa3,0x8f}
{0xde,0xb1,0x5a,0x49}, {0x25,0xba,0x1b,0x67}, {0x45,0xea,0x0e,0x98}, {0x5d,0xfe,0xc0,0xe1}
{0xc3,0x2f,0x75,0x02}, {0x81,0x4c,0xf0,0x12}, {0x8d,0x46,0x97,0xa3}, {0x6b,0xd3,0xf9,0xc6}
{0x03,0x8f,0x5f,0xe7}, {0x15,0x92,0x9c,0x95}, {0xbf,0x6d,0x7a,0xeb}, {0x95,0x52,0x59,0xda}
{0xd4,0xbe,0x83,0x2d}, {0x58,0x74,0x21,0xd3}, {0x49,0xe0,0x69,0x29}, {0x8e,0xc9,0xc8,0x44}
{0x75,0xc2,0x89,0x6a}, {0xf4,0x8e,0x79,0x78}, {0x99,0x58,0x3e,0x6b}, {0x27,0xb9,0x71,0xdd}
{0xbe,0xe1,0x4f,0xb6}, {0xf0,0x88,0xad,0x17}, {0xc9,0x20,0xac,0x66}, {0x7d,0xce,0x3a,0xb4}
{0x63,0xdf,0x4a,0x18}, {0xe5,0x1a,0x31,0x82}, {0x97,0x51,0x33,0x60}, {0x62,0x53,0x7f,0x45}
{0xb1,0x64,0x77,0xe0}, {0xbb,0x6b,0xae,0x84}, {0xfe,0x81,0xa0,0x1c}, {0xf9,0x08,0x2b,0x94}
{0x70,0x48,0x68,0x58}, {0x8f,0x45,0xfd,0x19}, {0x94,0xde,0x6c,0x87}, {0x52,0x7b,0xf8,0xb7}
{0xab,0x73,0xd3,0x23}, {0x72,0x4b,0x02,0xe2}, {0xe3,0x1f,0x8f,0x57}, {0x66,0x55,0xab,0x2a}
```

```
{0xb2,0xeb,0x28,0x07}, {0x2f,0xb5,0xc2,0x03}, {0x86,0xc5,0x7b,0x9a}, {0xd3,0x37,0x08,0xa5}
{0x30,0x28,0x87,0xf2}, {0x23,0xbf,0xa5,0xb2}, {0x02,0x03,0x6a,0xba}, {0xed,0x16,0x82,0x5c}
{0x8a,0xcf,0x1c,0x2b}, {0xa7,0x79,0xb4,0x92}, {0xf3,0x07,0xf2,0xf0}, {0x4e,0x69,0xe2,0xa1}
{0x65,0xda,0xf4,0xcd}, {0x06,0x05,0xbe,0xd5}, {0xd1,0x34,0x62,0x1f}, {0xc4,0xa6,0xfe,0x8a}
{0x34,0x2e,0x53,0x9d}, {0xa2,0xf3,0x55,0xa0}, {0x05,0x8a,0xe1,0x32}, {0xa4,0xf6,0xeb,0x75}
{0x0b,0x83,0xec,0x39}, {0x40,0x60,0xef,0xaa}, {0x5e,0x71,0x9f,0x06}, {0xbd,0x6e,0x10,0x51}
{0x3e,0x21,0x8a,0xf9}, {0x96,0xdd,0x06,0x3d}, {0xdd,0x3e,0x05,0xae}, {0x4d,0xe6,0xbd,0x46}
{0x91,0x54,0x8d,0xb5}, {0x71,0xc4,0x5d,0x05}, {0x04,0x06,0xd4,0x6f}, {0x60,0x50,0x15,0xff}
{0x19,0x98,0xfb,0x24}, {0xd6,0xbd,0xe9,0x97}, {0x89,0x40,0x43,0xcc}, {0x67,0xd9,0x9e,0x77}
{0xb0,0xe8,0x42,0xbd}, {0x07,0x89,0x8b,0x88}, {0xe7,0x19,0x5b,0x38}, {0x79,0xc8,0xee,0xdb}
{0xa1,0x7c,0x0a,0x47}, {0x7c,0x42,0x0f,0xe9}, {0xf8,0x84,0x1e,0xc9}, {0x00,0x00,0x00,0x00}
{0x09,0x80,0x86,0x83}, {0x32,0x2b,0xed,0x48}, {0x1e,0x11,0x70,0xac}, {0x6c,0x5a,0x72,0x4e}
{0xfd,0x0e,0xff,0xfb}, {0x0f,0x85,0x38,0x56}, {0x3d,0xae,0xd5,0x1e}, {0x36,0x2d,0x39,0x27}
{0x0a,0x0f,0xd9,0x64}, {0x68,0x5c,0xa6,0x21}, {0x9b,0x5b,0x54,0xd1}, {0x24,0x36,0x2e,0x3a}
{0x0c,0x0a,0x67,0xb1}, {0x93,0x57,0xe7,0x0f}, {0xb4,0xee,0x96,0xd2}, {0x1b,0x9b,0x91,0x9e}
{0x80,0xc0,0xc5,0x4f}, {0x61,0xdc,0x20,0xa2}, {0x5a,0x77,0x4b,0x69}, {0x1c,0x12,0x1a,0x16}
{0xe2,0x93,0xba,0x0a}, {0xc0,0xa0,0x2a,0xe5}, {0x3c,0x22,0xe0,0x43}, {0x12,0x1b,0x17,0x1d}
{0x0e,0x09,0x0d,0x0b}, {0xf2,0x8b,0xc7,0xad}, {0x2d,0xb6,0xa8,0xb9}, {0x14,0x1e,0xa9,0xc8}
{0x57,0xf1,0x19,0x85}, {0xaf,0x75,0x07,0x4c}, {0xee,0x99,0xdd,0xbb}, {0xa3,0x7f,0x60,0xfd}
{0xf7,0x01,0x26,0x9f}, {0x5c,0x72,0xf5,0xbc}, {0x44,0x66,0x3b,0xc5}, {0x5b,0xfb,0x7e,0x34}
{0x8b,0x43,0x29,0x76}, {0xcb,0x23,0xc6,0xdc}, {0xb6,0xed,0xfc,0x68}, {0xb8,0xe4,0xf1,0x63}
{0xd7,0x31,0xdc,0xca}, {0x42,0x63,0x85,0x10}, {0x13,0x97,0x22,0x40}, {0x84,0xc6,0x11,0x20}
{0x85,0x4a,0x24,0x7d}, {0xd2,0xbb,0x3d,0xf8}, {0xae,0xf9,0x32,0x11}, {0xc7,0x29,0xa1,0x6d}
{0x1d,0x9e,0x2f,0x4b}, {0xdc,0xb2,0x30,0xf3}, {0x0d,0x86,0x52,0xec}, {0x77,0xc1,0xe3,0xd0}
{0x2b,0xb3,0x16,0x6c}, {0xa9,0x70,0xb9,0x99}, {0x11,0x94,0x48,0xfa}, {0x47,0xe9,0x64,0x22}
{0xa8,0xfc,0x8c,0xc4}, {0xa0,0xf0,0x3f,0x1a}, {0x56,0x7d,0x2c,0xd8}, {0x22,0x33,0x90,0xef}
{0x87,0x49,0x4e,0xc7}, {0xd9,0x38,0xd1,0xc1}, {0x8c,0xca,0xa2,0xfe}, {0x98,0xd4,0x0b,0x36}
{0xa6,0xf5,0x81,0xcf}, {0xa5,0x7a,0xde,0x28}, {0xda,0xb7,0x8e,0x26}, {0x3f,0xad,0xbf,0xa4}
{0x2c,0x3a,0x9d,0xe4}, {0x50,0x78,0x92,0x0d}, {0x6a,0x5f,0xcc,0x9b}, {0x54,0x7e,0x46,0x62}
{0xf6,0x8d,0x13,0xc2}, {0x90,0xd8,0xb8,0xe8}, {0x2e,0x39,0xf7,0x5e}, {0x82,0xc3,0xaf,0xf5}
{0x9f,0x5d,0x80,0xbe}, {0x69,0xd0,0x93,0x7c}, {0x6f,0xd5,0x2d,0xa9}, {0xcf,0x25,0x12,0xb3}
{0xc8,0xac,0x99,0x3b}, {0x10,0x18,0x7d,0xa7}, {0xe8,0x9c,0x63,0x6e}, {0xdb,0x3b,0xbb,0x7b}
{0xcd,0x26,0x78,0x09}, {0x6e,0x59,0x18,0xf4}, {0xec,0x9a,0xb7,0x01}, {0x83,0x4f,0x9a,0xa8}
{0xe6,0x95,0x6e,0x65}, {0xaa,0xff,0xe6,0x7e}, {0x21,0xbc,0xcf,0x08}, {0xef,0x15,0xe8,0xe6}
{0xba,0xe7,0x9b,0xd9}, {0x4a,0x6f,0x36,0xce}, {0xea,0x9f,0x09,0xd4}, {0x29,0xb0,0x7c,0xd6}
{0x31,0xa4,0xb2,0xaf}, {0x2a,0x3f,0x23,0x31}, {0xc6,0xa5,0x94,0x30}, {0x35,0xa2,0x66,0xc0}
{0x74,0x4e,0xbc,0x37}, {0xfc,0x82,0xca,0xa6}, {0xe0,0x90,0xd0,0xb0}, {0x33,0xa7,0xd8,0x15}
```

```
{0xf1,0x04,0x98,0x4a}, {0x41,0xec,0xda,0xf7}, {0x7f,0xcd,0x50,0x0e}, {0x17,0x91,0xf6,0x2f},
{0x76,0x4d,0xd6,0x8d}, {0x43,0xef,0xb0,0x4d}, {0xcc,0xaa,0x4d,0x54}, {0xe4,0x96,0x04,0xdf},
{0x9e,0xd1,0xb5,0xe3}, {0x4c,0x6a,0x88,0x1b}, {0xc1,0x2c,0x1f,0xb8}, {0x46,0x65,0x51,0x7f},
{0x9d,0x5e,0xea,0x04}, {0x01,0x8c,0x35,0x5d}, {0xfa,0x87,0x74,0x73}, {0xfb,0x0b,0x41,0x2e},
{0xb3,0x67,0x1d,0x5a}, {0x92,0xdb,0xd2,0x52}, {0xe9,0x10,0x56,0x33}, {0x6d,0xd6,0x47,0x13},
{0x9a,0xd7,0x61,0x8c}, {0x37,0xa1,0x0c,0x7a}, {0x59,0xf8,0x14,0x8e}, {0xeb,0x13,0x3c,0x89},
{0xce,0xa9,0x27,0xee}, {0xb7,0x61,0xc9,0x35}, {0xe1,0x1c,0xe5,0xed}, {0x7a,0x47,0xb1,0x3c},
{0x9c,0xd2,0xdf,0x59}, {0x55,0xf2,0x73,0x3f}, {0x18,0x14,0xce,0x79}, {0x73,0xc7,0x37,0xbf},
{0x53,0xf7,0xcd,0xea}, {0x5f,0xfd,0xaa,0x5b}, {0xdf,0x3d,0x6f,0x14}, {0x78,0x44,0xdb,0x86},
{0xca,0xaf,0xf3,0x81}, {0xb9,0x68,0xc4,0x3e}, {0x38,0x24,0x34,0x2c}, {0xc2,0xa3,0x40,0x5f},
{0x16,0x1d,0xc3,0x72}, {0xbc,0xe2,0x25,0x0c}, {0x28,0x3c,0x49,0x8b}, {0xff,0x0d,0x95,0x41},
{0x39,0xa8,0x01,0x71}, {0x08,0x0c,0xb3,0xde}, {0xd8,0xb4,0xe4,0x9c}, {0x64,0x56,0xc1,0x90},
{0x7b,0xcb,0x84,0x61}, {0xd5,0x32,0xb6,0x70}, {0x48,0x6c,0x5c,0x74}, {0xd0,0xb8,0x57,0x42},
;
;
word8 T6[256][4] = {
{0x50,0x51,0xf4,0xa7}, {0x53,0x7e,0x41,0x65}, {0xc3,0x1a,0x17,0xa4}, {0x96,0x3a,0x27,0x5e},
{0xcb,0x3b,0xab,0x6b}, {0xf1,0x1f,0x9d,0x45}, {0xab,0xac,0xfa,0x58}, {0x93,0x4b,0xe3,0x03},
{0x55,0x20,0x30,0xfa}, {0xf6,0xad,0x76,0x6d}, {0x91,0x88,0xcc,0x76}, {0x25,0xf5,0x02,0x4c},
{0xfc,0x4f,0xe5,0xd7}, {0xd7,0xc5,0x2a,0xcb}, {0x80,0x26,0x35,0x44}, {0x8f,0xb5,0x62,0xa3},
{0x49,0xde,0xb1,0x5a}, {0x67,0x25,0xba,0x1b}, {0x98,0x45,0xea,0x0e}, {0xe1,0x5d,0xfe,0xc0},
{0x02,0xc3,0x2f,0x75}, {0x12,0x81,0x4c,0xf0}, {0xa3,0x8d,0x46,0x97}, {0xc6,0x6b,0xd3,0xf9},
{0xe7,0x03,0x8f,0x5f}, {0x95,0x15,0x92,0x9c}, {0xeb,0xbf,0x6d,0x7a}, {0xda,0x95,0x52,0x59},
{0x2d,0xd4,0xbe,0x83}, {0xd3,0x58,0x74,0x21}, {0x29,0x49,0xe0,0x69}, {0x44,0x8e,0xc9,0xc8},
{0x6a,0x75,0xc2,0x89}, {0x78,0xf4,0x8e,0x79}, {0x6b,0x99,0x58,0x3e}, {0xdd,0x27,0xb9,0x71},
{0xb6,0xbe,0xe1,0x4f}, {0x17,0xf0,0x88,0xad}, {0x66,0xc9,0x20,0xac}, {0xb4,0x7d,0xce,0x3a},
{0x18,0x63,0xdf,0x4a}, {0x82,0xe5,0x1a,0x31}, {0x60,0x97,0x51,0x33}, {0x45,0x62,0x53,0x7f},
{0xe0,0xb1,0x64,0x77}, {0x84,0xbb,0x6b,0xae}, {0x1c,0xfe,0x81,0xa0}, {0x94,0xf9,0x08,0x2b},
{0x58,0x70,0x48,0x68}, {0x19,0x8f,0x45,0xfd}, {0x87,0x94,0xde,0x6c}, {0xb7,0x52,0x7b,0xf8},
{0x23,0xab,0x73,0xd3}, {0xe2,0x72,0x4b,0x02}, {0x57,0xe3,0x1f,0x8f}, {0x2a,0x66,0x55,0xab},
{0x07,0xb2,0xeb,0x28}, {0x03,0x2f,0xb5,0xc2}, {0x9a,0x86,0xc5,0x7b}, {0xa5,0xd3,0x37,0x08},
{0xf2,0x30,0x28,0x87}, {0xb2,0x23,0xbf,0xa5}, {0xba,0x02,0x03,0x6a}, {0x5c,0xed,0x16,0x82},
{0x2b,0x8a,0xcf,0x1c}, {0x92,0xa7,0x79,0xb4}, {0xf0,0xf3,0x07,0xf2}, {0xa1,0x4e,0x69,0xe2},
{0xcd,0x65,0xda,0xf4}, {0xd5,0x06,0x05,0xbe}, {0x1f,0xd1,0x34,0x62}, {0x8a,0xc4,0xa6,0xfe},
{0x9d,0x34,0x2e,0x53}, {0xa0,0xa2,0xf3,0x55}, {0x32,0x05,0x8a,0xe1}, {0x75,0xa4,0xf6,0xeb},
{0x39,0x0b,0x83,0xec}, {0xaa,0x40,0x60,0xef}, {0x06,0x5e,0x71,0x9f}, {0x51,0xbd,0x6e,0x10},
{0xf9,0x3e,0x21,0x8a}, {0x3d,0x96,0xdd,0x06}, {0xae,0xdd,0x3e,0x05}, {0x46,0x4d,0xe6,0xbd},
{0xb5,0x91,0x54,0x8d}, {0x05,0x71,0xc4,0x5d}, {0x6f,0x04,0x06,0xd4}, {0xff,0x60,0x50,0x15},
;
```

{0x24,0x19,0x98,0xfb}, {0x97,0xd6,0xbd,0xe9}, {0xcc,0x89,0x40,0x43}, {0x77,0x67,0xd9,0x9e},  
{0xbd,0xb0,0xe8,0x42}, {0x88,0x07,0x89,0x8b}, {0x38,0xe7,0x19,0x5b}, {0xdb,0x79,0xc8,0xee},  
{0x47,0xa1,0x7c,0x0a}, {0xe9,0x7c,0x42,0x0f}, {0xc9,0xf8,0x84,0x1e}, {0x00,0x00,0x00,0x00},  
{0x83,0x09,0x80,0x86}, {0x48,0x32,0x2b,0xed}, {0xac,0x1e,0x11,0x70}, {0x4e,0x6c,0x5a,0x72},  
{0xfb,0xfd,0x0e,0xff}, {0x56,0x0f,0x85,0x38}, {0x1e,0x3d,0xae,0xd5}, {0x27,0x36,0x2d,0x39},  
{0x64,0x0a,0x0f,0xd9}, {0x21,0x68,0x5c,0xa6}, {0xd1,0x9b,0x5b,0x54}, {0x3a,0x24,0x36,0x2e},  
{0xb1,0x0c,0x0a,0x67}, {0x0f,0x93,0x57,0xe7}, {0xd2,0xb4,0xee,0x96}, {0x9e,0x1b,0x9b,0x91},  
{0x4f,0x80,0xc0,0xc5}, {0xa2,0x61,0xdc,0x20}, {0x69,0x5a,0x77,0x4b}, {0x16,0x1c,0x12,0x1a},  
{0x0a,0xe2,0x93,0xba}, {0xe5,0xc0,0xa0,0x2a}, {0x43,0x3c,0x22,0xe0}, {0x1d,0x12,0x1b,0x17},  
{0x0b,0x0e,0x09,0x0d}, {0xad,0xf2,0x8b,0xc7}, {0xb9,0x2d,0xb6,0xa8}, {0xc8,0x14,0x1e,0xa9},  
{0x85,0x57,0xf1,0x19}, {0x4c,0xaf,0x75,0x07}, {0xbb,0xee,0x99,0xdd}, {0xfd,0xa3,0x7f,0x60},  
{0x9f,0xf7,0x01,0x26}, {0xbc,0x5c,0x72,0xf5}, {0xc5,0x44,0x66,0x3b}, {0x34,0x5b,0xfb,0x7e},  
{0x76,0x8b,0x43,0x29}, {0xdc,0xcb,0x23,0xc6}, {0x68,0xb6,0xed,0xfc}, {0x63,0xb8,0xe4,0xf1},  
{0xca,0xd7,0x31,0xdc}, {0x10,0x42,0x63,0x85}, {0x40,0x13,0x97,0x22}, {0x20,0x84,0xc6,0x11},  
{0x7d,0x85,0x4a,0x24}, {0xf8,0xd2,0xbb,0x3d}, {0x11,0xae,0xf9,0x32}, {0x6d,0xc7,0x29,0xa1},  
{0x4b,0x1d,0x9e,0x2f}, {0xf3,0xdc,0xb2,0x30}, {0xec,0x0d,0x86,0x52}, {0xd0,0x77,0xc1,0xe3},  
{0x6c,0x2b,0xb3,0x16}, {0x99,0xa9,0x70,0xb9}, {0xfa,0x11,0x94,0x48}, {0x22,0x47,0xe9,0x64},  
{0xc4,0xa8,0xfc,0x8c}, {0x1a,0xa0,0xf0,0x3f}, {0xd8,0x56,0x7d,0x2c}, {0xef,0x22,0x33,0x90},  
{0xc7,0x87,0x49,0x4e}, {0xc1,0xd9,0x38,0xd1}, {0xfe,0x8c,0xca,0xa2}, {0x36,0x98,0xd4,0x0b},  
{0xcf,0xa6,0xf5,0x81}, {0x28,0xa5,0x7a,0xde}, {0x26,0xda,0xb7,0x8e}, {0xa4,0x3f,0xad,0xbf},  
{0xe4,0x2c,0x3a,0x9d}, {0x0d,0x50,0x78,0x92}, {0x9b,0x6a,0x5f,0xcc}, {0x62,0x54,0x7e,0x46},  
{0xc2,0xf6,0x8d,0x13}, {0xe8,0x90,0xd8,0xb8}, {0x5e,0x2e,0x39,0xf7}, {0xf5,0x82,0xc3,0xaf},  
{0xbe,0x9f,0x5d,0x80}, {0x7c,0x69,0xd0,0x93}, {0xa9,0x6f,0xd5,0x2d}, {0xb3,0xcf,0x25,0x12},  
{0x3b,0xc8,0xac,0x99}, {0xa7,0x10,0x18,0x7d}, {0x6e,0xe8,0x9c,0x63}, {0x7b,0xdb,0x3b,0xbb},  
{0x09,0xcd,0x26,0x78}, {0xf4,0x6e,0x59,0x18}, {0x01,0xec,0x9a,0xb7}, {0xa8,0x83,0x4f,0x9a},  
{0x65,0xe6,0x95,0x6e}, {0x7e,0xaa,0xff,0xe6}, {0x08,0x21,0xbc,0xcf}, {0xe6,0xef,0x15,0xe8},  
{0xd9,0xba,0xe7,0x9b}, {0xce,0x4a,0x6f,0x36}, {0xd4,0xea,0x9f,0x09}, {0xd6,0x29,0xb0,0x7c},  
{0xaf,0x31,0xa4,0xb2}, {0x31,0x2a,0x3f,0x23}, {0x30,0xc6,0xa5,0x94}, {0xc0,0x35,0xa2,0x66},  
{0x37,0x74,0x4e,0xbc}, {0xa6,0xfc,0x82,0xca}, {0xb0,0xe0,0x90,0xd0}, {0x15,0x33,0xa7,0xd8},  
{0x4a,0xf1,0x04,0x98}, {0xf7,0x41,0xec,0xda}, {0x0e,0x7f,0xcd,0x50}, {0x2f,0x17,0x91,0xf6},  
{0x8d,0x76,0x4d,0xd6}, {0x4d,0x43,0xef,0xb0}, {0x54,0xcc,0xaa,0x4d}, {0xdf,0xe4,0x96,0x04},  
{0xe3,0x9e,0xd1,0xb5}, {0x1b,0x4c,0x6a,0x88}, {0xb8,0xc1,0x2c,0x1f}, {0x7f,0x46,0x65,0x51},  
{0x04,0x9d,0x5e,0xea}, {0x5d,0x01,0x8c,0x35}, {0x73,0xfa,0x87,0x74}, {0x2e,0xfb,0x0b,0x41},  
{0x5a,0xb3,0x67,0x1d}, {0x52,0x92,0xdb,0xd2}, {0x33,0xe9,0x10,0x56}, {0x13,0x6d,0xd6,0x47},  
{0x8c,0x9a,0xd7,0x61}, {0x7a,0x37,0xa1,0x0c}, {0x8e,0x59,0xf8,0x14}, {0x89,0xeb,0x13,0x3c},  
{0xee,0xce,0xa9,0x27}, {0x35,0xb7,0x61,0xc9}, {0xed,0xe1,0x1c,0xe5}, {0x3c,0x7a,0x47,0xb1},  
{0x59,0x9c,0xd2,0xdf}, {0x3f,0x55,0xf2,0x73}, {0x79,0x18,0x14,0xce}, {0xbf,0x73,0xc7,0x37}

```
{0xea,0x53,0xf7,0xcd}, {0x5b,0x5f,0xfd,0xaa}, {0x14,0xdf,0x3d,0x6f}, {0x86,0x78,0x44,0xdb}
{0x81,0xca,0xaf,0xf3}, {0x3e,0xb9,0x68,0xc4}, {0x2c,0x38,0x24,0x34}, {0x5f,0xc2,0xa3,0x40}
{0x72,0x16,0x1d,0xc3}, {0x0c,0xbc,0xe2,0x25}, {0x8b,0x28,0x3c,0x49}, {0x41,0xff,0x0d,0x95}
{0x71,0x39,0xa8,0x01}, {0xde,0x08,0x0c,0xb3}, {0x9c,0xd8,0xb4,0xe4}, {0x90,0x64,0x56,0xc1}
{0x61,0x7b,0xcb,0x84}, {0x70,0xd5,0x32,0xb6}, {0x74,0x48,0x6c,0x5c}, {0x42,0xd0,0xb8,0x57}
;

word8 T7[256][4] = {
{0xa7,0x50,0x51,0xf4}, {0x65,0x53,0x7e,0x41}, {0xa4,0xc3,0x1a,0x17}, {0x5e,0x96,0x3a,0x27}
{0x6b,0xcb,0x3b,0xab}, {0x45,0xf1,0x1f,0x9d}, {0x58,0xab,0xac,0xfa}, {0x03,0x93,0x4b,0xe3}
{0xfa,0x55,0x20,0x30}, {0x6d,0xf6,0xad,0x76}, {0x76,0x91,0x88,0xcc}, {0x4c,0x25,0xf5,0x02}
{0xd7,0xfc,0x4f,0xe5}, {0xcb,0xd7,0xc5,0x2a}, {0x44,0x80,0x26,0x35}, {0xa3,0x8f,0xb5,0x62}
{0x5a,0x49,0xde,0xb1}, {0x1b,0x67,0x25,0xba}, {0x0e,0x98,0x45,0xea}, {0xc0,0xe1,0x5d,0xfe}
{0x75,0x02,0xc3,0x2f}, {0xf0,0x12,0x81,0x4c}, {0x97,0xa3,0x8d,0x46}, {0xf9,0xc6,0x6b,0xd3}
{0x5f,0xe7,0x03,0x8f}, {0x9c,0x95,0x15,0x92}, {0x7a,0xeb,0xbf,0x6d}, {0x59,0xda,0x95,0x52}
{0x83,0x2d,0xd4,0xbe}, {0x21,0xd3,0x58,0x74}, {0x69,0x29,0x49,0xe0}, {0xc8,0x44,0x8e,0xc9}
{0x89,0x6a,0x75,0xc2}, {0x79,0x78,0xf4,0x8e}, {0x3e,0x6b,0x99,0x58}, {0x71,0xdd,0x27,0xb9}
{0x4f,0xb6,0xbe,0xe1}, {0xad,0x17,0xf0,0x88}, {0xac,0x66,0xc9,0x20}, {0x3a,0xb4,0x7d,0xce}
{0x4a,0x18,0x63,0xdf}, {0x31,0x82,0xe5,0x1a}, {0x33,0x60,0x97,0x51}, {0x7f,0x45,0x62,0x53}
{0x77,0xe0,0xb1,0x64}, {0xae,0x84,0xbb,0x6b}, {0xa0,0x1c,0xfe,0x81}, {0x2b,0x94,0xf9,0x08}
{0x68,0x58,0x70,0x48}, {0xfd,0x19,0x8f,0x45}, {0x6c,0x87,0x94,0xde}, {0xf8,0xb7,0x52,0x7b}
{0xd3,0x23,0xab,0x73}, {0x02,0xe2,0x72,0x4b}, {0x8f,0x57,0xe3,0x1f}, {0xab,0x2a,0x66,0x55}
{0x28,0x07,0xb2,0xeb}, {0xc2,0x03,0x2f,0xb5}, {0x7b,0x9a,0x86,0xc5}, {0x08,0xa5,0xd3,0x37}
{0x87,0xf2,0x30,0x28}, {0xa5,0xb2,0x23,0xbf}, {0x6a,0xba,0x02,0x03}, {0x82,0x5c,0xed,0x16}
{0x1c,0x2b,0x8a,0xcf}, {0xb4,0x92,0xa7,0x79}, {0xf2,0xf0,0xf3,0x07}, {0xe2,0xa1,0x4e,0x69}
{0xf4,0xcd,0x65,0xda}, {0xbe,0xd5,0x06,0x05}, {0x62,0x1f,0xd1,0x34}, {0xfe,0x8a,0xc4,0xa6}
{0x53,0x9d,0x34,0x2e}, {0x55,0xa0,0xa2,0xf3}, {0xe1,0x32,0x05,0x8a}, {0xeb,0x75,0xa4,0xf6}
{0xec,0x39,0x0b,0x83}, {0xef,0xaa,0x40,0x60}, {0x9f,0x06,0x5e,0x71}, {0x10,0x51,0xbd,0x6e}
{0x8a,0xf9,0x3e,0x21}, {0x06,0x3d,0x96,0xdd}, {0x05,0xae,0xdd,0x3e}, {0xbd,0x46,0x4d,0xe6}
{0x8d,0xb5,0x91,0x54}, {0x5d,0x05,0x71,0xc4}, {0xd4,0x6f,0x04,0x06}, {0x15,0xff,0x60,0x50}
{0xfb,0x24,0x19,0x98}, {0xe9,0x97,0xd6,0xbd}, {0x43,0xcc,0x89,0x40}, {0x9e,0x77,0x67,0xd9}
{0x42,0xbd,0xb0,0xe8}, {0x8b,0x88,0x07,0x89}, {0x5b,0x38,0xe7,0x19}, {0xee,0xdb,0x79,0xc8}
{0x0a,0x47,0xa1,0x7c}, {0x0f,0xe9,0x7c,0x42}, {0x1e,0xc9,0xf8,0x84}, {0x00,0x00,0x00,0x00}
{0x86,0x83,0x09,0x80}, {0xed,0x48,0x32,0x2b}, {0x70,0xac,0x1e,0x11}, {0x72,0x4e,0x6c,0x5a}
{0xff,0xfb,0xfd,0x0e}, {0x38,0x56,0x0f,0x85}, {0xd5,0x1e,0x3d,0xae}, {0x39,0x27,0x36,0x2d}
{0xd9,0x64,0x0a,0x0f}, {0xa6,0x21,0x68,0x5c}, {0x54,0xd1,0x9b,0x5b}, {0x2e,0x3a,0x24,0x36}
{0x67,0xb1,0x0c,0x0a}, {0xe7,0x0f,0x93,0x57}, {0x96,0xd2,0xb4,0xee}, {0x91,0x9e,0x1b,0x9b}
{0xc5,0x4f,0x80,0xc0}, {0x20,0xa2,0x61,0xdc}, {0x4b,0x69,0x5a,0x77}, {0x1a,0x16,0x1c,0x12}
{0xba,0x0a,0xe2,0x93}, {0x2a,0xe5,0xc0,0xa0}, {0xe0,0x43,0x3c,0x22}, {0x17,0x1d,0x12,0x1b}
```



```
{0x0d,0x0b,0x0e,0x09}, {0xc7,0xad,0xf2,0x8b}, {0xa8,0xb9,0x2d,0xb6}, {0xa9,0xc8,0x14,0x1e
0x19,0x85,0x57,0xf1}, {0x07,0x4c,0xaf,0x75}, {0xdd,0xbb,0xee,0x99}, {0x60,0xfd,0xa3,0x7f
0x26,0x9f,0xf7,0x01}, {0xf5,0xbc,0x5c,0x72}, {0x3b,0xc5,0x44,0x66}, {0x7e,0x34,0x5b,0xfb
0x29,0x76,0x8b,0x43}, {0xc6,0xdc,0xcb,0x23}, {0xfc,0x68,0xb6,0xed}, {0xf1,0x63,0xb8,0xe4
0xdc,0xca,0xd7,0x31}, {0x85,0x10,0x42,0x63}, {0x22,0x40,0x13,0x97}, {0x11,0x20,0x84,0xc6
0x24,0x7d,0x85,0x4a}, {0x3d,0xf8,0xd2,0xbb}, {0x32,0x11,0xae,0xf9}, {0xa1,0x6d,0xc7,0x29
0x2f,0x4b,0x1d,0x9e}, {0x30,0xf3,0xdc,0xb2}, {0x52,0xec,0x0d,0x86}, {0xe3,0xd0,0x77,0xc1
0x16,0x6c,0x2b,0xb3}, {0xb9,0x99,0xa9,0x70}, {0x48,0xfa,0x11,0x94}, {0x64,0x22,0x47,0xe9
0x8c,0xc4,0xa8,0xfc}, {0x3f,0x1a,0xa0,0xf0}, {0x2c,0xd8,0x56,0x7d}, {0x90,0xef,0x22,0x33
0x4e,0xc7,0x87,0x49}, {0xd1,0xc1,0xd9,0x38}, {0xa2,0xfe,0x8c,0xca}, {0x0b,0x36,0x98,0xd4
0x81,0xcf,0xa6,0xf5}, {0xde,0x28,0xa5,0x7a}, {0x8e,0x26,0xda,0xb7}, {0xbf,0xa4,0x3f,0xad
0x9d,0xe4,0x2c,0x3a}, {0x92,0x0d,0x50,0x78}, {0xcc,0x9b,0x6a,0x5f}, {0x46,0x62,0x54,0x7e
0x13,0xc2,0xf6,0x8d}, {0xb8,0xe8,0x90,0xd8}, {0xf7,0x5e,0x2e,0x39}, {0xaf,0xf5,0x82,0xc3
0x80,0xbe,0x9f,0x5d}, {0x93,0x7c,0x69,0xd0}, {0x2d,0xa9,0x6f,0xd5}, {0x12,0xb3,0xcf,0x25
0x99,0x3b,0xc8,0xac}, {0x7d,0xa7,0x10,0x18}, {0x63,0x6e,0xe8,0x9c}, {0xbb,0x7b,0xdb,0x3b
0x78,0x09,0xcd,0x26}, {0x18,0xf4,0x6e,0x59}, {0xb7,0x01,0xec,0x9a}, {0x9a,0xa8,0x83,0x4f
0x6e,0x65,0xe6,0x95}, {0xe6,0x7e,0xaa,0xff}, {0xcf,0x08,0x21,0xbc}, {0xe8,0xe6,0xef,0x15
0x9b,0xd9,0xba,0xe7}, {0x36,0xce,0x4a,0x6f}, {0x09,0xd4,0xea,0x9f}, {0x7c,0xd6,0x29,0xb0
0xb2,0xaf,0x31,0xa4}, {0x23,0x31,0x2a,0x3f}, {0x94,0x30,0xc6,0xa5}, {0x66,0xc0,0x35,0xa2
0xbc,0x37,0x74,0x4e}, {0xca,0xa6,0xfc,0x82}, {0xd0,0xb0,0xe0,0x90}, {0xd8,0x15,0x33,0xa7
0x98,0x4a,0xf1,0x04}, {0xda,0xf7,0x41,0xec}, {0x50,0x0e,0x7f,0xcd}, {0xf6,0x2f,0x17,0x91
0xd6,0x8d,0x76,0x4d}, {0xb0,0x4d,0x43,0xef}, {0x4d,0x54,0xcc,0xaa}, {0x04,0xdf,0xe4,0x96
0xb5,0xe3,0x9e,0xd1}, {0x88,0x1b,0x4c,0x6a}, {0x1f,0xb8,0xc1,0x2c}, {0x51,0x7f,0x46,0x65
0xea,0x04,0x9d,0x5e}, {0x35,0x5d,0x01,0x8c}, {0x74,0x73,0xfa,0x87}, {0x41,0x2e,0xfb,0x0b
0x1d,0x5a,0xb3,0x67}, {0xd2,0x52,0x92,0xdb}, {0x56,0x33,0xe9,0x10}, {0x47,0x13,0x6d,0xd6
0x61,0x8c,0x9a,0xd7}, {0x0c,0x7a,0x37,0xa1}, {0x14,0x8e,0x59,0xf8}, {0x3c,0x89,0xeb,0x13
0x27,0xee,0xce,0xa9}, {0xc9,0x35,0xb7,0x61}, {0xe5,0xed,0xe1,0x1c}, {0xb1,0x3c,0x7a,0x47
0xdf,0x59,0x9c,0xd2}, {0x73,0x3f,0x55,0xf2}, {0xce,0x79,0x18,0x14}, {0x37,0xbf,0x73,0xc7
0xcd,0xea,0x53,0xf7}, {0xaa,0x5b,0x5f,0xfd}, {0x6f,0x14,0xdf,0x3d}, {0xdb,0x86,0x78,0x44
0xf3,0x81,0xca,0xaf}, {0xc4,0x3e,0xb9,0x68}, {0x34,0x2c,0x38,0x24}, {0x40,0x5f,0xc2,0xa3
0xc3,0x72,0x16,0x1d}, {0x25,0x0c,0xbc,0xe2}, {0x49,0x8b,0x28,0x3c}, {0x95,0x41,0xff,0x0d
0x01,0x71,0x39,0xa8}, {0xb3,0xde,0x08,0x0c}, {0xe4,0x9c,0xd8,0xb4}, {0xc1,0x90,0x64,0x56
0x84,0x61,0x7b,0xcb}, {0xb6,0x70,0xd5,0x32}, {0x5c,0x74,0x48,0x6c}, {0x57,0x42,0xd0,0xb8
;
word8 T8[256][4] = {
0xf4,0xa7,0x50,0x51}, {0x41,0x65,0x53,0x7e}, {0x17,0xa4,0xc3,0x1a}, {0x27,0x5e,0x96,0x3a
0xab,0x6b,0xcb,0x3b}, {0x9d,0x45,0xf1,0x1f}, {0xfa,0x58,0xab,0xac}, {0xe3,0x03,0x93,0x4b
;
```

{0x30,0xfa,0x55,0x20}, {0x76,0x6d,0xf6,0xad}, {0xcc,0x76,0x91,0x88}, {0x02,0x4c,0x25,0xf5},  
{0xe5,0xd7,0xfc,0x4f}, {0x2a,0xcb,0xd7,0xc5}, {0x35,0x44,0x80,0x26}, {0x62,0xa3,0x8f,0xb5},  
{0xb1,0x5a,0x49,0xde}, {0xba,0x1b,0x67,0x25}, {0xea,0x0e,0x98,0x45}, {0xfe,0xc0,0xe1,0x5d},  
{0x2f,0x75,0x02,0xc3}, {0x4c,0xf0,0x12,0x81}, {0x46,0x97,0xa3,0x8d}, {0xd3,0xf9,0xc6,0x6b},  
{0x8f,0x5f,0xe7,0x03}, {0x92,0x9c,0x95,0x15}, {0x6d,0x7a,0xeb,0xbf}, {0x52,0x59,0xda,0x95},  
{0xbe,0x83,0x2d,0xd4}, {0x74,0x21,0xd3,0x58}, {0xe0,0x69,0x29,0x49}, {0xc9,0xc8,0x44,0x8e},  
{0xc2,0x89,0x6a,0x75}, {0x8e,0x79,0x78,0xf4}, {0x58,0x3e,0x6b,0x99}, {0xb9,0x71,0xdd,0x27},  
{0xe1,0x4f,0xb6,0xbe}, {0x88,0xad,0x17,0xf0}, {0x20,0xac,0x66,0xc9}, {0xce,0x3a,0xb4,0x7d},  
{0xdf,0x4a,0x18,0x63}, {0x1a,0x31,0x82,0xe5}, {0x51,0x33,0x60,0x97}, {0x53,0x7f,0x45,0x62},  
{0x64,0x77,0xe0,0xb1}, {0x6b,0xae,0x84,0xbb}, {0x81,0xa0,0x1c,0xfe}, {0x08,0x2b,0x94,0xf9},  
{0x48,0x68,0x58,0x70}, {0x45,0xfd,0x19,0x8f}, {0xde,0x6c,0x87,0x94}, {0x7b,0xf8,0xb7,0x52},  
{0x73,0xd3,0x23,0xab}, {0x4b,0x02,0xe2,0x72}, {0x1f,0x8f,0x57,0xe3}, {0x55,0xab,0x2a,0x66},  
{0xeb,0x28,0x07,0xb2}, {0xb5,0xc2,0x03,0x2f}, {0xc5,0x7b,0x9a,0x86}, {0x37,0x08,0xa5,0xd3},  
{0x28,0x87,0xf2,0x30}, {0xbf,0xa5,0xb2,0x23}, {0x03,0x6a,0xba,0x02}, {0x16,0x82,0x5c,0xed},  
{0xcf,0x1c,0x2b,0x8a}, {0x79,0xb4,0x92,0xa7}, {0x07,0xf2,0xf0,0xf3}, {0x69,0xe2,0xa1,0x4e},  
{0xda,0xf4,0xcd,0x65}, {0x05,0xbe,0xd5,0x06}, {0x34,0x62,0x1f,0xd1}, {0xa6,0xfe,0x8a,0xc4},  
{0x2e,0x53,0x9d,0x34}, {0xf3,0x55,0xa0,0xa2}, {0x8a,0xe1,0x32,0x05}, {0xf6,0xeb,0x75,0xa4},  
{0x83,0xec,0x39,0x0b}, {0x60,0xef,0xaa,0x40}, {0x71,0x9f,0x06,0x5e}, {0x6e,0x10,0x51,0xbd},  
{0x21,0x8a,0xf9,0x3e}, {0xdd,0x06,0x3d,0x96}, {0x3e,0x05,0xae,0xdd}, {0xe6,0xbd,0x46,0x4d},  
{0x54,0x8d,0xb5,0x91}, {0xc4,0x5d,0x05,0x71}, {0x06,0xd4,0x6f,0x04}, {0x50,0x15,0xff,0x60},  
{0x98,0xfb,0x24,0x19}, {0xbd,0xe9,0x97,0xd6}, {0x40,0x43,0xcc,0x89}, {0xd9,0x9e,0x77,0x67},  
{0xe8,0x42,0xbd,0xb0}, {0x89,0x8b,0x88,0x07}, {0x19,0x5b,0x38,0xe7}, {0xc8,0xee,0xdb,0x79},  
{0x7c,0x0a,0x47,0xa1}, {0x42,0x0f,0xe9,0x7c}, {0x84,0x1e,0xc9,0xf8}, {0x00,0x00,0x00,0x00},  
{0x80,0x86,0x83,0x09}, {0x2b,0xed,0x48,0x32}, {0x11,0x70,0xac,0x1e}, {0x5a,0x72,0x4e,0x6c},  
{0x0e,0xff,0xfb,0xfd}, {0x85,0x38,0x56,0x0f}, {0xae,0xd5,0x1e,0x3d}, {0x2d,0x39,0x27,0x36},  
{0x0f,0xd9,0x64,0x0a}, {0x5c,0xa6,0x21,0x68}, {0x5b,0x54,0xd1,0x9b}, {0x36,0x2e,0x3a,0x24},  
{0x0a,0x67,0xb1,0x0c}, {0x57,0xe7,0x0f,0x93}, {0xee,0x96,0xd2,0xb4}, {0x9b,0x91,0x9e,0x1b},  
{0xc0,0xc5,0x4f,0x80}, {0xdc,0x20,0xa2,0x61}, {0x77,0x4b,0x69,0x5a}, {0x12,0x1a,0x16,0x1c},  
{0x93,0xba,0x0a,0xe2}, {0xa0,0x2a,0xe5,0xc0}, {0x22,0xe0,0x43,0x3c}, {0x1b,0x17,0x1d,0x12},  
{0x09,0x0d,0x0b,0x0e}, {0x8b,0xc7,0xad,0xf2}, {0xb6,0xa8,0xb9,0x2d}, {0x1e,0xa9,0xc8,0x14},  
{0xf1,0x19,0x85,0x57}, {0x75,0x07,0x4c,0xaf}, {0x99,0xdd,0xbb,0xee}, {0x7f,0x60,0xfd,0xa3},  
{0x01,0x26,0x9f,0xf7}, {0x72,0xf5,0xbc,0x5c}, {0x66,0x3b,0xc5,0x44}, {0xfb,0x7e,0x34,0x5b},  
{0x43,0x29,0x76,0x8b}, {0x23,0xc6,0xdc,0xcb}, {0xed,0xfc,0x68,0xb6}, {0xe4,0xf1,0x63,0xb8},  
{0x31,0xdc,0xca,0xd7}, {0x63,0x85,0x10,0x42}, {0x97,0x22,0x40,0x13}, {0xc6,0x11,0x20,0x84},  
{0x4a,0x24,0x7d,0x85}, {0xbb,0x3d,0xf8,0xd2}, {0xf9,0x32,0x11,0xae}, {0x29,0xa1,0x6d,0xc7},  
{0x9e,0x2f,0x4b,0x1d}, {0xb2,0x30,0xf3,0xdc}, {0x86,0x52,0xec,0x0d}, {0xc1,0xe3,0xd0,0x77},  
{0xb3,0x16,0x6c,0x2b}, {0x70,0xb9,0x99,0xa9}, {0x94,0x48,0xfa,0x11}, {0xe9,0x64,0x22,0x47},  
,

```
{0xfc,0x8c,0xc4,0xa8}, {0xf0,0x3f,0x1a,0xa0}, {0x7d,0x2c,0xd8,0x56}, {0x33,0x90,0xef,0x22},
{0x49,0x4e,0xc7,0x87}, {0x38,0xd1,0xc1,0xd9}, {0xca,0xa2,0xfe,0x8c}, {0xd4,0x0b,0x36,0x98},
{0xf5,0x81,0xcf,0xa6}, {0x7a,0xde,0x28,0xa5}, {0xb7,0x8e,0x26,0xda}, {0xad,0xbf,0xa4,0x3f},
{0x3a,0x9d,0xe4,0x2c}, {0x78,0x92,0x0d,0x50}, {0x5f,0xcc,0x9b,0x6a}, {0x7e,0x46,0x62,0x54},
{0x8d,0x13,0xc2,0xf6}, {0xd8,0xb8,0xe8,0x90}, {0x39,0xf7,0x5e,0x2e}, {0xc3,0xaf,0xf5,0x82},
{0x5d,0x80,0xbe,0x9f}, {0xd0,0x93,0x7c,0x69}, {0xd5,0x2d,0xa9,0x6f}, {0x25,0x12,0xb3,0xcf},
{0xac,0x99,0x3b,0xc8}, {0x18,0x7d,0xa7,0x10}, {0x9c,0x63,0x6e,0xe8}, {0x3b,0xbb,0x7b,0xdb},
{0x26,0x78,0x09,0xcd}, {0x59,0x18,0xf4,0x6e}, {0x9a,0xb7,0x01,0xec}, {0x4f,0x9a,0xa8,0x83},
{0x95,0x6e,0x65,0xe6}, {0xff,0xe6,0x7e,0xaa}, {0xbc,0xcf,0x08,0x21}, {0x15,0xe8,0xe6,0xef},
{0xe7,0x9b,0xd9,0xba}, {0x6f,0x36,0xce,0x4a}, {0x9f,0x09,0xd4,0xea}, {0xb0,0x7c,0xd6,0x29},
{0xa4,0xb2,0xaf,0x31}, {0x3f,0x23,0x31,0x2a}, {0xa5,0x94,0x30,0xc6}, {0xa2,0x66,0xc0,0x35},
{0x4e,0xbc,0x37,0x74}, {0x82,0xca,0xa6,0xfc}, {0x90,0xd0,0xb0,0xe0}, {0xa7,0xd8,0x15,0x33},
{0x04,0x98,0x4a,0xf1}, {0xec,0xda,0xf7,0x41}, {0xcd,0x50,0x0e,0x7f}, {0x91,0xf6,0x2f,0x17},
{0x4d,0xd6,0x8d,0x76}, {0xef,0xb0,0x4d,0x43}, {0xaa,0x4d,0x54,0xcc}, {0x96,0x04,0xdf,0xe4},
{0xd1,0xb5,0xe3,0x9e}, {0x6a,0x88,0x1b,0x4c}, {0x2c,0x1f,0xb8,0xc1}, {0x65,0x51,0x7f,0x46},
{0x5e,0xea,0x04,0x9d}, {0x8c,0x35,0x5d,0x01}, {0x87,0x74,0x73,0xfa}, {0x0b,0x41,0x2e,0xfb},
{0x67,0x1d,0x5a,0xb3}, {0xdb,0xd2,0x52,0x92}, {0x10,0x56,0x33,0xe9}, {0xd6,0x47,0x13,0x6d},
{0xd7,0x61,0x8c,0x9a}, {0xa1,0x0c,0x7a,0x37}, {0xf8,0x14,0x8e,0x59}, {0x13,0x3c,0x89,0xeb},
{0xa9,0x27,0xee,0xce}, {0x61,0xc9,0x35,0xb7}, {0x1c,0xe5,0xed,0xe1}, {0x47,0xb1,0x3c,0x7a},
{0xd2,0xdf,0x59,0x9c}, {0xf2,0x73,0x3f,0x55}, {0x14,0xce,0x79,0x18}, {0xc7,0x37,0xbf,0x73},
{0xf7,0xcd,0xea,0x53}, {0xfd,0xaa,0x5b,0x5f}, {0x3d,0x6f,0x14,0xdf}, {0x44,0xdb,0x86,0x78},
{0xaf,0xf3,0x81,0xca}, {0x68,0xc4,0x3e,0xb9}, {0x24,0x34,0x2c,0x38}, {0xa3,0x40,0x5f,0xc2},
{0x1d,0xc3,0x72,0x16}, {0xe2,0x25,0x0c,0xbc}, {0x3c,0x49,0x8b,0x28}, {0x0d,0x95,0x41,0xff},
{0xa8,0x01,0x71,0x39}, {0x0c,0xb3,0xde,0x08}, {0xb4,0xe4,0x9c,0xd8}, {0x56,0xc1,0x90,0x64},
{0xcb,0x84,0x61,0x7b}, {0x32,0xb6,0x70,0xd5}, {0x6c,0x5c,0x74,0x48}, {0xb8,0x57,0x42,0xd0},
;
```

```
word8 S5[256] = {
0x52,0x09,0x6a,0xd5,
0x30,0x36,0xa5,0x38,
0xbf,0x40,0xa3,0x9e,
0x81,0xf3,0xd7,0xfb,
0x7c,0xe3,0x39,0x82,
0x9b,0x2f,0xff,0x87,
0x34,0x8e,0x43,0x44,
0xc4,0xde,0xe9,0xcb,
0x54,0x7b,0x94,0x32,
0xa6,0xc2,0x23,0x3d,
0xee,0x4c,0x95,0x0b,
0x42,0xfa,0xc3,0x4e,
0x08,0x2e,0xa1,0x66,
0x28,0xd9,0x24,0xb2,
0x76,0x5b,0xa2,0x49,
0x6d,0x8b,0xd1,0x25,
0x72,0xf8,0xf6,0x64,
0x86,0x68,0x98,0x16,
0xd4,0xa4,0x5c,0xcc,
0x5d,0x65,0xb6,0x92,
0x6c,0x70,0x48,0x50,
```

```
0xfd,0xed,0xb9,0xda,
0x5e,0x15,0x46,0x57,
0xa7,0x8d,0x9d,0x84,
0x90,0xd8,0xab,0x00,
0x8c,0xbc,0xd3,0x0a,
0xf7,0xe4,0x58,0x05,
0xb8,0xb3,0x45,0x06,
0xd0,0x2c,0x1e,0x8f,
0xca,0x3f,0x0f,0x02,
0xc1,0xaf,0xbd,0x03,
0x01,0x13,0x8a,0x6b,
0x3a,0x91,0x11,0x41,
0x4f,0x67,0xdc,0xea,
0x97,0xf2,0xcf,0xce,
0xf0,0xb4,0xe6,0x73,
0x96,0xac,0x74,0x22,
0xe7,0xad,0x35,0x85,
0xe2,0xf9,0x37,0xe8,
0x1c,0x75,0xdf,0x6e,
0x47,0xf1,0x1a,0x71,
0x1d,0x29,0xc5,0x89,
0x6f,0xb7,0x62,0x0e,
0xaa,0x18,0xbe,0x1b,
0xfc,0x56,0x3e,0x4b,
0xc6,0xd2,0x79,0x20,
0x9a,0xdb,0xc0,0xfe,
0x78,0xcd,0x5a,0xf4,
0x1f,0xdd,0xa8,0x33,
0x88,0x07,0xc7,0x31,
0xb1,0x12,0x10,0x59,
0x27,0x80,0xec,0x5f,
0x60,0x51,0x7f,0xa9,
0x19,0xb5,0x4a,0x0d,
0x2d,0xe5,0x7a,0x9f,
0x93,0xc9,0x9c,0xef,
0xa0,0xe0,0x3b,0x4d,
0xae,0x2a,0xf5,0xb0,
0xc8,0xeb,0xbb,0x3c,
0x83,0x53,0x99,0x61,
0x17,0x2b,0x04,0x7e,
0xba,0x77,0xd6,0x26,
0xe1,0x69,0x14,0x63,
0x55,0x21,0x0c,0x7d,
};
```

```
word8 U1[256][4] = {
{0x00,0x00,0x00,0x00}, {0x0e,0x09,0x0d,0x0b}, {0x1c,0x12,0x1a,0x16}, {0x12,0x1b,0x17,0x1d},
{0x38,0x24,0x34,0x2c}, {0x36,0x2d,0x39,0x27}, {0x24,0x36,0x2e,0x3a}, {0x2a,0x3f,0x23,0x31},
{0x70,0x48,0x68,0x58}, {0x7e,0x41,0x65,0x53}, {0x6c,0x5a,0x72,0x4e}, {0x62,0x53,0x7f,0x45},
{0x48,0x6c,0x5c,0x74}, {0x46,0x65,0x51,0x7f}, {0x54,0x7e,0x46,0x62}, {0x5a,0x77,0x4b,0x69},
{0xe0,0x90,0xd0,0xb0}, {0xee,0x99,0xdd,0xbb}, {0xfc,0x82,0xca,0xa6}, {0xf2,0x8b,0xc7,0xad},
{0xd8,0xb4,0xe4,0x9c}, {0xd6,0xbd,0xe9,0x97}, {0xc4,0xa6,0xfe,0x8a}, {0xca,0xaf,0xf3,0x81},
{0x90,0xd8,0xb8,0xe8}, {0x9e,0xd1,0xb5,0xe3}, {0x8c,0xca,0xa2,0xfe}, {0x82,0xc3,0xaf,0xf5},
{0xa8,0xfc,0x8c,0xc4}, {0xa6,0xf5,0x81,0xcf}, {0xb4,0xee,0x96,0xd2}, {0xba,0xe7,0x9b,0xd9},
{0xdb,0x3b,0xbb,0x7b}, {0xd5,0x32,0xb6,0x70}, {0xc7,0x29,0xa1,0x6d}, {0xc9,0x20,0xac,0x66},
{0xe3,0x1f,0x8f,0x57}, {0xed,0x16,0x82,0x5c}, {0xff,0x0d,0x95,0x41}, {0xf1,0x04,0x98,0x4a},
{0xab,0x73,0xd3,0x23}, {0xa5,0x7a,0xde,0x28}, {0xb7,0x61,0xc9,0x35}, {0xb9,0x68,0xc4,0x3e},
{0x93,0x57,0xe7,0x0f}, {0x9d,0x5e,0xea,0x04}, {0x8f,0x45,0xfd,0x19}, {0x81,0x4c,0xf0,0x12},
{0x3b,0xab,0x6b,0xcb}, {0x35,0xa2,0x66,0xc0}, {0x27,0xb9,0x71,0xdd}, {0x29,0xb0,0x7c,0xd6},
{0x03,0x8f,0x5f,0xe7}, {0x0d,0x86,0x52,0xec}, {0x1f,0x9d,0x45,0xf1}, {0x11,0x94,0x48,0xfa},
};
```

{0x4b,0xe3,0x03,0x93}, {0x45,0xea,0x0e,0x98}, {0x57,0xf1,0x19,0x85}, {0x59,0xf8,0x14,0x8e},  
{0x73,0xc7,0x37,0xbf}, {0x7d,0xce,0x3a,0xb4}, {0x6f,0xd5,0x2d,0xa9}, {0x61,0xdc,0x20,0xa2},  
{0xad,0x76,0x6d,0xf6}, {0xa3,0x7f,0x60,0xfd}, {0xb1,0x64,0x77,0xe0}, {0xbf,0x6d,0x7a,0xeb},  
{0x95,0x52,0x59,0xda}, {0x9b,0x5b,0x54,0xd1}, {0x89,0x40,0x43,0xcc}, {0x87,0x49,0x4e,0xc7},  
{0xdd,0x3e,0x05,0xae}, {0xd3,0x37,0x08,0xa5}, {0xc1,0x2c,0x1f,0xb8}, {0xcf,0x25,0x12,0xb3},  
{0xe5,0x1a,0x31,0x82}, {0xeb,0x13,0x3c,0x89}, {0xf9,0x08,0x2b,0x94}, {0xf7,0x01,0x26,0x9f},  
{0x4d,0xe6,0xbd,0x46}, {0x43,0xef,0xb0,0x4d}, {0x51,0xf4,0xa7,0x50}, {0x5f,0xfd,0xaa,0x5b},  
{0x75,0xc2,0x89,0x6a}, {0x7b,0xcb,0x84,0x61}, {0x69,0xd0,0x93,0x7c}, {0x67,0xd9,0x9e,0x77},  
{0x3d,0xae,0xd5,0x1e}, {0x33,0xa7,0xd8,0x15}, {0x21,0xbc,0xcf,0x08}, {0x2f,0xb5,0xc2,0x03},  
{0x05,0x8a,0xe1,0x32}, {0x0b,0x83,0xec,0x39}, {0x19,0x98,0xfb,0x24}, {0x17,0x91,0xf6,0x2f},  
{0x76,0x4d,0xd6,0x8d}, {0x78,0x44,0xdb,0x86}, {0x6a,0x5f,0xcc,0x9b}, {0x64,0x56,0xc1,0x90},  
{0x4e,0x69,0xe2,0xa1}, {0x40,0x60,0xef,0xaa}, {0x52,0x7b,0xf8,0xb7}, {0x5c,0x72,0xf5,0xbc},  
{0x06,0x05,0xbe,0xd5}, {0x08,0x0c,0xb3,0xde}, {0x1a,0x17,0xa4,0xc3}, {0x14,0x1e,0xa9,0xc8},  
{0x3e,0x21,0x8a,0xf9}, {0x30,0x28,0x87,0xf2}, {0x22,0x33,0x90,0xef}, {0x2c,0x3a,0x9d,0xe4},  
{0x96,0xdd,0x06,0x3d}, {0x98,0xd4,0x0b,0x36}, {0x8a,0xcf,0x1c,0x2b}, {0x84,0xc6,0x11,0x20},  
{0xae,0xf9,0x32,0x11}, {0xa0,0xf0,0x3f,0x1a}, {0xb2,0xeb,0x28,0x07}, {0xbc,0xe2,0x25,0x0c},  
{0xe6,0x95,0x6e,0x65}, {0xe8,0x9c,0x63,0x6e}, {0xfa,0x87,0x74,0x73}, {0xf4,0x8e,0x79,0x78},  
{0xde,0xb1,0x5a,0x49}, {0xd0,0xb8,0x57,0x42}, {0xc2,0xa3,0x40,0x5f}, {0xcc,0xaa,0x4d,0x54},  
{0x41,0xec,0xda,0xf7}, {0x4f,0xe5,0xd7,0xfc}, {0x5d,0xfe,0xc0,0xe1}, {0x53,0xf7,0xcd,0xea},  
{0x79,0xc8,0xee,0xdb}, {0x77,0xc1,0xe3,0xd0}, {0x65,0xda,0xf4,0xcd}, {0x6b,0xd3,0xf9,0xc6},  
{0x31,0xa4,0xb2,0xaf}, {0x3f,0xad,0xbf,0xa4}, {0x2d,0xb6,0xa8,0xb9}, {0x23,0xbf,0xa5,0xb2},  
{0x09,0x80,0x86,0x83}, {0x07,0x89,0x8b,0x88}, {0x15,0x92,0x9c,0x95}, {0x1b,0x9b,0x91,0x9e},  
{0xa1,0x7c,0x0a,0x47}, {0xaf,0x75,0x07,0x4c}, {0xbd,0x6e,0x10,0x51}, {0xb3,0x67,0x1d,0x5a},  
{0x99,0x58,0x3e,0x6b}, {0x97,0x51,0x33,0x60}, {0x85,0x4a,0x24,0x7d}, {0x8b,0x43,0x29,0x76},  
{0xd1,0x34,0x62,0x1f}, {0xdf,0x3d,0x6f,0x14}, {0xcd,0x26,0x78,0x09}, {0xc3,0x2f,0x75,0x02},  
{0xe9,0x10,0x56,0x33}, {0xe7,0x19,0x5b,0x38}, {0xf5,0x02,0x4c,0x25}, {0xfb,0x0b,0x41,0x2e},  
{0x9a,0xd7,0x61,0x8c}, {0x94,0xde,0x6c,0x87}, {0x86,0xc5,0x7b,0x9a}, {0x88,0xcc,0x76,0x91},  
{0xa2,0xf3,0x55,0xa0}, {0xac,0xfa,0x58,0xab}, {0xbe,0xe1,0x4f,0xb6}, {0xb0,0xe8,0x42,0xbd},  
{0xea,0x9f,0x09,0xd4}, {0xe4,0x96,0x04,0xdf}, {0xf6,0x8d,0x13,0xc2}, {0xf8,0x84,0x1e,0xc9},  
{0xd2,0xbb,0x3d,0xf8}, {0xdc,0xb2,0x30,0xf3}, {0xce,0xa9,0x27,0xee}, {0xc0,0xa0,0x2a,0xe5},  
{0x7a,0x47,0xb1,0x3c}, {0x74,0x4e,0xbc,0x37}, {0x66,0x55,0xab,0x2a}, {0x68,0x5c,0xa6,0x21},  
{0x42,0x63,0x85,0x10}, {0x4c,0x6a,0x88,0x1b}, {0x5e,0x71,0x9f,0x06}, {0x50,0x78,0x92,0x0d},  
{0x0a,0x0f,0xd9,0x64}, {0x04,0x06,0xd4,0x6f}, {0x16,0x1d,0xc3,0x72}, {0x18,0x14,0xce,0x79},  
{0x32,0x2b,0xed,0x48}, {0x3c,0x22,0xe0,0x43}, {0x2e,0x39,0xf7,0x5e}, {0x20,0x30,0xfa,0x55},  
{0xec,0x9a,0xb7,0x01}, {0xe2,0x93,0xba,0x0a}, {0xf0,0x88,0xad,0x17}, {0xfe,0x81,0xa0,0x1c},  
{0xd4,0xbe,0x83,0x2d}, {0xda,0xb7,0x8e,0x26}, {0xc8,0xac,0x99,0x3b}, {0xc6,0xa5,0x94,0x30},  
{0x9c,0xd2,0xdf,0x59}, {0x92,0xdb,0xd2,0x52}, {0x80,0xc0,0xc5,0x4f}, {0x8e,0xc9,0xc8,0x44}

```
{0xa4,0xf6,0xeb,0x75}, {0xaa,0xff,0xe6,0x7e}, {0xb8,0xe4,0xf1,0x63}, {0xb6,0xed,0xfc,0x68},
{0x0c,0x0a,0x67,0xb1}, {0x02,0x03,0x6a,0xba}, {0x10,0x18,0x7d,0xa7}, {0x1e,0x11,0x70,0xac},
{0x34,0x2e,0x53,0x9d}, {0x3a,0x27,0x5e,0x96}, {0x28,0x3c,0x49,0x8b}, {0x26,0x35,0x44,0x80},
{0x7c,0x42,0x0f,0xe9}, {0x72,0x4b,0x02,0xe2}, {0x60,0x50,0x15,0xff}, {0x6e,0x59,0x18,0xf4},
{0x44,0x66,0x3b,0xc5}, {0x4a,0x6f,0x36,0xce}, {0x58,0x74,0x21,0xd3}, {0x56,0x7d,0x2c,0xd8},
{0x37,0xa1,0x0c,0x7a}, {0x39,0xa8,0x01,0x71}, {0x2b,0xb3,0x16,0x6c}, {0x25,0xba,0x1b,0x67},
{0x0f,0x85,0x38,0x56}, {0x01,0x8c,0x35,0x5d}, {0x13,0x97,0x22,0x40}, {0x1d,0x9e,0x2f,0x4b},
{0x47,0xe9,0x64,0x22}, {0x49,0xe0,0x69,0x29}, {0x5b,0xfb,0x7e,0x34}, {0x55,0xf2,0x73,0x3f},
{0x7f,0xcd,0x50,0x0e}, {0x71,0xc4,0x5d,0x05}, {0x63,0xdf,0x4a,0x18}, {0x6d,0xd6,0x47,0x13},
{0xd7,0x31,0xdc,0xca}, {0xd9,0x38,0xd1,0xc1}, {0xcb,0x23,0xc6,0xdc}, {0xc5,0x2a,0xcb,0xd7},
{0xef,0x15,0xe8,0xe6}, {0xe1,0x1c,0xe5,0xed}, {0xf3,0x07,0xf2,0xf0}, {0xfd,0x0e,0xff,0xfb},
{0xa7,0x79,0xb4,0x92}, {0xa9,0x70,0xb9,0x99}, {0xbb,0x6b,0xae,0x84}, {0xb5,0x62,0xa3,0x8f},
{0x9f,0x5d,0x80,0xbe}, {0x91,0x54,0x8d,0xb5}, {0x83,0x4f,0x9a,0xa8}, {0x8d,0x46,0x97,0xa3},
:,
```

```
word8 U2[256][4] = {
{0x00,0x00,0x00,0x00}, {0x0b,0x0e,0x09,0x0d}, {0x16,0x1c,0x12,0x1a}, {0x1d,0x12,0x1b,0x17},
{0x2c,0x38,0x24,0x34}, {0x27,0x36,0x2d,0x39}, {0x3a,0x24,0x36,0x2e}, {0x31,0x2a,0x3f,0x23},
{0x58,0x70,0x48,0x68}, {0x53,0x7e,0x41,0x65}, {0x4e,0x6c,0x5a,0x72}, {0x45,0x62,0x53,0x7f},
{0x74,0x48,0x6c,0x5c}, {0x7f,0x46,0x65,0x51}, {0x62,0x54,0x7e,0x46}, {0x69,0x5a,0x77,0x4b},
{0xb0,0xe0,0x90,0xd0}, {0xbb,0xee,0x99,0xdd}, {0xa6,0xfc,0x82,0xca}, {0xad,0xf2,0x8b,0xc7},
{0x9c,0xd8,0xb4,0xe4}, {0x97,0xd6,0xbd,0xe9}, {0x8a,0xc4,0xa6,0xfe}, {0x81,0xca,0xaf,0xf3},
{0xe8,0x90,0xd8,0xb8}, {0xe3,0x9e,0xd1,0xb5}, {0xfe,0x8c,0xca,0xa2}, {0xf5,0x82,0xc3,0xaf},
{0xc4,0xa8,0xfc,0x8c}, {0xcf,0xa6,0xf5,0x81}, {0xd2,0xb4,0xee,0x96}, {0xd9,0xba,0xe7,0x9b},
{0x7b,0xdb,0x3b,0xbb}, {0x70,0xd5,0x32,0xb6}, {0x6d,0xc7,0x29,0xa1}, {0x66,0xc9,0x20,0xac},
{0x57,0xe3,0x1f,0x8f}, {0x5c,0xed,0x16,0x82}, {0x41,0xff,0x0d,0x95}, {0x4a,0xf1,0x04,0x98},
{0x23,0xab,0x73,0xd3}, {0x28,0xa5,0x7a,0xde}, {0x35,0xb7,0x61,0xc9}, {0x3e,0xb9,0x68,0xc4},
{0x0f,0x93,0x57,0xe7}, {0x04,0x9d,0x5e,0xea}, {0x19,0x8f,0x45,0xfd}, {0x12,0x81,0x4c,0xf0},
{0xcb,0x3b,0xab,0x6b}, {0xc0,0x35,0xa2,0x66}, {0xdd,0x27,0xb9,0x71}, {0xd6,0x29,0xb0,0x7c},
{0xe7,0x03,0x8f,0x5f}, {0xec,0x0d,0x86,0x52}, {0xf1,0x1f,0x9d,0x45}, {0xfa,0x11,0x94,0x48},
{0x93,0x4b,0xe3,0x03}, {0x98,0x45,0xea,0x0e}, {0x85,0x57,0xf1,0x19}, {0x8e,0x59,0xf8,0x14},
{0xbf,0x73,0xc7,0x37}, {0xb4,0x7d,0xce,0x3a}, {0xa9,0x6f,0xd5,0x2d}, {0xa2,0x61,0xdc,0x20},
{0xf6,0xad,0x76,0x6d}, {0xfd,0xa3,0x7f,0x60}, {0xe0,0xb1,0x64,0x77}, {0xeb,0xbf,0x6d,0x7a},
{0xda,0x95,0x52,0x59}, {0xd1,0x9b,0x5b,0x54}, {0xcc,0x89,0x40,0x43}, {0xc7,0x87,0x49,0x4e},
{0xae,0xdd,0x3e,0x05}, {0xa5,0xd3,0x37,0x08}, {0xb8,0xc1,0x2c,0x1f}, {0xb3,0xcf,0x25,0x12},
{0x82,0xe5,0x1a,0x31}, {0x89,0xeb,0x13,0x3c}, {0x94,0xf9,0x08,0x2b}, {0x9f,0xf7,0x01,0x26},
{0x46,0x4d,0xe6,0xbd}, {0x4d,0x43,0xef,0xb0}, {0x50,0x51,0xf4,0xa7}, {0x5b,0x5f,0xfd,0xaa},
{0x6a,0x75,0xc2,0x89}, {0x61,0x7b,0xcb,0x84}, {0x7c,0x69,0xd0,0x93}, {0x77,0x67,0xd9,0x9e},
{0x1e,0x3d,0xae,0xd5}, {0x15,0x33,0xa7,0xd8}, {0x08,0x21,0xbc,0xcf}, {0x03,0x2f,0xb5,0xc2}
```

```
{0x32,0x05,0x8a,0xe1}, {0x39,0x0b,0x83,0xec}, {0x24,0x19,0x98,0xfb}, {0x2f,0x17,0x91,0xf6}
{0x8d,0x76,0x4d,0xd6}, {0x86,0x78,0x44,0xdb}, {0x9b,0x6a,0x5f,0xcc}, {0x90,0x64,0x56,0xc1}
{0xa1,0x4e,0x69,0xe2}, {0xaa,0x40,0x60,0xef}, {0xb7,0x52,0x7b,0xf8}, {0xbc,0x5c,0x72,0xf5}
{0xd5,0x06,0x05,0xbe}, {0xde,0x08,0x0c,0xb3}, {0xc3,0x1a,0x17,0xa4}, {0xc8,0x14,0x1e,0xa9}
{0xf9,0x3e,0x21,0x8a}, {0xf2,0x30,0x28,0x87}, {0xef,0x22,0x33,0x90}, {0xe4,0x2c,0x3a,0x9d}
{0x3d,0x96,0xdd,0x06}, {0x36,0x98,0xd4,0x0b}, {0x2b,0x8a,0xcf,0x1c}, {0x20,0x84,0xc6,0x11}
{0x11,0xae,0xf9,0x32}, {0x1a,0xa0,0xf0,0x3f}, {0x07,0xb2,0xeb,0x28}, {0x0c,0xbc,0xe2,0x25}
{0x65,0xe6,0x95,0x6e}, {0x6e,0xe8,0x9c,0x63}, {0x73,0xfa,0x87,0x74}, {0x78,0xf4,0x8e,0x79}
{0x49,0xde,0xb1,0x5a}, {0x42,0xd0,0xb8,0x57}, {0x5f,0xc2,0xa3,0x40}, {0x54,0xcc,0xaa,0x4d}
{0xf7,0x41,0xec,0xda}, {0xfc,0x4f,0xe5,0xd7}, {0xe1,0x5d,0xfe,0xc0}, {0xea,0x53,0xf7,0xcd}
{0xdb,0x79,0xc8,0xee}, {0xd0,0x77,0xc1,0xe3}, {0xcd,0x65,0xda,0xf4}, {0xc6,0x6b,0xd3,0xf9}
{0xaf,0x31,0xa4,0xb2}, {0xa4,0x3f,0xad,0xbf}, {0xb9,0x2d,0xb6,0xa8}, {0xb2,0x23,0xbf,0xa5}
{0x83,0x09,0x80,0x86}, {0x88,0x07,0x89,0x8b}, {0x95,0x15,0x92,0x9c}, {0x9e,0x1b,0x9b,0x91}
{0x47,0xa1,0x7c,0x0a}, {0x4c,0xaf,0x75,0x07}, {0x51,0xbd,0x6e,0x10}, {0x5a,0xb3,0x67,0x1d}
{0x6b,0x99,0x58,0x3e}, {0x60,0x97,0x51,0x33}, {0x7d,0x85,0x4a,0x24}, {0x76,0x8b,0x43,0x29}
{0x1f,0xd1,0x34,0x62}, {0x14,0xdf,0x3d,0x6f}, {0x09,0xcd,0x26,0x78}, {0x02,0xc3,0x2f,0x75}
{0x33,0xe9,0x10,0x56}, {0x38,0xe7,0x19,0x5b}, {0x25,0xf5,0x02,0x4c}, {0x2e,0xfb,0x0b,0x41}
{0x8c,0x9a,0xd7,0x61}, {0x87,0x94,0xde,0x6c}, {0x9a,0x86,0xc5,0x7b}, {0x91,0x88,0xcc,0x76}
{0xa0,0xa2,0xf3,0x55}, {0xab,0xac,0xfa,0x58}, {0xb6,0xbe,0xe1,0x4f}, {0xbd,0xb0,0xe8,0x42}
{0xd4,0xea,0x9f,0x09}, {0xdf,0xe4,0x96,0x04}, {0xc2,0xf6,0x8d,0x13}, {0xc9,0xf8,0x84,0x1e}
{0xf8,0xd2,0xbb,0x3d}, {0xf3,0xdc,0xb2,0x30}, {0xee,0xce,0xa9,0x27}, {0xe5,0xc0,0xa0,0x2a}
{0x3c,0x7a,0x47,0xb1}, {0x37,0x74,0x4e,0xbc}, {0x2a,0x66,0x55,0xab}, {0x21,0x68,0x5c,0xa6}
{0x10,0x42,0x63,0x85}, {0x1b,0x4c,0x6a,0x88}, {0x06,0x5e,0x71,0x9f}, {0x0d,0x50,0x78,0x92}
{0x64,0x0a,0x0f,0xd9}, {0x6f,0x04,0x06,0xd4}, {0x72,0x16,0x1d,0xc3}, {0x79,0x18,0x14,0xce}
{0x48,0x32,0x2b,0xed}, {0x43,0x3c,0x22,0xe0}, {0x5e,0x2e,0x39,0xf7}, {0x55,0x20,0x30,0xfa}
{0x01,0xec,0x9a,0xb7}, {0x0a,0xe2,0x93,0xba}, {0x17,0xf0,0x88,0xad}, {0x1c,0xfe,0x81,0xa0}
{0x2d,0xd4,0xbe,0x83}, {0x26,0xda,0xb7,0x8e}, {0x3b,0xc8,0xac,0x99}, {0x30,0xc6,0xa5,0x94}
{0x59,0x9c,0xd2,0xdf}, {0x52,0x92,0xdb,0xd2}, {0x4f,0x80,0xc0,0xc5}, {0x44,0x8e,0xc9,0xc8}
{0x75,0xa4,0xf6,0xeb}, {0x7e,0xaa,0xff,0xe6}, {0x63,0xb8,0xe4,0xf1}, {0x68,0xb6,0xed,0xfc}
{0xb1,0x0c,0x0a,0x67}, {0xba,0x02,0x03,0x6a}, {0xa7,0x10,0x18,0x7d}, {0xac,0x1e,0x11,0x70}
{0x9d,0x34,0x2e,0x53}, {0x96,0x3a,0x27,0x5e}, {0x8b,0x28,0x3c,0x49}, {0x80,0x26,0x35,0x44}
{0xe9,0x7c,0x42,0x0f}, {0xe2,0x72,0x4b,0x02}, {0xff,0x60,0x50,0x15}, {0xf4,0x6e,0x59,0x18}
{0xc5,0x44,0x66,0x3b}, {0xce,0x4a,0x6f,0x36}, {0xd3,0x58,0x74,0x21}, {0xd8,0x56,0x7d,0x2c}
{0x7a,0x37,0xa1,0x0c}, {0x71,0x39,0xa8,0x01}, {0x6c,0x2b,0xb3,0x16}, {0x67,0x25,0xba,0x1b}
{0x56,0x0f,0x85,0x38}, {0x5d,0x01,0x8c,0x35}, {0x40,0x13,0x97,0x22}, {0x4b,0x1d,0x9e,0x2f}
{0x22,0x47,0xe9,0x64}, {0x29,0x49,0xe0,0x69}, {0x34,0x5b,0xfb,0x7e}, {0x3f,0x55,0xf2,0x73}
{0x0e,0x7f,0xcd,0x50}, {0x05,0x71,0xc4,0x5d}, {0x18,0x63,0xdf,0x4a}, {0x13,0x6d,0xd6,0x47}
```

```
,
0xca,0xd7,0x31,0xdc}, {0xc1,0xd9,0x38,0xd1}, {0xdc,0xcb,0x23,0xc6}, {0xd7,0xc5,0x2a,0xcb
0xe6,0xef,0x15,0xe8}, {0xed,0xe1,0x1c,0xe5}, {0xf0,0xf3,0x07,0xf2}, {0xfb,0xfd,0x0e,0xff
0x92,0xa7,0x79,0xb4}, {0x99,0xa9,0x70,0xb9}, {0x84,0xbb,0x6b,0xae}, {0x8f,0xb5,0x62,0xa3
0xbe,0x9f,0x5d,0x80}, {0xb5,0x91,0x54,0x8d}, {0xa8,0x83,0x4f,0x9a}, {0xa3,0x8d,0x46,0x97
;
word8 U3[256][4] = {
0x00,0x00,0x00,0x00}, {0x0d,0x0b,0x0e,0x09}, {0x1a,0x16,0x1c,0x12}, {0x17,0x1d,0x12,0x1b
0x34,0x2c,0x38,0x24}, {0x39,0x27,0x36,0x2d}, {0x2e,0x3a,0x24,0x36}, {0x23,0x31,0x2a,0x3f
0x68,0x58,0x70,0x48}, {0x65,0x53,0x7e,0x41}, {0x72,0x4e,0x6c,0x5a}, {0x7f,0x45,0x62,0x53
0x5c,0x74,0x48,0x6c}, {0x51,0x7f,0x46,0x65}, {0x46,0x62,0x54,0x7e}, {0x4b,0x69,0x5a,0x77
0xd0,0xb0,0xe0,0x90}, {0xdd,0xbb,0xee,0x99}, {0xca,0xa6,0xfc,0x82}, {0xc7,0xad,0xf2,0x8b
0xe4,0x9c,0xd8,0xb4}, {0xe9,0x97,0xd6,0xbd}, {0xfe,0x8a,0xc4,0xa6}, {0xf3,0x81,0xca,0xaf
0xb8,0xe8,0x90,0xd8}, {0xb5,0xe3,0x9e,0xd1}, {0xa2,0xfe,0x8c,0xca}, {0xaf,0xf5,0x82,0xc3
0x8c,0xc4,0xa8,0xfc}, {0x81,0xcf,0xa6,0xf5}, {0x96,0xd2,0xb4,0xee}, {0x9b,0xd9,0xba,0xe7
0xbb,0x7b,0xdb,0x3b}, {0xb6,0x70,0xd5,0x32}, {0xa1,0x6d,0xc7,0x29}, {0xac,0x66,0xc9,0x20
0x8f,0x57,0xe3,0x1f}, {0x82,0x5c,0xed,0x16}, {0x95,0x41,0xff,0x0d}, {0x98,0x4a,0xf1,0x04
0xd3,0x23,0xab,0x73}, {0xde,0x28,0xa5,0x7a}, {0xc9,0x35,0xb7,0x61}, {0xc4,0x3e,0xb9,0x68
0xe7,0x0f,0x93,0x57}, {0xea,0x04,0x9d,0x5e}, {0xfd,0x19,0x8f,0x45}, {0xf0,0x12,0x81,0x4c
0x6b,0xcb,0x3b,0xab}, {0x66,0xc0,0x35,0xa2}, {0x71,0xdd,0x27,0xb9}, {0x7c,0xd6,0x29,0xb0
0x5f,0xe7,0x03,0x8f}, {0x52,0xec,0x0d,0x86}, {0x45,0xf1,0x1f,0x9d}, {0x48,0xfa,0x11,0x94
0x03,0x93,0x4b,0xe3}, {0x0e,0x98,0x45,0xea}, {0x19,0x85,0x57,0xf1}, {0x14,0x8e,0x59,0xf8
0x37,0xbf,0x73,0xc7}, {0x3a,0xb4,0x7d,0xce}, {0x2d,0xa9,0x6f,0xd5}, {0x20,0xa2,0x61,0xdc
0x6d,0xf6,0xad,0x76}, {0x60,0xfd,0xa3,0x7f}, {0x77,0xe0,0xb1,0x64}, {0x7a,0xeb,0xbf,0x6d
0x59,0xda,0x95,0x52}, {0x54,0xd1,0x9b,0x5b}, {0x43,0xcc,0x89,0x40}, {0x4e,0xc7,0x87,0x49
0x05,0xae,0xdd,0x3e}, {0x08,0xa5,0xd3,0x37}, {0x1f,0xb8,0xc1,0x2c}, {0x12,0xb3,0xcf,0x25
0x31,0x82,0xe5,0x1a}, {0x3c,0x89,0xeb,0x13}, {0x2b,0x94,0xf9,0x08}, {0x26,0x9f,0xf7,0x01
0xbd,0x46,0x4d,0xe6}, {0xb0,0x4d,0x43,0xef}, {0xa7,0x50,0x51,0xf4}, {0xaa,0x5b,0x5f,0xfd
0x89,0x6a,0x75,0xc2}, {0x84,0x61,0x7b,0xcb}, {0x93,0x7c,0x69,0xd0}, {0x9e,0x77,0x67,0xd9
0xd5,0x1e,0x3d,0xae}, {0xd8,0x15,0x33,0xa7}, {0xcf,0x08,0x21,0xbc}, {0xc2,0x03,0x2f,0xb5
0xe1,0x32,0x05,0x8a}, {0xec,0x39,0x0b,0x83}, {0xfb,0x24,0x19,0x98}, {0xf6,0x2f,0x17,0x91
0xd6,0x8d,0x76,0x4d}, {0xdb,0x86,0x78,0x44}, {0xcc,0x9b,0x6a,0x5f}, {0xc1,0x90,0x64,0x56
0xe2,0xa1,0x4e,0x69}, {0xef,0xaa,0x40,0x60}, {0xf8,0xb7,0x52,0x7b}, {0xf5,0xbc,0x5c,0x72
0xbe,0xd5,0x06,0x05}, {0xb3,0xde,0x08,0x0c}, {0xa4,0xc3,0x1a,0x17}, {0xa9,0xc8,0x14,0x1e
0x8a,0xf9,0x3e,0x21}, {0x87,0xf2,0x30,0x28}, {0x90,0xef,0x22,0x33}, {0x9d,0xe4,0x2c,0x3a
0x06,0x3d,0x96,0xdd}, {0x0b,0x36,0x98,0xd4}, {0x1c,0x2b,0x8a,0xcf}, {0x11,0x20,0x84,0xc6
0x32,0x11,0xae,0xf9}, {0x3f,0x1a,0xa0,0xf0}, {0x28,0x07,0xb2,0xeb}, {0x25,0x0c,0xbc,0xe2
0x6e,0x65,0xe6,0x95}, {0x63,0x6e,0xe8,0x9c}, {0x74,0x73,0xfa,0x87}, {0x79,0x78,0xf4,0x8e
,
```



```
{0x5a,0x49,0xde,0xb1}, {0x57,0x42,0xd0,0xb8}, {0x40,0x5f,0xc2,0xa3}, {0x4d,0x54,0xcc,0xaa
0xda,0xf7,0x41,0xec}, {0xd7,0xfc,0x4f,0xe5}, {0xc0,0xe1,0x5d,0xfe}, {0xcd,0xea,0x53,0xf7
0xee,0xdb,0x79,0xc8}, {0xe3,0xd0,0x77,0xc1}, {0xf4,0xcd,0x65,0xda}, {0xf9,0xc6,0x6b,0xd3
0xb2,0xaf,0x31,0xa4}, {0xbf,0xa4,0x3f,0xad}, {0xa8,0xb9,0x2d,0xb6}, {0xa5,0xb2,0x23,0xbf
0x86,0x83,0x09,0x80}, {0x8b,0x88,0x07,0x89}, {0x9c,0x95,0x15,0x92}, {0x91,0x9e,0x1b,0x9b
0x0a,0x47,0xa1,0x7c}, {0x07,0x4c,0xaf,0x75}, {0x10,0x51,0xbd,0x6e}, {0x1d,0x5a,0xb3,0x67
0x3e,0x6b,0x99,0x58}, {0x33,0x60,0x97,0x51}, {0x24,0x7d,0x85,0x4a}, {0x29,0x76,0x8b,0x43
0x62,0x1f,0xd1,0x34}, {0x6f,0x14,0xdf,0x3d}, {0x78,0x09,0xcd,0x26}, {0x75,0x02,0xc3,0x2f
0x56,0x33,0xe9,0x10}, {0x5b,0x38,0xe7,0x19}, {0x4c,0x25,0xf5,0x02}, {0x41,0x2e,0xfb,0x0b
0x61,0x8c,0x9a,0xd7}, {0x6c,0x87,0x94,0xde}, {0x7b,0x9a,0x86,0xc5}, {0x76,0x91,0x88,0xcc
0x55,0xa0,0xa2,0xf3}, {0x58,0xab,0xac,0xfa}, {0x4f,0xb6,0xbe,0xe1}, {0x42,0xbd,0xb0,0xe8
0x09,0xd4,0xea,0x9f}, {0x04,0xdf,0xe4,0x96}, {0x13,0xc2,0xf6,0x8d}, {0x1e,0xc9,0xf8,0x84
0x3d,0xf8,0xd2,0xbb}, {0x30,0xf3,0xdc,0xb2}, {0x27,0xee,0xce,0xa9}, {0x2a,0xe5,0xc0,0xa0
0xb1,0x3c,0x7a,0x47}, {0xbc,0x37,0x74,0x4e}, {0xab,0x2a,0x66,0x55}, {0xa6,0x21,0x68,0x5c
0x85,0x10,0x42,0x63}, {0x88,0x1b,0x4c,0x6a}, {0x9f,0x06,0x5e,0x71}, {0x92,0x0d,0x50,0x78
0xd9,0x64,0x0a,0x0f}, {0xd4,0x6f,0x04,0x06}, {0xc3,0x72,0x16,0x1d}, {0xce,0x79,0x18,0x14
0xed,0x48,0x32,0x2b}, {0xe0,0x43,0x3c,0x22}, {0xf7,0x5e,0x2e,0x39}, {0xfa,0x55,0x20,0x30
0xb7,0x01,0xec,0x9a}, {0xba,0x0a,0xe2,0x93}, {0xad,0x17,0xf0,0x88}, {0xa0,0x1c,0xfe,0x81
0x83,0x2d,0xd4,0xbe}, {0x8e,0x26,0xda,0xb7}, {0x99,0x3b,0xc8,0xac}, {0x94,0x30,0xc6,0xa5
0xdf,0x59,0x9c,0xd2}, {0xd2,0x52,0x92,0xdb}, {0xc5,0x4f,0x80,0xc0}, {0xc8,0x44,0x8e,0xc9
0xeb,0x75,0xa4,0xf6}, {0xe6,0x7e,0xaa,0xff}, {0xf1,0x63,0xb8,0xe4}, {0xfc,0x68,0xb6,0xed
0x67,0xb1,0x0c,0x0a}, {0x6a,0xba,0x02,0x03}, {0x7d,0xa7,0x10,0x18}, {0x70,0xac,0x1e,0x11
0x53,0x9d,0x34,0x2e}, {0x5e,0x96,0x3a,0x27}, {0x49,0x8b,0x28,0x3c}, {0x44,0x80,0x26,0x35
0x0f,0xe9,0x7c,0x42}, {0x02,0xe2,0x72,0x4b}, {0x15,0xff,0x60,0x50}, {0x18,0xf4,0x6e,0x59
0x3b,0xc5,0x44,0x66}, {0x36,0xce,0x4a,0x6f}, {0x21,0xd3,0x58,0x74}, {0x2c,0xd8,0x56,0x7d
0x0c,0x7a,0x37,0xa1}, {0x01,0x71,0x39,0xa8}, {0x16,0x6c,0x2b,0xb3}, {0x1b,0x67,0x25,0xba
0x38,0x56,0x0f,0x85}, {0x35,0x5d,0x01,0x8c}, {0x22,0x40,0x13,0x97}, {0x2f,0x4b,0x1d,0x9e
0x64,0x22,0x47,0xe9}, {0x69,0x29,0x49,0xe0}, {0x7e,0x34,0x5b,0xfb}, {0x73,0x3f,0x55,0xf2
0x50,0x0e,0x7f,0xcd}, {0x5d,0x05,0x71,0xc4}, {0x4a,0x18,0x63,0xdf}, {0x47,0x13,0x6d,0xd6
0xdc,0xca,0xd7,0x31}, {0xd1,0xc1,0xd9,0x38}, {0xc6,0xdc,0xcb,0x23}, {0xcb,0xd7,0xc5,0x2a
0xe8,0xe6,0xef,0x15}, {0xe5,0xed,0xe1,0x1c}, {0xf2,0xf0,0xf3,0x07}, {0xff,0xfb,0xfd,0x0e
0xb4,0x92,0xa7,0x79}, {0xb9,0x99,0xa9,0x70}, {0xae,0x84,0xbb,0x6b}, {0xa3,0x8f,0xb5,0x62
0x80,0xbe,0x9f,0x5d}, {0x8d,0xb5,0x91,0x54}, {0x9a,0xa8,0x83,0x4f}, {0x97,0xa3,0x8d,0x46
;
```

```
word8 U4[256][4] = {
0x00,0x00,0x00,0x00}, {0x09,0x0d,0x0b,0x0e}, {0x12,0x1a,0x16,0x1c}, {0x1b,0x17,0x1d,0x12
0x24,0x34,0x2c,0x38}, {0x2d,0x39,0x27,0x36}, {0x36,0x2e,0x3a,0x24}, {0x3f,0x23,0x31,0x2a
0x48,0x68,0x58,0x70}, {0x41,0x65,0x53,0x7e}, {0x5a,0x72,0x4e,0x6c}, {0x53,0x7f,0x45,0x62
```

```
{0x6c,0x5c,0x74,0x48}, {0x65,0x51,0x7f,0x46}, {0x7e,0x46,0x62,0x54}, {0x77,0x4b,0x69,0x5a},
{0x90,0xd0,0xb0,0xe0}, {0x99,0xdd,0xbb,0xee}, {0x82,0xca,0xa6,0xfc}, {0x8b,0xc7,0xad,0xf2},
{0xb4,0xe4,0x9c,0xd8}, {0xbd,0xe9,0x97,0xd6}, {0xa6,0xfe,0x8a,0xc4}, {0xaf,0xf3,0x81,0xca},
{0xd8,0xb8,0xe8,0x90}, {0xd1,0xb5,0xe3,0x9e}, {0xca,0xa2,0xfe,0x8c}, {0xc3,0xaf,0xf5,0x82},
{0xfc,0x8c,0xc4,0xa8}, {0xf5,0x81,0xcf,0xa6}, {0xee,0x96,0xd2,0xb4}, {0xe7,0x9b,0xd9,0xba},
{0x3b,0xbb,0x7b,0xdb}, {0x32,0xb6,0x70,0xd5}, {0x29,0xa1,0x6d,0xc7}, {0x20,0xac,0x66,0xc9},
{0x1f,0x8f,0x57,0xe3}, {0x16,0x82,0x5c,0xed}, {0x0d,0x95,0x41,0xff}, {0x04,0x98,0x4a,0xf1},
{0x73,0xd3,0x23,0xab}, {0x7a,0xde,0x28,0xa5}, {0x61,0xc9,0x35,0xb7}, {0x68,0xc4,0x3e,0xb9},
{0x57,0xe7,0x0f,0x93}, {0x5e,0xea,0x04,0x9d}, {0x45,0xfd,0x19,0x8f}, {0x4c,0xf0,0x12,0x81},
{0xab,0x6b,0xcb,0x3b}, {0xa2,0x66,0xc0,0x35}, {0xb9,0x71,0xdd,0x27}, {0xb0,0x7c,0xd6,0x29},
{0x8f,0x5f,0xe7,0x03}, {0x86,0x52,0xec,0x0d}, {0x9d,0x45,0xf1,0x1f}, {0x94,0x48,0xfa,0x11},
{0xe3,0x03,0x93,0x4b}, {0xea,0x0e,0x98,0x45}, {0xf1,0x19,0x85,0x57}, {0xf8,0x14,0x8e,0x59},
{0xc7,0x37,0xbf,0x73}, {0xce,0x3a,0xb4,0x7d}, {0xd5,0x2d,0xa9,0x6f}, {0xdc,0x20,0xa2,0x61},
{0x76,0x6d,0xf6,0xad}, {0x7f,0x60,0xfd,0xa3}, {0x64,0x77,0xe0,0xb1}, {0x6d,0x7a,0xeb,0xbf},
{0x52,0x59,0xda,0x95}, {0x5b,0x54,0xd1,0x9b}, {0x40,0x43,0xcc,0x89}, {0x49,0x4e,0xc7,0x87},
{0x3e,0x05,0xae,0xdd}, {0x37,0x08,0xa5,0xd3}, {0x2c,0x1f,0xb8,0xc1}, {0x25,0x12,0xb3,0xcf},
{0x1a,0x31,0x82,0xe5}, {0x13,0x3c,0x89,0xeb}, {0x08,0x2b,0x94,0xf9}, {0x01,0x26,0x9f,0xf7},
{0xe6,0xbd,0x46,0x4d}, {0xef,0xb0,0x4d,0x43}, {0xf4,0xa7,0x50,0x51}, {0xfd,0xaa,0x5b,0x5f},
{0xc2,0x89,0x6a,0x75}, {0xcb,0x84,0x61,0x7b}, {0xd0,0x93,0x7c,0x69}, {0xd9,0x9e,0x77,0x67},
{0xae,0xd5,0x1e,0x3d}, {0xa7,0xd8,0x15,0x33}, {0xbc,0xcf,0x08,0x21}, {0xb5,0xc2,0x03,0x2f},
{0x8a,0xe1,0x32,0x05}, {0x83,0xec,0x39,0x0b}, {0x98,0xfb,0x24,0x19}, {0x91,0xf6,0x2f,0x17},
{0x4d,0xd6,0x8d,0x76}, {0x44,0xdb,0x86,0x78}, {0x5f,0xcc,0x9b,0x6a}, {0x56,0xc1,0x90,0x64},
{0x69,0xe2,0xa1,0x4e}, {0x60,0xef,0xaa,0x40}, {0x7b,0xf8,0xb7,0x52}, {0x72,0xf5,0xbc,0x5c},
{0x05,0xbe,0xd5,0x06}, {0x0c,0xb3,0xde,0x08}, {0x17,0xa4,0xc3,0x1a}, {0x1e,0xa9,0xc8,0x14},
{0x21,0x8a,0xf9,0x3e}, {0x28,0x87,0xf2,0x30}, {0x33,0x90,0xef,0x22}, {0x3a,0x9d,0xe4,0x2c},
{0xdd,0x06,0x3d,0x96}, {0xd4,0x0b,0x36,0x98}, {0xcf,0x1c,0x2b,0x8a}, {0xc6,0x11,0x20,0x84},
{0xf9,0x32,0x11,0xae}, {0xf0,0x3f,0x1a,0xa0}, {0xeb,0x28,0x07,0xb2}, {0xe2,0x25,0x0c,0xbc},
{0x95,0x6e,0x65,0xe6}, {0x9c,0x63,0x6e,0xe8}, {0x87,0x74,0x73,0xfa}, {0x8e,0x79,0x78,0xf4},
{0xb1,0x5a,0x49,0xde}, {0xb8,0x57,0x42,0xd0}, {0xa3,0x40,0x5f,0xc2}, {0xaa,0x4d,0x54,0xcc},
{0xec,0xda,0xf7,0x41}, {0xe5,0xd7,0xfc,0x4f}, {0xfe,0xc0,0xe1,0x5d}, {0xf7,0xcd,0xea,0x53},
{0xc8,0xee,0xdb,0x79}, {0xc1,0xe3,0xd0,0x77}, {0xda,0xf4,0xcd,0x65}, {0xd3,0xf9,0xc6,0x6b},
{0xa4,0xb2,0xaf,0x31}, {0xad,0xbf,0xa4,0x3f}, {0xb6,0xa8,0xb9,0x2d}, {0xbf,0xa5,0xb2,0x23},
{0x80,0x86,0x83,0x09}, {0x89,0x8b,0x88,0x07}, {0x92,0x9c,0x95,0x15}, {0x9b,0x91,0x9e,0x1b},
{0x7c,0x0a,0x47,0xa1}, {0x75,0x07,0x4c,0xaf}, {0x6e,0x10,0x51,0xbd}, {0x67,0x1d,0x5a,0xb3},
{0x58,0x3e,0x6b,0x99}, {0x51,0x33,0x60,0x97}, {0x4a,0x24,0x7d,0x85}, {0x43,0x29,0x76,0x8b},
{0x34,0x62,0x1f,0xd1}, {0x3d,0x6f,0x14,0xdf}, {0x26,0x78,0x09,0xcd}, {0x2f,0x75,0x02,0xc3},
{0x10,0x56,0x33,0xe9}, {0x19,0x5b,0x38,0xe7}, {0x02,0x4c,0x25,0xf5}, {0x0b,0x41,0x2e,0xfb}
```

```
,
0xd7,0x61,0x8c,0x9a}, {0xde,0x6c,0x87,0x94}, {0xc5,0x7b,0x9a,0x86}, {0xcc,0x76,0x91,0x88
0xf3,0x55,0xa0,0xa2}, {0xfa,0x58,0xab,0xac}, {0xe1,0x4f,0xb6,0xbe}, {0xe8,0x42,0xbd,0xb0
0x9f,0x09,0xd4,0xea}, {0x96,0x04,0xdf,0xe4}, {0x8d,0x13,0xc2,0xf6}, {0x84,0x1e,0xc9,0xf8
0xbb,0x3d,0xf8,0xd2}, {0xb2,0x30,0xf3,0xdc}, {0xa9,0x27,0xee,0xce}, {0xa0,0x2a,0xe5,0xc0
0x47,0xb1,0x3c,0x7a}, {0x4e,0xbc,0x37,0x74}, {0x55,0xab,0x2a,0x66}, {0x5c,0xa6,0x21,0x68
0x63,0x85,0x10,0x42}, {0x6a,0x88,0x1b,0x4c}, {0x71,0x9f,0x06,0x5e}, {0x78,0x92,0x0d,0x50
0x0f,0xd9,0x64,0x0a}, {0x06,0xd4,0x6f,0x04}, {0x1d,0xc3,0x72,0x16}, {0x14,0xce,0x79,0x18
0x2b,0xed,0x48,0x32}, {0x22,0xe0,0x43,0x3c}, {0x39,0xf7,0x5e,0x2e}, {0x30,0xfa,0x55,0x20
0x9a,0xb7,0x01,0xec}, {0x93,0xba,0x0a,0xe2}, {0x88,0xad,0x17,0xf0}, {0x81,0xa0,0x1c,0xfe
0xbe,0x83,0x2d,0xd4}, {0xb7,0x8e,0x26,0xda}, {0xac,0x99,0x3b,0xc8}, {0xa5,0x94,0x30,0xc6
0xd2,0xdf,0x59,0x9c}, {0xdb,0xd2,0x52,0x92}, {0xc0,0xc5,0x4f,0x80}, {0xc9,0xc8,0x44,0x8e
0xf6,0xeb,0x75,0xa4}, {0xff,0xe6,0x7e,0xaa}, {0xe4,0xf1,0x63,0xb8}, {0xed,0xfc,0x68,0xb6
0x0a,0x67,0xb1,0x0c}, {0x03,0x6a,0xba,0x02}, {0x18,0x7d,0xa7,0x10}, {0x11,0x70,0xac,0x1e
0x2e,0x53,0x9d,0x34}, {0x27,0x5e,0x96,0x3a}, {0x3c,0x49,0x8b,0x28}, {0x35,0x44,0x80,0x26
0x42,0x0f,0xe9,0x7c}, {0x4b,0x02,0xe2,0x72}, {0x50,0x15,0xff,0x60}, {0x59,0x18,0xf4,0x6e
0x66,0x3b,0xc5,0x44}, {0x6f,0x36,0xce,0x4a}, {0x74,0x21,0xd3,0x58}, {0x7d,0x2c,0xd8,0x56
0xa1,0x0c,0x7a,0x37}, {0xa8,0x01,0x71,0x39}, {0xb3,0x16,0x6c,0x2b}, {0xba,0x1b,0x67,0x25
0x85,0x38,0x56,0x0f}, {0x8c,0x35,0x5d,0x01}, {0x97,0x22,0x40,0x13}, {0x9e,0x2f,0x4b,0x1d
0xe9,0x64,0x22,0x47}, {0xe0,0x69,0x29,0x49}, {0xfb,0x7e,0x34,0x5b}, {0xf2,0x73,0x3f,0x55
0xcd,0x50,0x0e,0x7f}, {0xc4,0x5d,0x05,0x71}, {0xdf,0x4a,0x18,0x63}, {0xd6,0x47,0x13,0x6d
0x31,0xdc,0xca,0xd7}, {0x38,0xd1,0xc1,0xd9}, {0x23,0xc6,0xdc,0xcb}, {0x2a,0xcb,0xd7,0xc5
0x15,0xe8,0xe6,0xef}, {0x1c,0xe5,0xed,0xe1}, {0x07,0xf2,0xf0,0xf3}, {0x0e,0xff,0xfb,0xfd
0x79,0xb4,0x92,0xa7}, {0x70,0xb9,0x99,0xa9}, {0x6b,0xae,0x84,0xbb}, {0x62,0xa3,0x8f,0xb5
0x5d,0x80,0xbe,0x9f}, {0x54,0x8d,0xb5,0x91}, {0x4f,0x9a,0xa8,0x83}, {0x46,0x97,0xa3,0x8d
;
```

```
word32 rcon[30] = {
    0x01,0x02, 0x04, 0x08, 0x10, 0x20, 0x40, 0x80, 0x1b, 0x36, 0x6c, 0xd8, 0xab, 0x4d, 0x9a
    , 0x2f, 0x5e, 0xbc, 0x63, 0xc6, 0x97, 0x35, 0x6a, 0xd4, 0xb3, 0x7d, 0xfa, 0xef, 0xc5, 0x9
1
};

/*
 * $PchId: boxes.dat,v 1.2 2001/01/10 21:55:17 philip Exp $
 */
```

```
/*
 * rijndael-alg.h    v2.4    April '2000
 *
 * Optimised ANSI C code
 */

#ifndef __RIJNDAEL_ALG_H
#define __RIJNDAEL_ALG_H

#define MAXKC                (256/32)
#define MAXROUNDS            14

/* Fix me: something generic based on inttypes.h */
#include "word_i386.h"

int rijndael_KeySched(word8 k[MAXKC][4], word8 rk[MAXROUNDS+1][4][4], int ROUNDS);

int rijndael_KeyEncToDec(word8 W[MAXROUNDS+1][4][4], int ROUNDS);

int rijndael_Encrypt(const void *a, void *b, word8 rk[MAXROUNDS+1][4][4], int ROUNDS);

#ifdef INTERMEDIATE_VALUE_KAT
int rijndaelEncryptRound(word8 a[4][4], word8 rk[MAXROUNDS+1][4][4], int ROUNDS, int rounds);
#endif /* INTERMEDIATE_VALUE_KAT */

int rijndael_Decrypt(const void *a, void *b, word8 rk[MAXROUNDS+1][4][4], int ROUNDS);

#ifdef INTERMEDIATE_VALUE_KAT
int rijndaelDecryptRound(word8 a[4][4], word8 rk[MAXROUNDS+1][4][4], int ROUNDS, int rounds);
#endif /* INTERMEDIATE_VALUE_KAT */

#endif /* __RIJNDAEL_ALG_H */

/*
 * $PchId: rijndael-alg.h,v 1.3 2003/09/29 09:19:17 philip Exp $
 */
```

```
/*      rijndael-api.h - Rijndael encryption programming interface.
 *
 *                                          Author: Kees J. Bot
 *                                          3 Nov 2000
 * Heavily based on the original API code by Antoon Bosselaers,
 * Vincent Rijmen, and Paulo Barreto, but with a different interface.
 *
 * This code (.h and .c) is in the public domain.
 */

#ifndef __RIJNDAEL_API_H
#define __RIJNDAEL_API_H

/* Error codes. */
#define RD_BAD_KEY_MAT      -1 /* Key material not of correct length */
#define RD_BAD_BLOCK_LENGTH -2 /* Data is not a block multiple */
#define RD_BAD_DATA        -3 /* Data contents are invalid (bad padding?) */

/* Key information. */
#define RD_KEY_HEX          -1 /* Key is in hex (otherwise octet length) */
#define RD_MAXROUNDS        14 /* Max number of encryption rounds. */

typedef struct {
    int      rounds;          /* Key-length-dependent number of rounds */
    unsigned char encsched[RD_MAXROUNDS+1][4][4]; /* Encr key schedule */
    unsigned char decsched[RD_MAXROUNDS+1][4][4]; /* Decr key schedule */
} rd_keyinstance;

/* Function prototypes. */

int rijndael_makekey(rd_keyinstance *_key,
    size_t _keylen, const void *_keymaterial);

ssize_t rijndael_ecb_encrypt(rd_keyinstance *_key,
    const void *_input, void *_output, size_t _length, void *_dummyIV);

ssize_t rijndael_ecb_decrypt(rd_keyinstance *_key,
    const void *_input, void *_output, size_t _length, void *_dummyIV);

ssize_t rijndael_cbc_encrypt(rd_keyinstance *_key,
    const void *_input, void *_output, size_t _length, void *_IV);

ssize_t rijndael_cbc_decrypt(rd_keyinstance *_key,
    const void *_input, void *_output, size_t _length, void *_IV);

ssize_t rijndael_cfb1_encrypt(rd_keyinstance *_key,
    const void *_input, void *_output, size_t _length, void *_IV);

ssize_t rijndael_cfb1_decrypt(rd_keyinstance *_key,
    const void *_input, void *_output, size_t _length, void *_IV);

ssize_t rijndael_cfb8_encrypt(rd_keyinstance *_key,
    const void *_input, void *_output, size_t _length, void *_IV);

ssize_t rijndael_cfb8_decrypt(rd_keyinstance *_key,
    const void *_input, void *_output, size_t _length, void *_IV);

ssize_t rijndael_pad(void *_input, size_t _length);

ssize_t rijndael_unpad(const void *_input, size_t _length);

typedef ssize_t (*rd_function)(rd_keyinstance *_key,
    const void *_input, void *_output, size_t _length, void *_IV);

#ifdef INTERMEDIATE_VALUE_KAT

void cipherEncryptUpdateRounds(rd_keyinstance *_key,
    const void *_input, void *_output, int rounds);

void cipherDecryptUpdateRounds(rd_keyinstance *_key,
    const void *_input, void *_output, int rounds);

#endif /* INTERMEDIATE_VALUE_KAT */

#endif /* __RIJNDAEL_API_H */
```

```
/*  
 * $PchId: rijndael-api.h,v 1.2 2001/01/10 22:02:21 philip Exp $  
 */
```

```

/*      rijndael-api.h - Rijndael encryption programming interface.
 *
 *                                          Author: Kees J. Bot
 *                                          3 Nov 2000
 * Heavily based on the original API code by Antoon Bosselaers,
 * Vincent Rijmen, and Paulo Barreto, but with a different interface.
 *
 * This code (.h and .c) is in the public domain.
 */

#ifndef _CRYPTO__RIJNDAEL_H
#define _CRYPTO__RIJNDAEL_H

/* Error codes. */
#define RD_BAD_KEY_MAT      -1 /* Key material not of correct length */
#define RD_BAD_BLOCK_LENGTH -2 /* Data is not a block multiple */
#define RD_BAD_DATA        -3 /* Data contents are invalid (bad padding?) */

/* Key information. */
#define RD_KEY_HEX          -1 /* Key is in hex (otherwise octet length) */
#define RD_MAXROUNDS        14 /* Max number of encryption rounds. */

typedef struct {
    int      rounds;          /* Key-length-dependent number of rounds */
    unsigned char encsched[RD_MAXROUNDS+1][4][4]; /* Encr key schedule */
    unsigned char decsched[RD_MAXROUNDS+1][4][4]; /* Decr key schedule */
} rd_keyinstance;

#define AES_BLOCKSIZE      16

/* Function prototypes. */

int rijndael_makekey(rd_keyinstance *_key,
    size_t _keylen, const void *_keymaterial);

ssize_t rijndael_ecb_encrypt(rd_keyinstance *_key,
    const void *_input, void *_output, size_t _length, void *_dummyIV);

ssize_t rijndael_ecb_decrypt(rd_keyinstance *_key,
    const void *_input, void *_output, size_t _length, void *_dummyIV);

ssize_t rijndael_cbc_encrypt(rd_keyinstance *_key,
    const void *_input, void *_output, size_t _length, void *_IV);

ssize_t rijndael_cbc_decrypt(rd_keyinstance *_key,
    const void *_input, void *_output, size_t _length, void *_IV);

ssize_t rijndael_cfb1_encrypt(rd_keyinstance *_key,
    const void *_input, void *_output, size_t _length, void *_IV);

ssize_t rijndael_cfb1_decrypt(rd_keyinstance *_key,
    const void *_input, void *_output, size_t _length, void *_IV);

ssize_t rijndael_cfb8_encrypt(rd_keyinstance *_key,
    const void *_input, void *_output, size_t _length, void *_IV);

ssize_t rijndael_cfb8_decrypt(rd_keyinstance *_key,
    const void *_input, void *_output, size_t _length, void *_IV);

ssize_t rijndael_pad(void *_input, size_t _length);

ssize_t rijndael_unpad(const void *_input, size_t _length);

typedef ssize_t (*rd_function)(rd_keyinstance *_key,
    const void *_input, void *_output, size_t _length, void *_IV);

#ifdef INTERMEDIATE_VALUE_KAT

void cipherEncryptUpdateRounds(rd_keyinstance *_key,
    const void *_input, void *_output, int rounds);

void cipherDecryptUpdateRounds(rd_keyinstance *_key,
    const void *_input, void *_output, int rounds);

#endif /* INTERMEDIATE_VALUE_KAT */

```

```
#endif /* __CRYPTO__RIJNDAEL_H */
```

```
/*  
 * $PchId: rijndael.h,v 1.1 2005/06/01 10:13:45 philip Exp $  
 */
```



```

/*
 * rijndael-alg.c   v2.4   April '2000
 *
 * Optimised ANSI C code
 *
 * authors: v1.0: Antoon Bosselaers
 *          v2.0: Vincent Rijmen, K.U.Leuven
 *          v2.3: Paulo Barreto
 *          v2.4: Vincent Rijmen, K.U.Leuven
 *
 * This code is placed in the public domain.
 */

#include <stdio.h>
#include <stdlib.h>

#include "rijndael-alg.h"
#include "boxes.dat"

int rijndael_KeySched(word8 k[MAXKC][4], word8 W[MAXROUNDS+1][4][4], int ROUNDS) {
    /* Calculate the necessary round keys
     * The number of calculations depends on keyBits and blockBits
     */
    int j, r, t, rconpointer = 0;
    word8 tk[MAXKC][4];
    int KC = ROUNDS - 6;

    for (j = KC-1; j >= 0; j--) {
        *((word32*)tk[j]) = *((word32*)k[j]);
    }
    r = 0;
    t = 0;
    /* copy values into round key array */
    for (j = 0; (j < KC) && (r < ROUNDS + 1); ) {
        for (; (j < KC) && (t < 4); j++, t++) {
            *((word32*)W[r][t]) = *((word32*)tk[j]);
        }
        if (t == 4) {
            r++;
            t = 0;
        }
    }

    while (r < ROUNDS + 1) { /* while not enough round key material calculated */
        /* calculate new values */
        tk[0][0] ^= S[tk[KC-1][1]];
        tk[0][1] ^= S[tk[KC-1][2]];
        tk[0][2] ^= S[tk[KC-1][3]];
        tk[0][3] ^= S[tk[KC-1][0]];
        tk[0][0] ^= rcon[rconpointer++];

        if (KC != 8) {
            for (j = 1; j < KC; j++) {
                *((word32*)tk[j]) ^= *((word32*)tk[j-1]);
            }
        } else {
            for (j = 1; j < KC/2; j++) {
                *((word32*)tk[j]) ^= *((word32*)tk[j-1]);
            }
            tk[KC/2][0] ^= S[tk[KC/2 - 1][0]];
            tk[KC/2][1] ^= S[tk[KC/2 - 1][1]];
            tk[KC/2][2] ^= S[tk[KC/2 - 1][2]];
            tk[KC/2][3] ^= S[tk[KC/2 - 1][3]];
            for (j = KC/2 + 1; j < KC; j++) {
                *((word32*)tk[j]) ^= *((word32*)tk[j-1]);
            }
        }
        /* copy values into round key array */
        for (j = 0; (j < KC) && (r < ROUNDS + 1); ) {
            for (; (j < KC) && (t < 4); j++, t++) {
                *((word32*)W[r][t]) = *((word32*)tk[j]);
            }
            if (t == 4) {

```

```

        r++;
        t = 0;
    }
}
return 0;
}

int rijndael_KeyEncToDec(word8 W[MAXROUNDS+1][4][4], int ROUNDS) {
    int r;
    word8 *w;

    for (r = 1; r < ROUNDS; r++) {
        w = W[r][0];
        *((word32*)w) =
            *((word32*)U1[w[0]])
            ^ *((word32*)U2[w[1]])
            ^ *((word32*)U3[w[2]])
            ^ *((word32*)U4[w[3]]);

        w = W[r][1];
        *((word32*)w) =
            *((word32*)U1[w[0]])
            ^ *((word32*)U2[w[1]])
            ^ *((word32*)U3[w[2]])
            ^ *((word32*)U4[w[3]]);

        w = W[r][2];
        *((word32*)w) =
            *((word32*)U1[w[0]])
            ^ *((word32*)U2[w[1]])
            ^ *((word32*)U3[w[2]])
            ^ *((word32*)U4[w[3]]);

        w = W[r][3];
        *((word32*)w) =
            *((word32*)U1[w[0]])
            ^ *((word32*)U2[w[1]])
            ^ *((word32*)U3[w[2]])
            ^ *((word32*)U4[w[3]]);
    }
    return 0;
}

/**
 * Encrypt a single block.
 */
int rijndael_Encrypt(const void *va, void *vb, word8 rk[MAXROUNDS+1][4][4], int ROUNDS) {
    const word8 *a = va;
    word8 *b = vb;
    int r;
    word8 temp[4][4];

    *((word32*)temp[0]) = *((word32*)(a    )) ^ *((word32*)rk[0][0]);
    *((word32*)temp[1]) = *((word32*)(a+ 4)) ^ *((word32*)rk[0][1]);
    *((word32*)temp[2]) = *((word32*)(a+ 8)) ^ *((word32*)rk[0][2]);
    *((word32*)temp[3]) = *((word32*)(a+12)) ^ *((word32*)rk[0][3]);
    *((word32*)(b    )) = *((word32*)T1[temp[0][0]])
                        ^ *((word32*)T2[temp[1][1]])
                        ^ *((word32*)T3[temp[2][2]])
                        ^ *((word32*)T4[temp[3][3]]);
    *((word32*)(b + 4)) = *((word32*)T1[temp[1][0]])
                        ^ *((word32*)T2[temp[2][1]])
                        ^ *((word32*)T3[temp[3][2]])
                        ^ *((word32*)T4[temp[0][3]]);
    *((word32*)(b + 8)) = *((word32*)T1[temp[2][0]])
                        ^ *((word32*)T2[temp[3][1]])
                        ^ *((word32*)T3[temp[0][2]])
                        ^ *((word32*)T4[temp[1][3]]);
    *((word32*)(b +12)) = *((word32*)T1[temp[3][0]])
                        ^ *((word32*)T2[temp[0][1]])
                        ^ *((word32*)T3[temp[1][2]])
                        ^ *((word32*)T4[temp[2][3]]);

    for (r = 1; r < ROUNDS-1; r++) {

```

```

    *((word32*)temp[0]) = *((word32*)(b    )) ^ *((word32*)rk[r][0]);
    *((word32*)temp[1]) = *((word32*)(b+ 4)) ^ *((word32*)rk[r][1]);
    *((word32*)temp[2]) = *((word32*)(b+ 8)) ^ *((word32*)rk[r][2]);
    *((word32*)temp[3]) = *((word32*)(b+12)) ^ *((word32*)rk[r][3]);

    *((word32*)(b    )) = *((word32*)T1[temp[0][0]])
                          ^ *((word32*)T2[temp[1][1]])
                          ^ *((word32*)T3[temp[2][2]])
                          ^ *((word32*)T4[temp[3][3]]);
    *((word32*)(b + 4)) = *((word32*)T1[temp[1][0]])
                          ^ *((word32*)T2[temp[2][1]])
                          ^ *((word32*)T3[temp[3][2]])
                          ^ *((word32*)T4[temp[0][3]]);
    *((word32*)(b + 8)) = *((word32*)T1[temp[2][0]])
                          ^ *((word32*)T2[temp[3][1]])
                          ^ *((word32*)T3[temp[0][2]])
                          ^ *((word32*)T4[temp[1][3]]);
    *((word32*)(b +12)) = *((word32*)T1[temp[3][0]])
                          ^ *((word32*)T2[temp[0][1]])
                          ^ *((word32*)T3[temp[1][2]])
                          ^ *((word32*)T4[temp[2][3]]);
}
/* last round is special */
*((word32*)temp[0]) = *((word32*)(b    )) ^ *((word32*)rk[ROUNDS-1][0]);
*((word32*)temp[1]) = *((word32*)(b+ 4)) ^ *((word32*)rk[ROUNDS-1][1]);
*((word32*)temp[2]) = *((word32*)(b+ 8)) ^ *((word32*)rk[ROUNDS-1][2]);
*((word32*)temp[3]) = *((word32*)(b+12)) ^ *((word32*)rk[ROUNDS-1][3]);
b[ 0] = T1[temp[0][0]][1];
b[ 1] = T1[temp[1][1]][1];
b[ 2] = T1[temp[2][2]][1];
b[ 3] = T1[temp[3][3]][1];
b[ 4] = T1[temp[1][0]][1];
b[ 5] = T1[temp[2][1]][1];
b[ 6] = T1[temp[3][2]][1];
b[ 7] = T1[temp[0][3]][1];
b[ 8] = T1[temp[2][0]][1];
b[ 9] = T1[temp[3][1]][1];
b[10] = T1[temp[0][2]][1];
b[11] = T1[temp[1][3]][1];
b[12] = T1[temp[3][0]][1];
b[13] = T1[temp[0][1]][1];
b[14] = T1[temp[1][2]][1];
b[15] = T1[temp[2][3]][1];
*((word32*)(b    )) ^= *((word32*)rk[ROUNDS][0]);
*((word32*)(b+ 4)) ^= *((word32*)rk[ROUNDS][1]);
*((word32*)(b+ 8)) ^= *((word32*)rk[ROUNDS][2]);
*((word32*)(b+12)) ^= *((word32*)rk[ROUNDS][3]);

    return 0;
}

#ifdef INTERMEDIATE_VALUE_KAT
/**
 * Encrypt only a certain number of rounds.
 * Only used in the Intermediate Value Known Answer Test.
 */
int rijndaelEncryptRound(word8 a[4][4], word8 rk[MAXROUNDS+1][4][4], int ROUNDS, int rounds) {
    int r;
    word8 temp[4][4];

    /* make number of rounds sane */
    if (rounds > ROUNDS) {
        rounds = ROUNDS;
    }

    *((word32*)a[0]) = *((word32*)a[0]) ^ *((word32*)rk[0][0]);
    *((word32*)a[1]) = *((word32*)a[1]) ^ *((word32*)rk[0][1]);
    *((word32*)a[2]) = *((word32*)a[2]) ^ *((word32*)rk[0][2]);
    *((word32*)a[3]) = *((word32*)a[3]) ^ *((word32*)rk[0][3]);

    for (r = 1; (r <= rounds) && (r < ROUNDS); r++) {
        *((word32*)temp[0]) = *((word32*)T1[a[0][0]])
                              ^ *((word32*)T2[a[1][1]])

```

```

        ^ *((word32*)T3[a[2][2]]);
        ^ *((word32*)T4[a[3][3]]);
        *((word32*)temp[1]) = *((word32*)T1[a[1][0]]);
        ^ *((word32*)T2[a[2][1]]);
        ^ *((word32*)T3[a[3][2]]);
        ^ *((word32*)T4[a[0][3]]);
        *((word32*)temp[2]) = *((word32*)T1[a[2][0]]);
        ^ *((word32*)T2[a[3][1]]);
        ^ *((word32*)T3[a[0][2]]);
        ^ *((word32*)T4[a[1][3]]);
        *((word32*)temp[3]) = *((word32*)T1[a[3][0]]);
        ^ *((word32*)T2[a[0][1]]);
        ^ *((word32*)T3[a[1][2]]);
        ^ *((word32*)T4[a[2][3]]);
        *((word32*)a[0]) = *((word32*)temp[0]) ^ *((word32*)rk[r][0]);
        *((word32*)a[1]) = *((word32*)temp[1]) ^ *((word32*)rk[r][1]);
        *((word32*)a[2]) = *((word32*)temp[2]) ^ *((word32*)rk[r][2]);
        *((word32*)a[3]) = *((word32*)temp[3]) ^ *((word32*)rk[r][3]);
    }
    if (rounds == ROUNDS) {
        /* last round is special */
        temp[0][0] = T1[a[0][0]][1];
        temp[0][1] = T1[a[1][1]][1];
        temp[0][2] = T1[a[2][2]][1];
        temp[0][3] = T1[a[3][3]][1];
        temp[1][0] = T1[a[1][0]][1];
        temp[1][1] = T1[a[2][1]][1];
        temp[1][2] = T1[a[3][2]][1];
        temp[1][3] = T1[a[0][3]][1];
        temp[2][0] = T1[a[2][0]][1];
        temp[2][1] = T1[a[3][1]][1];
        temp[2][2] = T1[a[0][2]][1];
        temp[2][3] = T1[a[1][3]][1];
        temp[3][0] = T1[a[3][0]][1];
        temp[3][1] = T1[a[0][1]][1];
        temp[3][2] = T1[a[1][2]][1];
        temp[3][3] = T1[a[2][3]][1];
        *((word32*)a[0]) = *((word32*)temp[0]) ^ *((word32*)rk[ROUNDS][0]);
        *((word32*)a[1]) = *((word32*)temp[1]) ^ *((word32*)rk[ROUNDS][1]);
        *((word32*)a[2]) = *((word32*)temp[2]) ^ *((word32*)rk[ROUNDS][2]);
        *((word32*)a[3]) = *((word32*)temp[3]) ^ *((word32*)rk[ROUNDS][3]);
    }

    return 0;
}
#endif /* INTERMEDIATE_VALUE_KAT */

/**
 * Decrypt a single block.
 */
int rijndael_Decrypt(const void *va, void *vb, word8 rk[MAXROUNDS+1][4][4], int ROUNDS) {
    const word8 *a = va;
    word8 *b = vb;
    int r;
    word8 temp[4][4];

    *((word32*)temp[0]) = *((word32*)(a    )) ^ *((word32*)rk[ROUNDS][0]);
    *((word32*)temp[1]) = *((word32*)(a+ 4)) ^ *((word32*)rk[ROUNDS][1]);
    *((word32*)temp[2]) = *((word32*)(a+ 8)) ^ *((word32*)rk[ROUNDS][2]);
    *((word32*)temp[3]) = *((word32*)(a+12)) ^ *((word32*)rk[ROUNDS][3]);

    *((word32*)(b    )) = *((word32*)T5[temp[0][0]])
        ^ *((word32*)T6[temp[3][1]])
        ^ *((word32*)T7[temp[2][2]])
        ^ *((word32*)T8[temp[1][3]]);
    *((word32*)(b+ 4)) = *((word32*)T5[temp[1][0]])
        ^ *((word32*)T6[temp[0][1]])
        ^ *((word32*)T7[temp[3][2]])
        ^ *((word32*)T8[temp[2][3]]);
    *((word32*)(b+ 8)) = *((word32*)T5[temp[2][0]])
        ^ *((word32*)T6[temp[1][1]])
        ^ *((word32*)T7[temp[0][2]])
        ^ *((word32*)T8[temp[3][3]]);
    *((word32*)(b+12)) = *((word32*)T5[temp[3][0]])

```

```

    ^ *((word32*)T6[temp[2][1]])
    ^ *((word32*)T7[temp[1][2]])
    ^ *((word32*)T8[temp[0][3]]);
    for (r = ROUNDS-1; r > 1; r--) {
        *((word32*)temp[0]) = *((word32*)(b    )) ^ *((word32*)rk[r][0]);
        *((word32*)temp[1]) = *((word32*)(b+ 4)) ^ *((word32*)rk[r][1]);
        *((word32*)temp[2]) = *((word32*)(b+ 8)) ^ *((word32*)rk[r][2]);
        *((word32*)temp[3]) = *((word32*)(b+12)) ^ *((word32*)rk[r][3]);
        *((word32*)(b    )) = *((word32*)T5[temp[0][0]])
    }
    ^ *((word32*)T6[temp[3][1]])
    ^ *((word32*)T7[temp[2][2]])
    ^ *((word32*)T8[temp[1][3]]);
    *((word32*)(b+ 4)) = *((word32*)T5[temp[1][0]])
    ^ *((word32*)T6[temp[0][1]])
    ^ *((word32*)T7[temp[3][2]])
    ^ *((word32*)T8[temp[2][3]]);
    *((word32*)(b+ 8)) = *((word32*)T5[temp[2][0]])
    ^ *((word32*)T6[temp[1][1]])
    ^ *((word32*)T7[temp[0][2]])
    ^ *((word32*)T8[temp[3][3]]);
    *((word32*)(b+12)) = *((word32*)T5[temp[3][0]])
    ^ *((word32*)T6[temp[2][1]])
    ^ *((word32*)T7[temp[1][2]])
    ^ *((word32*)T8[temp[0][3]]);
}
/* last round is special */
*((word32*)temp[0]) = *((word32*)(b    )) ^ *((word32*)rk[1][0]);
*((word32*)temp[1]) = *((word32*)(b+ 4)) ^ *((word32*)rk[1][1]);
*((word32*)temp[2]) = *((word32*)(b+ 8)) ^ *((word32*)rk[1][2]);
*((word32*)temp[3]) = *((word32*)(b+12)) ^ *((word32*)rk[1][3]);
b[ 0] = S5[temp[0][0]];
b[ 1] = S5[temp[3][1]];
b[ 2] = S5[temp[2][2]];
b[ 3] = S5[temp[1][3]];
b[ 4] = S5[temp[1][0]];
b[ 5] = S5[temp[0][1]];
b[ 6] = S5[temp[3][2]];
b[ 7] = S5[temp[2][3]];
b[ 8] = S5[temp[2][0]];
b[ 9] = S5[temp[1][1]];
b[10] = S5[temp[0][2]];
b[11] = S5[temp[3][3]];
b[12] = S5[temp[3][0]];
b[13] = S5[temp[2][1]];
b[14] = S5[temp[1][2]];
b[15] = S5[temp[0][3]];
*((word32*)(b    )) ^= *((word32*)rk[0][0]);
*((word32*)(b+ 4)) ^= *((word32*)rk[0][1]);
*((word32*)(b+ 8)) ^= *((word32*)rk[0][2]);
*((word32*)(b+12)) ^= *((word32*)rk[0][3]);

    return 0;
}

#ifdef INTERMEDIATE_VALUE_KAT
/**
 * Decrypt only a certain number of rounds.
 * Only used in the Intermediate Value Known Answer Test.
 * Operations rearranged such that the intermediate values
 * of decryption correspond with the intermediate values
 * of encryption.
 */
int rijndaelDecryptRound(word8 a[4][4], word8 rk[MAXROUNDS+1][4][4], int ROUNDS, int rounds) {
    int r, i;
    word8 temp[4], shift;

    /* make number of rounds sane */
    if (rounds > ROUNDS) {
        rounds = ROUNDS;
    }
    /* first round is special: */
    *((word32 *)a[0]) ^= *((word32 *)rk[ROUNDS][0]);
    *((word32 *)a[1]) ^= *((word32 *)rk[ROUNDS][1]);

```

```

*(word32 *)a[2] ^= *(word32 *)rk[ROUNDS][2];
*(word32 *)a[3] ^= *(word32 *)rk[ROUNDS][3];
for (i = 0; i < 4; i++) {
    a[i][0] = Si[a[i][0]];
    a[i][1] = Si[a[i][1]];
    a[i][2] = Si[a[i][2]];
    a[i][3] = Si[a[i][3]];
}
for (i = 1; i < 4; i++) {
    shift = (4 - i) & 3;
    temp[0] = a[(0 + shift) & 3][i];
    temp[1] = a[(1 + shift) & 3][i];
    temp[2] = a[(2 + shift) & 3][i];
    temp[3] = a[(3 + shift) & 3][i];
    a[0][i] = temp[0];
    a[1][i] = temp[1];
    a[2][i] = temp[2];
    a[3][i] = temp[3];
}
/* ROUNDS-1 ordinary rounds */
for (r = ROUNDS-1; r > rounds; r--) {
    *(word32 *)a[0] ^= *(word32 *)rk[r][0];
    *(word32 *)a[1] ^= *(word32 *)rk[r][1];
    *(word32 *)a[2] ^= *(word32 *)rk[r][2];
    *(word32 *)a[3] ^= *(word32 *)rk[r][3];

    *((word32*)a[0]) =
        *((word32*)U1[a[0][0]])
        ^ *((word32*)U2[a[0][1]])
        ^ *((word32*)U3[a[0][2]])
        ^ *((word32*)U4[a[0][3]]);

    *((word32*)a[1]) =
        *((word32*)U1[a[1][0]])
        ^ *((word32*)U2[a[1][1]])
        ^ *((word32*)U3[a[1][2]])
        ^ *((word32*)U4[a[1][3]]);

    *((word32*)a[2]) =
        *((word32*)U1[a[2][0]])
        ^ *((word32*)U2[a[2][1]])
        ^ *((word32*)U3[a[2][2]])
        ^ *((word32*)U4[a[2][3]]);

    *((word32*)a[3]) =
        *((word32*)U1[a[3][0]])
        ^ *((word32*)U2[a[3][1]])
        ^ *((word32*)U3[a[3][2]])
        ^ *((word32*)U4[a[3][3]]);
    for (i = 0; i < 4; i++) {
        a[i][0] = Si[a[i][0]];
        a[i][1] = Si[a[i][1]];
        a[i][2] = Si[a[i][2]];
        a[i][3] = Si[a[i][3]];
    }
    for (i = 1; i < 4; i++) {
        shift = (4 - i) & 3;
        temp[0] = a[(0 + shift) & 3][i];
        temp[1] = a[(1 + shift) & 3][i];
        temp[2] = a[(2 + shift) & 3][i];
        temp[3] = a[(3 + shift) & 3][i];
        a[0][i] = temp[0];
        a[1][i] = temp[1];
        a[2][i] = temp[2];
        a[3][i] = temp[3];
    }
}
if (rounds == 0) {
    /* End with the extra key addition */
    *(word32 *)a[0] ^= *(word32 *)rk[0][0];
    *(word32 *)a[1] ^= *(word32 *)rk[0][1];
    *(word32 *)a[2] ^= *(word32 *)rk[0][2];
    *(word32 *)a[3] ^= *(word32 *)rk[0][3];
}

```

```
        return 0;
    }
#endif /* INTERMEDIATE_VALUE_KAT */

/*
 * $PchId: rijndael_alg.c,v 1.2 2001/01/10 21:57:12 philip Exp $
 */
```

```

/*      rijndael-api.c - Rijndael encryption programming interface.
 *
 *                                     Author: Kees J. Bot
 *                                     3 Nov 2000
 * Heavily based on the original API code by Antoon Bosselaers,
 * Vincent Rijmen, and Paulo Barreto, but with a different interface.
 *
 * Read this code top to bottom, not all comments are repeated.
 */

#include <stdlib.h>
#include <string.h>
#include <sys/types.h>

#include "rijndael-alg.h"
#include "rijndael-api.h"

/* Map a byte (?) address to a word address or vv. */
#define W(a)      ((word32 *) (a))
#define B(a)      ((word8 *) (a))

#if STRICT_ALIGN
/* This machine checks alignment religiously. (The code is not proper with
 * respect to alignment. We need a compiler that doesn't muck about with byte
 * arrays that follow words in structs, and that places automatic variables
 * at word boundaries if not odd-sized. Most compilers are this nice.)
 */

#define aligned(a)          (((unsigned) (a) & 3) == 0)
#define aligned2(a1, a2)   aligned((unsigned) (a1) | (unsigned) (a2))

static void blockcpy(void *dst, const void *src)
{
    int i= 0;

    do {
        B(dst)[i+0] = B(src)[i+0];
        B(dst)[i+1] = B(src)[i+1];
        B(dst)[i+2] = B(src)[i+2];
        B(dst)[i+3] = B(src)[i+3];
    } while ((i += 4) < 16);
}

#else /* !STRICT_ALIGN */
/* This machine doesn't mind misaligned accesses much. */

#define aligned(a)          ((void) (a), 1)
#define aligned2(a1, a2)   ((void) (a1), (void) (a2), 1)

#if __GNUC__
__inline
#endif
static void blockcpy(void *dst, const void *src)
{
    W(dst)[0] = W(src)[0];
    W(dst)[1] = W(src)[1];
    W(dst)[2] = W(src)[2];
    W(dst)[3] = W(src)[3];
}

#endif /* !STRICT_ALIGN */

#define between(a, c, z)      (((unsigned) (c) - (a) <= (unsigned) (z) - (a))

int rijndael_makekey(rd_keyinstance *key,
                    size_t keylen, const void *keymaterial)
{
    word8 k[MAXKC][4];

    /* Initialize key schedule: */
    if (keylen == RD_KEY_HEX) {
        const word8 *kp;
        int c, b;

        kp= keymaterial;

```



```

    keylen= 0;

    for (;;) {
        c= *kp++;
        if (between('0', c, '9')) b= (c - '0' + 0x0) << 4;
        else
            if (between('a', c, 'f')) b= (c - 'a' + 0xa) << 4;
        else
            if (between('A', c, 'F')) b= (c - 'A' + 0xA) << 4;
        else break;

        c= *kp++;
        if (between('0', c, '9')) b |= (c - '0' + 0x0);
        else
            if (between('a', c, 'f')) b |= (c - 'a' + 0xa);
        else
            if (between('A', c, 'F')) b |= (c - 'A' + 0xA);
        else break;

        if (keylen >= 256/8) return RD_BAD_KEY_MAT;
        B(k)[keylen++] = b;
    }
    if (c != 0) return RD_BAD_KEY_MAT;

    if (keylen != 128/8 && keylen != 192/8 && keylen != 256/8) {
        return RD_BAD_KEY_MAT;
    }
} else {
    if (keylen != 128/8 && keylen != 192/8 && keylen != 256/8) {
        return RD_BAD_KEY_MAT;
    }
    memcpy(k, keymaterial, keylen);
}

key->rounds= keylen * 8 / 32 + 6;

rijndael_KeySched(k, key->encsched, key->rounds);
memcpy(key->decsched, key->encsched, sizeof(key->decsched));
rijndael_KeyEncToDec(key->decsched, key->rounds);

return 0;
}

ssize_t rijndael_ecb_encrypt(rd_keyinstance *key,
    const void *input, void *output, size_t length, void *dummyIV)
{
    /* Encrypt blocks of data in Electronic Codebook mode. */
    const word8 *inp= input;
    word8 *outp= output;
    size_t i, nr_blocks, extra;
    word32 in[4], out[4];
    word8 t;

    /* Compute the number of whole blocks, and the extra bytes beyond the
     * last block. Those extra bytes, if any, are encrypted by stealing
     * enough bytes from the previous encrypted block to make a whole block.
     * This is done by encrypting the last block, exchanging the first few
     * encrypted bytes with the extra bytes, and encrypting the last whole
     * block again.
     */
    nr_blocks= length / 16;
    if ((extra= (length % 16)) > 0) {
        if (nr_blocks == 0) return RD_BAD_BLOCK_LENGTH;
        nr_blocks--;
    }

    /* Encrypt a number of blocks. */
    if (aligned2(inp, outp)) {
        for (i= 0; i < nr_blocks; i++) {
            rijndael_Encrypt(inp, outp, key->encsched, key->rounds);
            inp += 16;
            outp += 16;
        }
    } else {

```

```

    for (i= 0; i < nr_blocks; i++) {
        blockcpy(in, inp);
        rijndael_Encrypt(in, out, key->encsched, key->rounds);
        blockcpy(outp, out);
        inp += 16;
        outp += 16;
    }
}

/* Encrypt extra bytes by stealing from the last full block. */
if (extra > 0) {
    blockcpy(in, inp);
    rijndael_Encrypt(in, out, key->encsched, key->rounds);
    for (i= 0; i < extra; i++) {
        t= B(out)[i];
        B(out)[i] = inp[16 + i];
        outp[16 + i] = t;
    }
    rijndael_Encrypt(out, out, key->encsched, key->rounds);
    blockcpy(outp, out);
}
return length;
}

ssize_t rijndael_ecb_decrypt(rd_keyinstance *key,
    const void *input, void *output, size_t length, void *dummyIV)
{
    /* Decrypt blocks of data in Electronic Codebook mode. */
    const word8 *inp= input;
    word8 *outp= output;
    size_t i, nr_blocks, extra;
    word32 in[4], out[4];
    word8 t;

    nr_blocks= length / 16;
    if ((extra= (length % 16)) > 0) {
        if (nr_blocks == 0) return RD_BAD_BLOCK_LENGTH;
        nr_blocks--;
    }

    /* Decrypt a number of blocks. */
    if (aligned2(inp, outp)) {
        for (i= 0; i < nr_blocks; i++) {
            rijndael_Decrypt(inp, outp, key->decsched, key->rounds);
            inp += 16;
            outp += 16;
        }
    } else {
        for (i= 0; i < nr_blocks; i++) {
            blockcpy(in, inp);
            rijndael_Decrypt(in, out, key->decsched, key->rounds);
            blockcpy(outp, out);
            inp += 16;
            outp += 16;
        }
    }

    /* Decrypt extra bytes that stole from the last full block. */
    if (extra > 0) {
        blockcpy(in, inp);
        rijndael_Decrypt(in, out, key->decsched, key->rounds);
        for (i= 0; i < extra; i++) {
            t= B(out)[i];
            B(out)[i] = inp[16 + i];
            outp[16 + i] = t;
        }
        rijndael_Decrypt(out, out, key->decsched, key->rounds);
        blockcpy(outp, out);
    }
    return length;
}

ssize_t rijndael_cbc_encrypt(rd_keyinstance *key,
    const void *input, void *output, size_t length, void *IV)

```

```

{
    /* Encrypt blocks of data in Cypher Block Chaining mode. */
    const word8 *inp= input;
    word8 *outp= output;
    size_t i, nr_blocks, extra;
    word32 in[4], out[4], iv[4], *ivp;
    word8 t;

    nr_blocks= length / 16;
    if ((extra= (length % 16)) > 0) {
        if (nr_blocks == 0) return RD_BAD_BLOCK_LENGTH;
        nr_blocks--;
    }

    /* Each input block is first XORed with the previous encryption result.
     * The "Initialization Vector" is used to XOR the first block with.
     * When done the last crypted block is stored back as the new IV to be
     * used for another call to this function.
     */
    ivp= aligned(IV) ? IV : (blockcpy(iv, IV), iv);

    if (aligned2(inp, outp)) {
        for (i= 0; i < nr_blocks; i++) {
            in[0] = W(inp)[0] ^ ivp[0];
            in[1] = W(inp)[1] ^ ivp[1];
            in[2] = W(inp)[2] ^ ivp[2];
            in[3] = W(inp)[3] ^ ivp[3];
            rijndael_Encrypt(in, outp, key->encsched, key->rounds);
            ivp= W(outp);
            inp += 16;
            outp += 16;
        }
    } else {
        for (i= 0; i < nr_blocks; i++) {
            blockcpy(in, inp);
            in[0] ^= ivp[0];
            in[1] ^= ivp[1];
            in[2] ^= ivp[2];
            in[3] ^= ivp[3];
            rijndael_Encrypt(in, out, key->encsched, key->rounds);
            blockcpy(outp, out);
            ivp= out;
            inp += 16;
            outp += 16;
        }
    }
    if (extra > 0) {
        blockcpy(in, inp);
        in[0] ^= ivp[0];
        in[1] ^= ivp[1];
        in[2] ^= ivp[2];
        in[3] ^= ivp[3];
        rijndael_Encrypt(in, out, key->encsched, key->rounds);
        for (i= 0; i < extra; i++) {
            t= B(out)[i];
            B(out)[i] ^= inp[16 + i];
            outp[16 + i] = t;
        }
        rijndael_Encrypt(out, out, key->encsched, key->rounds);
        blockcpy(outp, out);
        ivp= out;
    }
    blockcpy(IV, ivp);          /* Store last IV back. */
    return length;
}

ssize_t rijndael_cbc_decrypt(rd_keyinstance *key,
    const void *input, void *output, size_t length, void *IV)
{
    /* Decrypt blocks of data in Cypher Block Chaining mode. */
    const word8 *inp= input;
    word8 *outp= output;
    size_t i, nr_blocks, extra;
    word32 in[4], out[4], iv[4];

```

```

word8 t;

nr_blocks= length / 16;
if ((extra= (length % 16)) > 0) {
    if (nr_blocks == 0) return RD_BAD_BLOCK_LENGTH;
    nr_blocks--;
}

blockcpy(iv, IV);

if (aligned2(inp, outp)) {
    for (i= 0; i < nr_blocks; i++) {
        rijndael_Decrypt(inp, out, key->decsched, key->rounds);
        out[0] ^= iv[0];
        out[1] ^= iv[1];
        out[2] ^= iv[2];
        out[3] ^= iv[3];
        iv[0] = W(inp)[0];
        iv[1] = W(inp)[1];
        iv[2] = W(inp)[2];
        iv[3] = W(inp)[3];
        W(outp)[0] = out[0];
        W(outp)[1] = out[1];
        W(outp)[2] = out[2];
        W(outp)[3] = out[3];
        inp += 16;
        outp += 16;
    }
} else {
    for (i= 0; i < nr_blocks; i++) {
        blockcpy(in, inp);
        rijndael_Decrypt(in, out, key->decsched, key->rounds);
        out[0] ^= iv[0];
        out[1] ^= iv[1];
        out[2] ^= iv[2];
        out[3] ^= iv[3];
        iv[0] = in[0];
        iv[1] = in[1];
        iv[2] = in[2];
        iv[3] = in[3];
        blockcpy(outp, out);
        inp += 16;
        outp += 16;
    }
}
if (extra > 0) {
    blockcpy(in, inp);
    blockcpy(IV, in);
    rijndael_Decrypt(in, out, key->decsched, key->rounds);
    for (i= 0; i < extra; i++) {
        t= B(out)[i] ^ inp[16 + i];
        B(out)[i] = inp[16 + i];
        outp[16 + i] = t;
    }
    rijndael_Decrypt(out, out, key->decsched, key->rounds);
    out[0] ^= iv[0];
    out[1] ^= iv[1];
    out[2] ^= iv[2];
    out[3] ^= iv[3];
    blockcpy(outp, out);
} else {
    blockcpy(IV, iv);
}
return length;
}

ssize_t rijndael_cfb1_encrypt(rd_keyinstance *key,
    const void *input, void *output, size_t length, void *IV)
{
    /* Encrypt blocks of data in Cypher Feedback mode, 1 bit at a time. */
    const word8 *inp= input;
    word8 *outp= output;
    word8 t;
    size_t i;

```

```

int b;
word32 iv[4], civ[4];

blockcpy(iv, IV);

for (i= 0; i < length; i++) {
    t= *inp++;
    for (b= 0; b < 8; b++) {
        rijndael_Encrypt(iv, civ, key->encsched, key->rounds);
        t ^= (B(civ)[0] & 0x80) >> b;
        B(iv)[ 0] = (B(iv)[ 0] << 1) | (B(iv)[ 1] >> 7);
        B(iv)[ 1] = (B(iv)[ 1] << 1) | (B(iv)[ 2] >> 7);
        B(iv)[ 2] = (B(iv)[ 2] << 1) | (B(iv)[ 3] >> 7);
        B(iv)[ 3] = (B(iv)[ 3] << 1) | (B(iv)[ 4] >> 7);
        B(iv)[ 4] = (B(iv)[ 4] << 1) | (B(iv)[ 5] >> 7);
        B(iv)[ 5] = (B(iv)[ 5] << 1) | (B(iv)[ 6] >> 7);
        B(iv)[ 6] = (B(iv)[ 6] << 1) | (B(iv)[ 7] >> 7);
        B(iv)[ 7] = (B(iv)[ 7] << 1) | (B(iv)[ 8] >> 7);
        B(iv)[ 8] = (B(iv)[ 8] << 1) | (B(iv)[ 9] >> 7);
        B(iv)[ 9] = (B(iv)[ 9] << 1) | (B(iv)[10] >> 7);
        B(iv)[10] = (B(iv)[10] << 1) | (B(iv)[11] >> 7);
        B(iv)[11] = (B(iv)[11] << 1) | (B(iv)[12] >> 7);
        B(iv)[12] = (B(iv)[12] << 1) | (B(iv)[13] >> 7);
        B(iv)[13] = (B(iv)[13] << 1) | (B(iv)[14] >> 7);
        B(iv)[14] = (B(iv)[14] << 1) | (B(iv)[15] >> 7);
        B(iv)[15] = (B(iv)[15] << 1) | ((t >> (7-b)) & 1);
    }
    *outp++ = t;
}
blockcpy(IV, iv);
return length;
}

ssize_t rijndael_cfb1_decrypt(rd_keyinstance *key,
    const void *input, void *output, size_t length, void *IV)
{
    /* Decrypt blocks of data in Cypher Feedback mode, 1 bit at a time. */
    const word8 *inp= input;
    word8 *outp= output;
    word8 t;
    size_t i;
    int b;
    word32 iv[4], civ[4];

    blockcpy(iv, IV);

    for (i= 0; i < length; i++) {
        t= *inp++;
        for (b= 0; b < 8; b++) {
            rijndael_Encrypt(iv, civ, key->encsched, key->rounds);
            B(iv)[ 0] = (B(iv)[ 0] << 1) | (B(iv)[ 1] >> 7);
            B(iv)[ 1] = (B(iv)[ 1] << 1) | (B(iv)[ 2] >> 7);
            B(iv)[ 2] = (B(iv)[ 2] << 1) | (B(iv)[ 3] >> 7);
            B(iv)[ 3] = (B(iv)[ 3] << 1) | (B(iv)[ 4] >> 7);
            B(iv)[ 4] = (B(iv)[ 4] << 1) | (B(iv)[ 5] >> 7);
            B(iv)[ 5] = (B(iv)[ 5] << 1) | (B(iv)[ 6] >> 7);
            B(iv)[ 6] = (B(iv)[ 6] << 1) | (B(iv)[ 7] >> 7);
            B(iv)[ 7] = (B(iv)[ 7] << 1) | (B(iv)[ 8] >> 7);
            B(iv)[ 8] = (B(iv)[ 8] << 1) | (B(iv)[ 9] >> 7);
            B(iv)[ 9] = (B(iv)[ 9] << 1) | (B(iv)[10] >> 7);
            B(iv)[10] = (B(iv)[10] << 1) | (B(iv)[11] >> 7);
            B(iv)[11] = (B(iv)[11] << 1) | (B(iv)[12] >> 7);
            B(iv)[12] = (B(iv)[12] << 1) | (B(iv)[13] >> 7);
            B(iv)[13] = (B(iv)[13] << 1) | (B(iv)[14] >> 7);
            B(iv)[14] = (B(iv)[14] << 1) | (B(iv)[15] >> 7);
            B(iv)[15] = (B(iv)[15] << 1) | ((t >> (7-b)) & 1);
            t ^= (B(civ)[0] & 0x80) >> b;
        }
        *outp++ = t;
    }
    blockcpy(IV, iv);
    return length;
}

```

```
ssize_t rijndael_cfb8_encrypt(rd_keyinstance *key,
    const void *input, void *output, size_t length, void *IV)
{
    /* Encrypt blocks of data in Cypher Feedback mode, 8 bits at a time. */
    const word8 *inp= input;
    word8 *outp= output;
    word8 t;
    size_t i;
    word32 iv[4], civ[4];

    blockcpy(iv, IV);

    for (i= 0; i < length; i++) {
        t= *inp++;
        rijndael_Encrypt(iv, civ, key->encsched, key->rounds);
        t ^= B(civ)[0];
        B(iv)[ 0] = B(iv)[ 1];
        B(iv)[ 1] = B(iv)[ 2];
        B(iv)[ 2] = B(iv)[ 3];
        B(iv)[ 3] = B(iv)[ 4];
        B(iv)[ 4] = B(iv)[ 5];
        B(iv)[ 5] = B(iv)[ 6];
        B(iv)[ 6] = B(iv)[ 7];
        B(iv)[ 7] = B(iv)[ 8];
        B(iv)[ 8] = B(iv)[ 9];
        B(iv)[ 9] = B(iv)[10];
        B(iv)[10] = B(iv)[11];
        B(iv)[11] = B(iv)[12];
        B(iv)[12] = B(iv)[13];
        B(iv)[13] = B(iv)[14];
        B(iv)[14] = B(iv)[15];
        B(iv)[15] = t;
        *outp++ = t;
    }
    blockcpy(IV, iv);
    return length;
}

ssize_t rijndael_cfb8_decrypt(rd_keyinstance *key,
    const void *input, void *output, size_t length, void *IV)
{
    /* Decrypt blocks of data in Cypher Feedback mode, 1 byte at a time. */
    const word8 *inp= input;
    word8 *outp= output;
    word8 t;
    size_t i;
    word32 iv[4], civ[4];

    blockcpy(iv, IV);

    for (i= 0; i < length; i++) {
        t= *inp++;
        rijndael_Encrypt(iv, civ, key->encsched, key->rounds);
        B(iv)[ 0] = B(iv)[ 1];
        B(iv)[ 1] = B(iv)[ 2];
        B(iv)[ 2] = B(iv)[ 3];
        B(iv)[ 3] = B(iv)[ 4];
        B(iv)[ 4] = B(iv)[ 5];
        B(iv)[ 5] = B(iv)[ 6];
        B(iv)[ 6] = B(iv)[ 7];
        B(iv)[ 7] = B(iv)[ 8];
        B(iv)[ 8] = B(iv)[ 9];
        B(iv)[ 9] = B(iv)[10];
        B(iv)[10] = B(iv)[11];
        B(iv)[11] = B(iv)[12];
        B(iv)[12] = B(iv)[13];
        B(iv)[13] = B(iv)[14];
        B(iv)[14] = B(iv)[15];
        B(iv)[15] = t;
        t ^= B(civ)[0];
        *outp++ = t;
    }
    blockcpy(IV, iv);
    return length;
}
```

```

}

ssize_t rijndael_pad(void *input, size_t length)
{
    /* Adds at most one block of RFC-2040 style padding to the input to make
     * it a whole number of blocks for easier encryption. To be used if the
     * input may be less than one block in size, otherwise let the encryption
     * routines use cypher stealing. The input buffer should allow enough
     * space for the padding. The new length of the input is returned.
     */
    word8 *inp= input;
    size_t padlen;

    /* Add padding up until the next block boundary. */
    padlen= 16 - (length % 16);
    memset(inp + length, padlen, padlen);
    return length + padlen;
}

ssize_t rijndael_unpad(const void *input, size_t length)
{
    /* Remove RFC-2040 style padding after decryption. The true length of
     * the input is returned, or the usual errors if the padding is incorrect.
     */
    const word8 *inp= input;
    size_t i, padlen;

    if (length == 0 || (length % 16) != 0) return RD_BAD_BLOCK_LENGTH;
    padlen = inp[length-1];
    if (padlen <= 0 || padlen > 16) return RD_BAD_DATA;
    for (i= 2; i <= padlen; i++) {
        if (inp[length-i] != padlen) return RD_BAD_DATA;
    }
    return length - padlen;
}

#ifdef INTERMEDIATE_VALUE_KAT

void cipherEncryptUpdateRounds(rd_keyinstance *key,
    const void *input, void *output, int rounds)
{
    /* Encrypt a block only a specified number of rounds. */
    word8 block[4][4];

    blockcpy(block, input);

    rijndaelEncryptRound(block, key->encsched, key->rounds, rounds);

    blockcpy(output, block);
}

void cipherDecryptUpdateRounds(rd_keyinstance *key,
    const void *input, void *output, int rounds)
{
    /* Decrypt a block only a specified number of rounds. */
    word8 block[4][4];

    blockcpy(block, input);

    rijndaelDecryptRound(block, key->decsched, key->rounds, rounds);

    blockcpy(output, block);
}

#endif /* INTERMEDIATE_VALUE_KAT */

/*
 * $PchId: rijndael_api.c,v 1.2 2001/01/10 22:01:20 philip Exp $
 */

```

```
typedef unsigned char byte;
typedef unsigned char word8;
typedef unsigned short word16;
typedef unsigned word32;
#define STRICT_ALIGN 1

/*
 * $PchId: word_i386.h,v 1.1 2003/09/29 09:20:13 philip Exp $
 */
```



```
# Makefile for rescue driver
DRIVER = rescue

# directories
u = /usr
i = $u/include
s = $i/sys
m = $i/minix
b = $i/ibm
d = ..

# programs, flags, etc.
MAKE = exec make
CC = exec cc
CFLAGS = -I$i
LDFLAGS = -i
LIBS = -lsys -lsysutil

OBJ = rescue.o
LIBDRIVER = $d/libdriver/driver.o

# build local binary
all build: $(DRIVER)
$(DRIVER): $(OBJ) $(LIBDRIVER)
    $(CC) -o $@ $(LDFLAGS) $(OBJ) $(LIBDRIVER) $(LIBS)
    install -S 1024w $(DRIVER)

$(LIBDRIVER):
    cd $d/libdriver && $(MAKE)

# install with other drivers
install: /sbin/$(DRIVER)
/sbin/$(DRIVER): $(DRIVER)
    install -o root -cs $? $@

# clean up local files
clean:
    rm -f $(DRIVER) *.o *.bak

depend:
    /usr/bin/mkdep "$(CC) -E $(CPPFLAGS)" *.c ../libdriver/*.c > .depend

# Include generated dependencies.
include .depend
```

```

/* This file contains the rescue device driver (/dev/rescue)
 *
 * Changes:
 *      Oct 21, 1992      created (Jorrit N. Herder)
 */

#include "../drivers.h"
#include "../libdriver/driver.h"
#include "../kernel/const.h"
#include "../kernel/config.h"
#include "../kernel/type.h"

#define VERBOSE          0          /* enable/ disable messages */
#define NR_DEVS          1          /* number of rescue devices */
#define RESCUE_KBYTES    128       /* default size in kilobytes */

PRIVATE struct device m_geom[NR_DEVS]; /* base and size of each device */
PRIVATE int m_seg[NR_DEVS]; /* segment index of each device */
PRIVATE int m_device; /* current device */

extern int errno; /* error number for PM calls */

FORWARD _PROTOTYPE( void m_init, (int argc, char **argv) );
FORWARD _PROTOTYPE( char *m_name, (void) );
FORWARD _PROTOTYPE( struct device *m_prepare, (int device) );
FORWARD _PROTOTYPE( int m_transfer, (int proc_nr, int opcode, off_t position,
                                     iovec_t *iov, unsigned nr_req) );
FORWARD _PROTOTYPE( int m_do_open, (struct driver *dp, message *m_ptr) );
FORWARD _PROTOTYPE( void m_geometry, (struct partition *entry) );

/* Entry points to this driver. */
PRIVATE struct driver m_dtab = {
    m_name, /* current device's name */
    m_do_open, /* open or mount */
    do_nop, /* nothing on a close */
    do_diocntl, /* standard I/O controls */
    m_prepare, /* prepare for I/O on a given minor device */
    m_transfer, /* do the I/O */
    nop_cleanup, /* no need to clean up */
    m_geometry, /* memory device "geometry" */
    nop_signal, /* system signals */
    nop_alarm,
    nop_cancel,
    nop_select,
    NULL,
    NULL
};

/*=====
 *                               main                               *
 *=====*/
PUBLIC int main(int argc, char **argv)
{
    /* Main program. Initialize the rescue driver and start the main loop. */
    m_init(argc, argv);
    driver_task(&m_dtab);
    return(OK);
}

/*=====
 *                               m_name                               *
 *=====*/
PRIVATE char *m_name()
{
    /* Return a name for the current device. */
    static char name[] = "rescue";
    return name;
}

/*=====
 *                               m_prepare                               *
 *=====*/
PRIVATE struct device *m_prepare(device)

```

```

int device;
{
/* Prepare for I/O on a device: check if the minor device number is ok. */
if (device < 0 || device >= NR_DEVS) return(NIL_DEV);
m_device = device;

return(&m_geom[device]);
}

/*=====
*                               m_transfer                               *
*=====*/
PRIVATE int m_transfer(proc_nr, opcode, position, iov, nr_req)
int proc_nr;          /* process doing the request */
int opcode;           /* DEV_GATHER or DEV_SCATTER */
off_t position;       /* offset on device to read or write */
iovec_t *iov;         /* pointer to read or write request vector */
unsigned nr_req;      /* length of request vector */
{
/* Read or write one the driver's minor devices. */
int seg;
unsigned count, left, chunk;
vir_bytes user_vir;
struct device *dv;
unsigned long dv_size;
int s;

/* Get and check minor device number. */
if ((unsigned) m_device > NR_DEVS - 1) return(ENXIO);
dv = &m_geom[m_device];
dv_size = cv64ul(dv->dv_size);

while (nr_req > 0) {

/* How much to transfer and where to / from. */
count = iov->iov_size;
user_vir = iov->iov_addr;

/* Virtual copying. For rescue device. */
if (position >= dv_size) return(OK); /* check for EOF */
if (position + count > dv_size) count = dv_size - position;
seg = m_seg[m_device];

if (opcode == DEV_GATHER) { /* copy actual data */
sys_vircopy(SELF,seg,position, proc_nr,D,user_vir, count);
} else {
sys_vircopy(proc_nr,D,user_vir, SELF,seg,position, count);
}

/* Book the number of bytes transferred. */
position += count;
iov->iov_addr += count;
if ((iov->iov_size -= count) == 0) { iov++; nr_req--; }

}
return(OK);
}

/*=====
*                               m_do_open                               *
*=====*/
PRIVATE int m_do_open(dp, m_ptr)
struct driver *dp;
message *m_ptr;
{
/* Check device number on open. */
if (m_prepare(m_ptr->DEVICE) == NIL_DEV) return(ENXIO);
return(OK);
}

/*=====
*                               m_init                               *
*=====*/
PRIVATE void m_init(argc,argv)

```

```

int argc;
char **argv;
{
    /* Initialize this task. All minor devices are initialized one by one. */
    phys_bytes rescue_size;
    phys_bytes rescue_base;
    message m;
    int i, s;

    /* Initialize all rescue devices in a loop. */
    for (i=0; i< NR_DEVS; i++) {

        /* Determine size and base of rescue disks. See if rescue disk details
         * exist in the data store. If no memory for the rescue disk was claimed
         * yet, do it below.
         */
        m.DS_KEY = (RESCUE_MAJOR << 8) + i;
        if (OK == (s = _taskcall(DS_PROC_NR, DS_RETRIEVE, &m))) {
            rescue_size = m.DS_VAL_L1;
            rescue_base = m.DS_VAL_L2;
        }
        else {
            /* no details known */
            if (argc>i+1) rescue_size = atoi(argv[i+1]) * 1024;
            else rescue_size = RESCUE_KBYTES * 1024;

            if (allocmem(rescue_size, &rescue_base) < 0) {
                report("RESCUE", "warning, allocmem failed", errno);
                rescue_size = 0;
            }
        }

        /* Now that we have the base and size of the rescue disk, set up all
         * data structures if the rescue has a positive (nonzero) size.
         */
        if (rescue_size > 0) {

            /* Create a new remote segment to make virtual copies. */
            if (OK != (s=sys_segctl(&m_seg[i], (ul6_t *) &s,
                (vir_bytes *) &s, rescue_base, rescue_size))) {
                panic("RESCUE", "Couldn't install remote segment.", s);
            }

            /* Set the device geometry for the outside world. */
            m_geom[i].dv_base = cvul64(rescue_base);
            m_geom[i].dv_size = cvul64(rescue_size);

            /* Store the values in the data store for future retrieval. */
            m.DS_KEY = (RESCUE_MAJOR << 8) + i;
            m.DS_VAL_L1 = rescue_size;
            m.DS_VAL_L2 = rescue_base;
            if (OK != (s = _taskcall(DS_PROC_NR, DS_PUBLISH, &m))) {
                panic("RESCUE", "Couldn't store rescue disk details at DS.", s);
            }
        }
    }

    #if VERBOSE
        printf("RESCUE disk %d (size %u/base %u) initialized\n",
            i, rescue_size, rescue_base);
    #endif
}

/*=====
 *                               m_geometry                               *
 *=====*/
PRIVATE void m_geometry(entry)
struct partition *entry;
{
    /* Memory devices don't have a geometry, but the outside world insists. */
    entry->cylinders = div64u(m_geom[m_device].dv_size, SECTOR_SIZE) / (64 * 32);
    entry->heads = 64;
    entry->sectors = 32;
}

```

```
# Makefile for the Realtek RTL8139 ethernet driver (RTL8139)
```

```
DRIVER = rtl8139
```

```
# directories
```

```
u = /usr
```

```
i = $u/include
```

```
s = $i/sys
```

```
m = $i/minix
```

```
b = $i/ibm
```

```
d = ..
```

```
# programs, flags, etc.
```

```
MAKE = exec make
```

```
CC = exec cc
```

```
CFLAGS = -I$i
```

```
LDFLAGS = -i
```

```
LIBS = -lsys -lsysutil -ltimers
```

```
OBJ = rtl8139.o
```

```
# build local binary
```

```
all build: $(DRIVER)
```

```
$(DRIVER): $(OBJ)
```

```
$(CC) -o $@ $(LDFLAGS) $(OBJ) $(LIBS)
```

```
install -S 50kw $(DRIVER)
```

```
# install with other drivers
```

```
install: /usr/sbin/$(DRIVER)
```

```
/usr/sbin/$(DRIVER): $(DRIVER)
```

```
install -o root -cs $? $@
```

```
# clean up local files
```

```
clean:
```

```
rm -f $(DRIVER) *.o *.bak
```

```
depend:
```

```
/usr/bin/mkdep "$(CC) -E $(CPPFLAGS)" *.c > .depend
```

```
# Include generated dependencies.
```

```
include .depend
```

```

/*
 * rtl8139.c
 *
 * This file contains a ethernet device driver for Realtek rtl8139 based
 * ethernet cards.
 *
 * The valid messages and their parameters are:
 *
 *      m_type      DL_PORT      DL_PROC      DL_COUNT      DL_MODE      DL_ADDR
 *      /-----+-----+-----+-----+-----+-----/
 *      / HARD_INT  /             /             /             /             /
 *      /-----+-----+-----+-----+-----+-----/
 *      / DL_WRITE  / port nr    / proc nr    / count      / mode       / address    /
 *      /-----+-----+-----+-----+-----+-----/
 *      / DL_WRITEV / port nr    / proc nr    / count      / mode       / address    /
 *      /-----+-----+-----+-----+-----+-----/
 *      / DL_READ   / port nr    / proc nr    / count      /             / address    /
 *      /-----+-----+-----+-----+-----+-----/
 *      / DL_READV  / port nr    / proc nr    / count      /             / address    /
 *      /-----+-----+-----+-----+-----+-----/
 *      / DL_INIT   / port nr    / proc nr    / mode       /             / address    /
 *      /-----+-----+-----+-----+-----+-----/
 *      / DL_GETSTAT / port nr    / proc nr    /             /             / address    /
 *      /-----+-----+-----+-----+-----+-----/
 *      / DL_GETNAME /             /             /             /             /
 *      /-----+-----+-----+-----+-----+-----/
 *      / DL_STOP   / port_nr    /             /             /             /
 *      /-----+-----+-----+-----+-----+-----/
 *
 * The messages sent are:
 *
 *      m-type      DL_POR T      DL_PROC      DL_COUNT      DL_STAT      DL_CLOCK
 *      /-----+-----+-----+-----+-----+-----/
 *      / DL_TASK_REPL / port nr    / proc nr    / rd-count    / err/stat    / clock      /
 *      /-----+-----+-----+-----+-----+-----/
 *
 *      m_type      m3_i1      m3_i2      m3_ca1
 *      /-----+-----+-----+-----+-----/
 *      / DL_INIT_REPL / port nr    / last port    / ethernet addr /
 *      /-----+-----+-----+-----+-----/
 *
 * Created:      Aug 2003 by Philip Homburg <philip@cs.vu.nl>
 * Changes:
 *   Aug 15, 2004  sync alarms replace watchdogs timers  (Jorrit N. Herder)
 *   May 02, 2004  flag alarms replace micro_elapsed()  (Jorrit N. Herder)
 */

#include "../drivers.h"

#include <stdlib.h>
#include <stdio.h>
#include <string.h>
#include <stddef.h>
#include <minix/com.h>
#include <minix/keymap.h>
#include <minix/syslib.h>
#include <minix/type.h>
#include <minix/sysutil.h>
#include <timers.h>
#include <ibm/portio.h>
#include <net/hton.h>
#include <net/gen/ether.h>
#include <net/gen/eth_io.h>
#include <ibm/pci.h>

#include <sys/types.h>
#include <fcntl.h>
#include <assert.h>
#include <unistd.h>
#include <sys/ioc_memory.h>
#include "../kernel/const.h"
#include "../kernel/config.h"
#include "../kernel/type.h"

```

```

#define tmra_ut          timer_t
#define tmra_inittimer(tp) tmr_inittimer(tp)
#define Proc_number(p)   proc_number(p)
#define debug            0
#define printW()         ((void)0)
#define vm_lphys2bus(p)  (p)

#define VERBOSE          1          /* display message during init */

#include "rtl8139.h"

#define RX_BUFSIZE       RL_RCR_RBLLEN_64K_SIZE
#define RX_BUFBITS       RL_RCR_RBLLEN_64K
#define N_TX_BUF         RL_N_TX

#define RE_PORT_NR        1          /* Minix */

/* I/O vectors are handled IOVEC_NR entries at a time. */
#define IOVEC_NR          16

/* Configuration */
#define RL_ENVVAR          "RTLETH"

PRIVATE struct pcitab
{
    u16_t vid;
    u16_t did;
    int checkclass;
} pcitab[] =
{
    { 0x10ec, 0x8139, 0 },          /* Realtek RTL8139 */
    { 0x1186, 0x1300, 0 },          /* D-Link RTL8139 */
    { 0x0000, 0x0000, 0 }
};

typedef struct re
{
    port_t re_base_port;
    int re_irq;
    int re_mode;
    int re_flags;
    int re_client;
    int re_link_up;
    int re_got_int;
    int re_send_int;
    int re_report_link;
    int re_clear_rx;
    int re_need_reset;
    int re_tx_alive;
    char *re_model;

    /* Rx */
    phys_bytes re_rx_buf;
    char *v_re_rx_buf;
    vir_bytes re_read_s;

    /* Tx */
    int re_tx_head;
    int re_tx_tail;
    struct
    {
        int ret_busy;
        phys_bytes ret_buf;
        char *v_ret_buf;
    } re_tx[N_TX_BUF];
    u32_t re_ertxth;          /* Early Tx Threshold */

    /* PCI related */
    int re_seen;              /* TRUE iff device available */
    u8_t re_pcibus;
    u8_t re_pcidev;
    u8_t re_pcifunc;

```

```

    /* 'large' items */
    int re_hook_id;                /* IRQ hook id at kernel */
    eth_stat_t re_stat;
    ether_addr_t re_address;
    message re_rx_mess;
    message re_tx_mess;
    char re_name[sizeof("rtl8139\n")];
    iovect_t re_iovec[IOVEC_NR];
}
re_t;

#define REM_DISABLED      0x0
#define REM_ENABLED      0x1

#define REF_PACK_SENT     0x001
#define REF_PACK_RECV     0x002
#define REF_SEND_AVAIL    0x004
#define REF_READING       0x010
#define REF_EMPTY         0x000
#define REF_PROMISC       0x040
#define REF_MULTI         0x080
#define REF_BROAD         0x100
#define REF_ENABLED       0x200

static re_t re_table[RE_PORT_NR];

static u16_t eth_ign_proto;
static tmra_ut rl_watchdog;

FORWARD _PROTOTYPE( unsigned my_inb, (U16_t port) );
FORWARD _PROTOTYPE( unsigned my_inw, (U16_t port) );
FORWARD _PROTOTYPE( unsigned my_inl, (U16_t port) );
static unsigned my_inb(U16_t port) {
    u32_t value;
    int s;
    if ((s=sys_inb(port, &value)) !=OK)
        printf("RTL8139: warning, sys_inb failed: %d\n", s);
    return value;
}
static unsigned my_inw(U16_t port) {
    u32_t value;
    int s;
    if ((s=sys_inw(port, &value)) !=OK)
        printf("RTL8139: warning, sys_inw failed: %d\n", s);
    return value;
}
static unsigned my_inl(U16_t port) {
    U32_t value;
    int s;
    if ((s=sys_inl(port, &value)) !=OK)
        printf("RTL8139: warning, sys_inl failed: %d\n", s);
    return value;
}
#define rl_inb(port, offset)    (my_inb((port) + (offset)))
#define rl_inw(port, offset)    (my_inw((port) + (offset)))
#define rl_inl(port, offset)    (my_inl((port) + (offset)))

FORWARD _PROTOTYPE( void my_outb, (U16_t port, U8_t value) );
FORWARD _PROTOTYPE( void my_outw, (U16_t port, U16_t value) );
FORWARD _PROTOTYPE( void my_outl, (U16_t port, U32_t value) );
static void my_outb(U16_t port, U8_t value) {
    int s;
    if ((s=sys_outb(port, value)) !=OK)
        printf("RTL8139: warning, sys_outb failed: %d\n", s);
}
static void my_outw(U16_t port, U16_t value) {
    int s;
    if ((s=sys_outw(port, value)) !=OK)
        printf("RTL8139: warning, sys_outw failed: %d\n", s);
}
static void my_outl(U16_t port, U32_t value) {
    int s;
    if ((s=sys_outl(port, value)) !=OK)

```



```

        printf("RTL8139: warning, sys_outl failed: %d\n", s);
    }
#define rl_outb(port, offset, value)    (my_outb((port) + (offset), (value)))
#define rl_outw(port, offset, value)    (my_outw((port) + (offset), (value)))
#define rl_outl(port, offset, value)    (my_outl((port) + (offset), (value)))

_PROTOTYPE( static void sig_handler, (void) ) ;
_PROTOTYPE( static void rl_init, (message *mp) ) ;
_PROTOTYPE( static void rl_pci_conf, (void) ) ;
_PROTOTYPE( static int rl_probe, (re_t *rep) ) ;
_PROTOTYPE( static void rl_conf_hw, (re_t *rep) ) ;
_PROTOTYPE( static void rl_init_buf, (re_t *rep) ) ;
_PROTOTYPE( static void rl_init_hw, (re_t *rep) ) ;
_PROTOTYPE( static void rl_reset_hw, (re_t *rep) ) ;
_PROTOTYPE( static void rl_confaddr, (re_t *rep) ) ;
_PROTOTYPE( static void rl_rec_mode, (re_t *rep) ) ;
_PROTOTYPE( static void rl_readv, (message *mp, int from_int,
                                   int vectored) ) ;
_PROTOTYPE( static void rl_writev, (message *mp, int from_int,
                                   int vectored) ) ;
_PROTOTYPE( static void rl_check_ints, (re_t *rep) ) ;
_PROTOTYPE( static void rl_report_link, (re_t *rep) ) ;
_PROTOTYPE( static void mii_print_techab, (U16_t techab) ) ;
_PROTOTYPE( static void mii_print_stat_speed, (U16_t stat,
                                              U16_t extstat) ) ;
_PROTOTYPE( static void rl_clear_rx, (re_t *rep) ) ;
_PROTOTYPE( static void rl_do_reset, (re_t *rep) ) ;
_PROTOTYPE( static void rl_getstat, (message *mp) ) ;
_PROTOTYPE( static void rl_getname, (message *mp) ) ;
_PROTOTYPE( static void reply, (re_t *rep, int err, int may_block) ) ;
_PROTOTYPE( static void mess_reply, (message *req, message *reply) ) ;
_PROTOTYPE( static void put_userdata, (int user_proc,
                                       vir_bytes user_addr, vir_bytes count, void *loc_addr) ) ;
_PROTOTYPE( static void rtl8139_stop, (void) ) ;
_PROTOTYPE( static void check_int_events, (void) ) ;
_PROTOTYPE( static int do_hard_int, (void) ) ;
_PROTOTYPE( static void rtl8139_dump, (message *m) ) ;
#if 0
_PROTOTYPE( static void dump_phy, (re_t *rep) ) ;
#endif
_PROTOTYPE( static int rl_handler, (re_t *rep) ) ;
_PROTOTYPE( static void rl_watchdog_f, (timer_t *tp) ) ;

/* The message used in the main loop is made global, so that rl_watchdog_f()
 * can change its message type to fake a HARD_INT message.
 */
PRIVATE message m;
PRIVATE int int_event_check;          /* set to TRUE if events arrived */

static char *progname;
extern int errno;

/*=====
 *                               main                               *
 *=====*/
int main(int argc, char *argv[])
{
    int fkeys, sfkeys;
    int inet_proc_nr;
    int i, r;
    re_t *rep;
    long v;

    env_setargs(argc, argv);

    v= 0;
    (void) env_parse("ETH_IGN_PROTO", "x", 0, &v, 0x0000L, 0xFFFFL);
    eth_ign_proto= htons((u16_t) v);

    /* Observe some function key for debug dumps. */
    fkeys = sfkeys = 0; bit_set(sfkeys, 9);
    if ((r=fkey_map(&fkeys, &sfkeys)) != OK)
        printf("Warning: RTL8139 couldn't observe Shift+F9 key: %d\n", r);

```

```

/* Claim buffer memory now under Minix, before MM takes it all. */
for (rep= &re_table[0]; rep < re_table+RE_PORT_NR; rep++)
    rl_init_buf(rep);

/* Try to notify INET that we are present (again). If INET cannot
 * be found, assume this is the first time we started and INET is
 * not yet alive.
 */
(progname=strrchr(argv[0], '/')) ? progname++ : (progname=argv[0]);
r = _pm_findproc("inet", &inet_proc_nr);
if (r == OK) notify(inet_proc_nr);

while (TRUE)
{
    if ((r= receive(ANY, &m)) != OK)
        panic("rtl8139", "receive failed", r);

    switch (m.m_type)
    {
        case DEV_PING: notify(m.m_source);                continue;
        case DL_WRITEV: rl_writev(&m, FALSE, TRUE);       break;
        case DL_WRITE:  rl_writev(&m, FALSE, FALSE);      break;
        case DL_READ:   do_vread(&m, FALSE);              break;
        case DL_READV:  rl_readv(&m, FALSE, TRUE);        break;
        case DL_INIT:   rl_init(&m);                      break;
        case DL_GETSTAT: rl_getstat(&m);                  break;
        case DL_GETNAME: rl_getname(&m);                  break;
        case DL_STOP:   do_stop(&m);                      break;
        case SYN_ALARM:
            /* Under MINIX, synchronous alarms are used instead of
             * watchdog functions. The approach is very different:
             * MINIX VMD timeouts are handled within the kernel
             * (the watchdog is executed by CLOCK), and notify()
             * the driver in some cases.
             * MINIX timeouts result in a SYN_ALARM message to the
             * driver and thus are handled where they should be
             * handled. Locally, watchdog functions are used again.
             */
            rl_watchdog_f(NULL);
            break;
        case HARD_INT:
            do_hard_int();
            if (int_event_check)
                check_int_events();
            break;
        case FKEY_PRESSED: rtl8139_dump(&m);              break;
        case PROC_EVENT:
            sig_handler();
            break;
        default:
            panic("rtl8139", "illegal message", m.m_type);
    }
}

}

/*=====
 *                               sig_handler                               *
 *=====*/
PRIVATE void sig_handler()
{
    sigset_t sigset;
    int sig;

    /* Try to obtain signal set from PM. */
    if (getsigset(&sigset) != 0) return;

    /* Check for known signals. */
    if (sigismember(&sigset, SIGTERM)) {
        rtl8139_stop();
    }
}

```

```

}
}

/*=====
 *                               check_int_events                               *
 *=====*/
static void check_int_events(void)
{
    int i;
    re_t *rep;

    for (i= 0, rep= &re_table[0]; i<RE_PORT_NR; i++, rep++)
    {
        if (rep->re_mode != REM_ENABLED)
            continue;
        if (!rep->re_got_int)
            continue;
        rep->re_got_int= 0;
        assert(rep->re_flags & REF_ENABLED);
        rl_check_ints(rep);
    }
}

/*=====
 *                               rtl8139_stop                               *
 *=====*/
static void rtl8139_stop()
{
    int i;
    re_t *rep;

    for (i= 0, rep= &re_table[0]; i<RE_PORT_NR; i++, rep++)
    {
        if (rep->re_mode != REM_ENABLED)
            continue;
        rl_outb(rep->re_base_port, RL_CR, 0);
    }
    sys_exit(0);
}

/*=====
 *                               rtl8139_dump                               *
 *=====*/
static void rtl8139_dump(m)
message *m;                               /* pointer to request message */
{
    re_t *rep;
    int i;

    printf("\n");
    for (i= 0, rep = &re_table[0]; i<RE_PORT_NR; i++, rep++)
    {
        if (rep->re_mode == REM_DISABLED)
            printf("Realtek RTL 8139 port %d is disabled\n", i);

        if (rep->re_mode != REM_ENABLED)
            continue;

        printf("Realtek RTL 8139 statistics of port %d:\n", i);

        printf("recvErr  :%8ld\t", rep->re_stat.ets_recvErr);
        printf("sendErr   :%8ld\t", rep->re_stat.ets_sendErr);
        printf("OVW      :%8ld\n", rep->re_stat.ets_OVW);

        printf("CRCerr    :%8ld\t", rep->re_stat.ets_CRCerr);
        printf("frameAll  :%8ld\t", rep->re_stat.ets_frameAll);
        printf("missedP   :%8ld\n", rep->re_stat.ets_missedP);

        printf("packetR   :%8ld\t", rep->re_stat.ets_packetR);
        printf("packetT   :%8ld\t", rep->re_stat.ets_packetT);
        printf("transDef  :%8ld\n", rep->re_stat.ets_transDef);

        printf("collision :%8ld\t", rep->re_stat.ets_collision);
        printf("transAb   :%8ld\t", rep->re_stat.ets_transAb);
        printf("carrSense :%8ld\n", rep->re_stat.ets_carrSense);
    }
}

```

```

        printf("fifoUnder :%8ld\t", rep->re_stat.ets_fifoUnder);
        printf("fifoOver :%8ld\t", rep->re_stat.ets_fifoOver);
        printf("CDheartbeat:%8ld\n", rep->re_stat.ets_CDheartbeat);

        printf("OWC      :%8ld\t", rep->re_stat.ets_OWC);

        printf("re_flags = 0x%x\n", rep->re_flags);

        printf(
"TSAD: 0x%04x, TSD: 0x%08x, 0x%08x, 0x%08x, 0x%08x\n",
        rl_inw(rep->re_base_port, RL_TSAD),
        rl_inl(rep->re_base_port, RL_TSD0+0*4),
        rl_inl(rep->re_base_port, RL_TSD0+1*4),
        rl_inl(rep->re_base_port, RL_TSD0+2*4),
        rl_inl(rep->re_base_port, RL_TSD0+3*4));
        printf("tx_head %d, tx_tail %d, busy: %d %d %d %d\n",
        rep->re_tx_head, rep->re_tx_tail,
        rep->re_tx[0].ret_busy, rep->re_tx[1].ret_busy,
        rep->re_tx[2].ret_busy, rep->re_tx[3].ret_busy);
    }
}

/*=====
 *                               do_init                               *
 *=====*/
static void rl_init(mp)
message *mp;
{
    static int first_time= 1;

    int port;
    re_t *rep;
    message reply_mess;

    if (first_time)
    {
        first_time= 0;
        rl_pci_conf(); /* Configure PCI devices. */

        tmra_inittimer(&rl_watchdog);
        /* Use a synchronous alarm instead of a watchdog timer. */
        sys_setalarm(HZ, 0);
    }

    port = mp->DL_PORT;
    if (port < 0 || port >= RE_PORT_NR)
    {
        reply_mess.m_type= DL_INIT_REPLY;
        reply_mess.m3_il= ENXIO;
        mess_reply(mp, &reply_mess);
        return;
    }
    rep= &re_table[port];
    if (rep->re_mode == REM_DISABLED)
    {
        /* This is the default, try to (re)locate the device. */
        rl_conf_hw(rep);
        if (rep->re_mode == REM_DISABLED)
        {
            /* Probe failed, or the device is configured off. */
            reply_mess.m_type= DL_INIT_REPLY;
            reply_mess.m3_il= ENXIO;
            mess_reply(mp, &reply_mess);
            return;
        }
        if (rep->re_mode == REM_ENABLED)
            rl_init_hw(rep);
    }
    #if VERBOSE
    /* load silently ... can always check status later */
    rl_report_link(rep);
    #endif
}

assert(rep->re_mode == REM_ENABLED);

```

```

    assert(rep->re_flags & REF_ENABLED);

    rep->re_flags &= ~(REF_PROMISC | REF_MULTI | REF_BROAD);

    if (mp->DL_MODE & DL_PROMISC_REQ)
        rep->re_flags |= REF_PROMISC;
    if (mp->DL_MODE & DL_MULTI_REQ)
        rep->re_flags |= REF_MULTI;
    if (mp->DL_MODE & DL_BROAD_REQ)
        rep->re_flags |= REF_BROAD;

    rep->re_client = mp->m_source;
    rl_rec_mode(rep);

    reply_mess.m_type = DL_INIT_REPLY;
    reply_mess.m3_i1 = mp->DL_PORT;
    reply_mess.m3_i2 = RE_PORT_NR;
    *(ether_addr_t *) reply_mess.m3_cal = rep->re_address;

    mess_reply(mp, &reply_mess);
}

/*=====
 *                               rl_pci_conf                               *
 *=====*/
static void rl_pci_conf()
{
    int i, h;
    re_t *rep;
    static char envvar[] = RL_ENVVAR "#";
    static char envfmt[] = ":%d.d.d";
    static char val[128];
    long v;

    for (i= 0, rep= re_table; i<RE_PORT_NR; i++, rep++)
    {
        strcpy(rep->re_name, "rtl8139#0");
        rep->re_name[8] += i;
        rep->re_seen= FALSE;
        envvar[sizeof(RL_ENVVAR)-1]= '0'+i;
        if (0 == env_get_param(envvar, val, sizeof(val)) &&
            ! env_prefix(envvar, "pci")) {
            env_panic(envvar);
        }
        v= 0;
        (void) env_parse(envvar, envfmt, 1, &v, 0, 255);
        rep->re_pcibus= v;
        v= 0;
        (void) env_parse(envvar, envfmt, 2, &v, 0, 255);
        rep->re_pcidev= v;
        v= 0;
        (void) env_parse(envvar, envfmt, 3, &v, 0, 255);
        rep->re_pcifunc= v;
    }

    pci_init();

    for (h= 1; h >= 0; h--) {
        for (i= 0, rep= re_table; i<RE_PORT_NR; i++, rep++)
        {
            if (((rep->re_pcibus | rep->re_pcidev |
                rep->re_pcifunc) != 0) != h)
            {
                continue;
            }
            if (rl_probe(rep))
                rep->re_seen= TRUE;
        }
    }
}

/*=====
 *                               rl_probe                               *
 *=====*/

```

```

static int rl_probe(rep)
re_t *rep;
{
    int i, r, devind, just_one;
    ul6_t vid, did;
    u32_t bar;
    u8_t ilr;
    char *dname;

    if ((rep->re_pcibus | rep->re_pcidev | rep->re_pcifunc) != 0)
    {
        /* Look for specific PCI device */
        r = pci_find_dev(rep->re_pcibus, rep->re_pcidev,
            rep->re_pcifunc, &devind);
        if (r == 0)
        {
            printf("%s: no PCI found at %d.%d.%d\n",
                rep->re_name, rep->re_pcibus,
                rep->re_pcidev, rep->re_pcifunc);
            return 0;
        }
        pci_ids(devind, &vid, &did);
        just_one= TRUE;
    }
    else
    {
        r = pci_first_dev(&devind, &vid, &did);
        if (r == 0)
            return 0;
        just_one= FALSE;
    }

    for(;;)
    {
        for (i= 0; pcitab[i].vid != 0; i++)
        {
            if (pcitab[i].vid != vid)
                continue;
            if (pcitab[i].did != did)
                continue;
            if (pcitab[i].checkclass)
            {
                panic("rtl_probe",
                    "class check not implemented", NO_NUM);
            }
            break;
        }
        if (pcitab[i].vid != 0)
            break;

        if (just_one)
        {
            printf(
                "%s: wrong PCI device (%04x/%04x) found at %d.%d.%d\n",
                rep->re_name, vid, did,
                rep->re_pcibus,
                rep->re_pcidev, rep->re_pcifunc);
            return 0;
        }

        r = pci_next_dev(&devind, &vid, &did);
        if (!r)
            return 0;
    }

    #if VERBOSE    /* stay silent at startup, can always get status later */
        dname= pci_dev_name(vid, did);
        if (!dname)
            dname= "unknown device";
        printf("%s: ", rep->re_name);
        printf("%s (%x/%x) at %s\n", dname, vid, did, pci_slot_name(devind));
    #endif
    pci_reserve(devind);
    /* printf("cr = 0x%x\n", pci_attr_r16(devind, PCI_CR)); */
}

```

```

    bar= pci_attr_r32(devind, PCI_BAR) & 0xffffffff0;
    if (bar < 0x400)
    {
        panic("rtl_probe",
              "base address is not properly configured", NO_NUM);
    }
    rep->re_base_port= bar;

    ilr= pci_attr_r8(devind, PCI_ILR);
    rep->re_irq= ilr;
    if (debug)
    {
        printf("%s: using I/O address 0x%lx, IRQ %d\n",
              rep->re_name, (unsigned long)bar, ilr);
    }

    return TRUE;
}

/*=====
 *                               rl_conf_hw                               *
 *=====*/
static void rl_conf_hw(rep)
re_t *rep;
{
    static eth_stat_t empty_stat = {0, 0, 0, 0, 0, 0      /* ,... */ };

    rep->re_mode= REM_DISABLED;      /* Superfluous */

    if (rep->re_seen)
    {
        /* PCI device is present */
        rep->re_mode= REM_ENABLED;
    }
    if (rep->re_mode != REM_ENABLED)
        return;

    rep->re_flags= REF_EMPTY;
    rep->re_link_up= -1;      /* Unknown */
    rep->re_got_int= 0;
    rep->re_send_int= 0;
    rep->re_report_link= 0;
    rep->re_clear_rx= 0;
    rep->re_need_reset= 0;
    rep->re_tx_alive= 0;
    rep->re_read_s= 0;
    rep->re_tx_head= 0;
    rep->re_tx_tail= 0;
    rep->re_ertxth= RL_TSD_ERTXTH_8;
    rep->re_stat= empty_stat;
}

/*=====
 *                               rl_init_buf                               *
 *=====*/
static void rl_init_buf(rep)
re_t *rep;
{
    size_t rx_bufsize, tx_bufsize, tot_bufsize;
    phys_bytes buf;
    char *mallocbuf;
    static struct memory chunk;
    int fd, s, i, off;

    /* Allocate receive and transmit buffers */
    tx_bufsize= ETH_MAX_PACK_SIZE_TAGGED;
    if (tx_bufsize % 4)
        tx_bufsize += 4-(tx_bufsize % 4);      /* Align */
    rx_bufsize= RX_BUFSIZE;
    tot_bufsize= N_TX_BUF*tx_bufsize + rx_bufsize;

    /* Now try to allocate a kernel memory buffer. */
    chunk.size = tot_bufsize;

```

```

#define BUF_ALIGNMENT (64*1024)

    if(!(mallocbuf = malloc(BUF_ALIGNMENT + tot_bufsize))) {
        panic("RTL8139", "Couldn't allocate kernel buffer", i);
    }

    if(OK != (i = sys_umap(SELF, D, (vir_bytes) mallocbuf, tot_bufsize, &buf))) {
        panic("RTL8139", "Couldn't re-map malloced buffer", i);
    }

    /* click-align malloced buffer. this is what we used to get
     * from kmalloc() too.
     */
    if((off = buf % BUF_ALIGNMENT)) {
        mallocbuf += BUF_ALIGNMENT - off;
        buf += BUF_ALIGNMENT - off;
    }

    for (i= 0; i<N_TX_BUF; i++)
    {
        rep->re_tx[i].ret_buf= buf;
        rep->re_tx[i].v_ret_buf= mallocbuf;
        buf += tx_bufsize;
        mallocbuf += tx_bufsize;
    }
    rep->re_rx_buf= buf;
    rep->v_re_rx_buf= mallocbuf;
}

/*=====
 *                               rl_init_hw                               *
 *=====*/
static void rl_init_hw(rep)
re_t *rep;
{
    int s, i;

    rep->re_flags = REF_EMPTY;
    rep->re_flags |= REF_ENABLED;

    /* Set the interrupt handler. The policy is to only send HARD_INT
     * notifications. Don't reenale interrupts automatically. The id
     * that is passed back is the interrupt line number.
     */
    rep->re_hook_id = rep->re_irq;
    if ((s=sys_irqsetpolicy(rep->re_irq, 0, &rep->re_hook_id)) != OK)
        printf("RTL8139: error, couldn't set IRQ policy: %d\n", s);

    rl_reset_hw(rep);

    if ((s=sys_irgenable(&rep->re_hook_id)) != OK)
        printf("RTL8139: error, couldn't enable interrupts: %d\n", s);

#if VERBOSE    /* stay silent during startup, can always get status later */
    if (rep->re_model) {
        printf("%s: model %s\n", rep->re_name, rep->re_model);
    } else
    {
        printf("%s: unknown model 0x%08x\n",
            rep->re_name,
            rl_inl(rep->re_base_port, RL_TCR) &
            (RL_TCR_HWVER_AM | RL_TCR_HWVER_BM));
    }
#endif

    rl_confaddr(rep);
    if (debug)
    {
        printf("%s: Ethernet address ", rep->re_name);
        for (i= 0; i < 6; i++)
        {
            printf("%x%c", rep->re_address.ea_addr[i],
                i < 5 ? ':' : '\n');
        }
    }
}

```



```

    }
}

/*=====
 *                               rl_reset_hw                               *
 *=====*/
static void rl_reset_hw(rep)
re_t *rep;
{
    port_t port;
    u32_t t;
    phys_bytes bus_buf;
    int i;
    clock_t t0,t1;

    port= rep->re_base_port;

#if 0
    /* Reset the PHY */
    rl_outb(port, RL_BMCR, MII_CTRL_RST);
    getuptime(&t0);
    do {
        if (!(rl_inb(port, RL_BMCR) & MII_CTRL_RST))
            break;
    } while (getuptime(&t1)==OK && (t1-t0) < HZ);
    if (rl_inb(port, RL_BMCR) & MII_CTRL_RST)
        panic("rtl8139", "reset PHY failed to complete", NO_NUM);
#endif

    /* Reset the device */
    printf("rl_reset_hw: (before reset) port = 0x%x, RL_CR = 0x%x\n",
        port, rl_inb(port, RL_CR));
    rl_outb(port, RL_CR, RL_CR_RST);
    getuptime(&t0);
    do {
        if (!(rl_inb(port, RL_CR) & RL_CR_RST))
            break;
    } while (getuptime(&t1)==OK && (t1-t0) < HZ);
    printf("rl_reset_hw: (after reset) port = 0x%x, RL_CR = 0x%x\n",
        port, rl_inb(port, RL_CR));
    if (rl_inb(port, RL_CR) & RL_CR_RST)
        printf("rtl8139: reset failed to complete");

    t= rl_inl(port, RL_TCR);
    switch(t & (RL_TCR_HWVER_AM | RL_TCR_HWVER_BM))
    {
    case RL_TCR_HWVER_RTL8139: rep->re_model= "RTL8139"; break;
    case RL_TCR_HWVER_RTL8139A: rep->re_model= "RTL8139A"; break;
    case RL_TCR_HWVER_RTL8139AG:
        rep->re_model= "RTL8139A-G / RTL8139C";
        break;
    case RL_TCR_HWVER_RTL8139B:
        rep->re_model= "RTL8139B / RTL8130";
        break;
    case RL_TCR_HWVER_RTL8100: rep->re_model= "RTL8100"; break;
    case RL_TCR_HWVER_RTL8100B:
        rep->re_model= "RTL8100B/RTL8139D";
        break;
    case RL_TCR_HWVER_RTL8139CP: rep->re_model= "RTL8139C+"; break;
    case RL_TCR_HWVER_RTL8101: rep->re_model= "RTL8101"; break;
    default:
        rep->re_model= NULL;
        break;
    }

#if 0
    printf("REVID: 0x%02x\n", rl_inb(port, RL_REVID));
#endif

    /* Intialize Rx */

    /* Should init multicast mask */
#if 0
    08-0f    R/W        MAR[0-7]        multicast

```

```

#endif
    bus_buf= vm_lphys2bus(rep->re_rx_buf);
    rl_outl(port, RL_RBSTART, bus_buf);

    /* Initialize Tx */
    for (i= 0; i<N_TX_BUF; i++)
    {
        rep->re_tx[i].ret_busy= FALSE;
        bus_buf= vm_lphys2bus(rep->re_tx[i].ret_buf);
        rl_outl(port, RL_TSAD0+i*4, bus_buf);
        t= rl_inl(port, RL_TSD0+i*4);
        assert(t & RL_TSD_OWN);
    }

    #if 0
    dump_phy(rep);
    #endif

    t= rl_inw(port, RL_IMR);
    rl_outw(port, RL_IMR, t | (RL_IMR_SERR | RL_IMR_TIMEOUT |
        RL_IMR_LENCHG));

    t= rl_inw(port, RL_IMR);
    rl_outw(port, RL_IMR, t | (RL_IMR_FOVW | RL_IMR_PUN |
        RL_IMR_RXOVW | RL_IMR_RER | RL_IMR_ROK));

    t= rl_inw(port, RL_IMR);
    rl_outw(port, RL_IMR, t | (RL_IMR_TER | RL_IMR_TOK));

    t= rl_inb(port, RL_CR);
    rl_outb(port, RL_CR, t | RL_CR_RE);

    t= rl_inb(port, RL_CR);
    rl_outb(port, RL_CR, t | RL_CR_TE);

    rl_outl(port, RL_RCR, RX_BUFBITS);

    t= rl_inl(port, RL_TCR);
    rl_outl(port, RL_TCR, t | RL_TCR_IFG_STD);
}

/*=====
 *                               rl_confaddr                               *
 *=====*/
static void rl_confaddr(rep)
re_t *rep;
{
    static char eakey[]= RL_ENVVAR "#_EA";
    static char eafmt[]= "x:x:x:x:x";

    int i;
    port_t port;
    u32_t w;
    long v;

    /* User defined ethernet address? */
    eakey[sizeof(RL_ENVVAR)-1]= '0' + (rep-re_table);

    port= rep->re_base_port;

    for (i= 0; i < 6; i++)
    {
        if (env_parse(eakey, eafmt, i, &v, 0x00L, 0xFFL) != EP_SET)
            break;
        rep->re_address.ea_addr[i]= v;
    }

    if (i != 0 && i != 6) env_panic(eakey); /* It's all or nothing */

    /* Should update ethernet address in hardware */
    if (i == 6)
    {
        port= rep->re_base_port;
        rl_outb(port, RL_9346CR, RL_9346CR_EEM_CONFIG);
    }
}

```

```

        w= 0;
        for (i= 0; i<4; i++)
            w |= (rep->re_address.ea_addr[i] << (i*8));
        rl_outl(port, RL_IDR, w);
        w= 0;
        for (i= 4; i<6; i++)
            w |= (rep->re_address.ea_addr[i] << ((i-4)*8));
        rl_outl(port, RL_IDR+4, w);
        rl_outb(port, RL_9346CR, RL_9346CR_EEM_NORMAL);
    }

    /* Get ethernet address */
    for (i= 0; i<6; i++)
        rep->re_address.ea_addr[i]= rl_inb(port, RL_IDR+i);
}

/*=====
 *                               rl_rec_mode                               *
 *=====*/
static void rl_rec_mode(rep)
re_t *rep;
{
    port_t port;
    u32_t rcr;

    port= rep->re_base_port;
    rcr= rl_inl(port, RL_RCR);
    rcr &= ~(RL_RCR_AB|RL_RCR_AM|RL_RCR_APM|RL_RCR_AAP);
    if (rep->re_flags & REF_PROMISC)
        rcr |= RL_RCR_AB | RL_RCR_AM | RL_RCR_AAP;
    if (rep->re_flags & REF_BROAD)
        rcr |= RL_RCR_AB;
    if (rep->re_flags & REF_MULTI)
        rcr |= RL_RCR_AM;
    rcr |= RL_RCR_APM;

    rl_outl(port, RL_RCR, rcr);
}

/*=====
 *                               rl_readv                               *
 *=====*/
static void rl_readv(mp, from_int, vectored)
message *mp;
int from_int;
int vectored;
{
    int i, j, n, o, s, sl, dl_port, re_client, count, size;
    port_t port;
    unsigned amount, totlen, packlen;
    phys_bytes src_phys, dst_phys, iov_src;
    u16_t d_start, d_end;
    u32_t l, rxstat = 0x12345678;
    re_t *rep;
    iovec_t *iovp;
    int cps;

    dl_port = mp->DL_PORT;
    count = mp->DL_COUNT;
    if (dl_port < 0 || dl_port >= RE_PORT_NR)
        panic("rtl8139", "illegal port", dl_port);
    rep= &re_table[dl_port];
    re_client= mp->DL_PROC;
    rep->re_client= re_client;

    if (rep->re_clear_rx)
        goto suspend;    /* Buffer overflow */

    assert(rep->re_mode == REM_ENABLED);
    assert(rep->re_flags & REF_ENABLED);

    port= rep->re_base_port;

    /* Assume that the RL_CR_BUFEE check was been done by rl_checks_ints

```

```
    /*
    if (!from_int && (rl_inb(port, RL_CR) & RL_CR_BUFE))
    {
        /* Receive buffer is empty, suspend */
        goto suspend;
    }

    d_start= rl_inw(port, RL_CAPR) + RL_CAPR_DATA_OFF;
    d_end= rl_inw(port, RL_CBR) % RX_BUFSIZE;

    if (d_start >= RX_BUFSIZE)
    {
        printf("rl_readv: strange value in RL_CAPR: 0x%x\n",
            rl_inw(port, RL_CAPR));
        d_start %= RX_BUFSIZE;
    }

    if (d_end > d_start)
        amount= d_end-d_start;
    else
        amount= d_end+RX_BUFSIZE - d_start;

    rxstat = *(u32_t *) (rep->v_re_rx_buf + d_start);

#if DEAD_CODE
    src_phys= rep->re_rx_buf + d_start;
    cps = sys_physcopy(
        NONE, PHYS_SEG, src_phys,
        SELF, D, (vir_bytes) &rxstat, sizeof(rxstat));
    if (cps != OK) printf("RTL8139: warning, sys_abscopy failed: %d\n", cps);
#endif

    if (rep->re_clear_rx)
    {
#if 0
        printf("rl_readv: late buffer overflow\n");
#endif
        goto suspend; /* Buffer overflow */
    }

    /* Should convert from little endian to host byte order */

    if (!(rxstat & RL_RXS_ROK))
    {
        printf("rxstat= 0x%08lx\n", rxstat);
        printf("d_start: 0x%x, d_end: 0x%x, rxstat: 0x%lx\n",
            d_start, d_end, rxstat);
        panic("rtl8139", "received packet not OK", NO_NUM);
    }
    totlen= (rxstat >> RL_RXS_LEN_S);
    if (totlen < 8 || totlen > 2*ETH_MAX_PACK_SIZE)
    {
        /* Someting went wrong */
        printf(
            "rl_readv: bad length (%u) in status 0x%08lx at offset 0x%x\n",
            totlen, rxstat, d_start);
        printf(
            "d_start: 0x%x, d_end: 0x%x, totlen: %d, rxstat: 0x%lx\n",
            d_start, d_end, totlen, rxstat);
        panic(NULL, NULL, NO_NUM);
    }

#if 0
    printf("d_start: 0x%x, d_end: 0x%x, totlen: %d, rxstat: 0x%x\n",
        d_start, d_end, totlen, rxstat);
#endif

    if (totlen+4 > amount)
    {
        printf("rl_readv: packet not yet ready\n");
        goto suspend;
    }

    /* Should subtract the CRC */
```

```

    packlen= totlen - ETH_CRC_SIZE;

    if (vectored)
    {
        int iov_offset = 0;
#   if 0
        if ((cps = sys_umap(re_client, D, (vir_bytes) mp->DL_ADDR,
            count * sizeof(rep->re_iovec[0]), &iov_src)) != OK)
            printf("sys_umap failed: %d\n", cps);
#   endif

        size= 0;
        o= d_start+4;
        src_phys= rep->re_rx_buf;
        for (i= 0; i<count; i += IOVEC_NR,
            iov_src += IOVEC_NR * sizeof(rep->re_iovec[0]),
            iov_offset += IOVEC_NR * sizeof(rep->re_iovec[0]))
        {
            n= IOVEC_NR;
            if (i+n > count)
                n= count-i;
#   if 0
                cps = sys_physcopy(NONE, PHYS_SEG, iov_src, SELF, D, (vir_bytes)
rep->re_iovec,
                    n * sizeof(rep->re_iovec[0]));
            if (cps != OK) printf("RTL8139: warning, sys_abscopy failed: %d\n", cps);
#   else
                cps = sys_vircopy(re_client, D, (vir_bytes) mp->DL_ADDR + iov_off
set,
                    SELF, D, (vir_bytes) rep->re_iovec, n * sizeof(rep->re_io
vec[0]));
            if (cps != OK) printf("RTL8139: warning, sys_vircopy failed: %d(%d)\n", cps, __LINE__);
#   endif

            for (j= 0, iovp= rep->re_iovec; j<n; j++, iovp++)
            {
                s= iovp->iov_size;
                if (size + s > packlen)
                {
                    assert(packlen > size);
                    s= packlen-size;
                }

#   if 0
                if (sys_umap(re_client, D, iovp->iov_addr, s, &dst_phys)
!= OK)
                    panic("rtl8139", "umap_local failed\n", NO_NUM);
#   endif

                if (o >= RX_BUFSIZE)
                {
                    o -= RX_BUFSIZE;
                    assert(o < RX_BUFSIZE);
                }

                if (o+s > RX_BUFSIZE)
                {
                    assert(o<RX_BUFSIZE);
                    s1= RX_BUFSIZE-o;

#   if 0
                    cps = sys_abscopy(src_phys+o, dst_phys, s1);
                    if (cps != OK) printf("RTL8139: warning, sys_abscopy failed: %d\n", cps);
                    cps = sys_abscopy(src_phys, dst_phys+s1, s-s1);
                    if (cps != OK) printf("RTL8139: warning, sys_abscopy failed: %d\n", cps);
#   else
                    cps = sys_vircopy(SELF, D, (vir_bytes) rep->v_re
rx_buf+o,
                        re_client, D, iovp->iov_addr, s1);
                    if (cps != OK) printf("RTL8139: warning, sys_vircopy failed: %d(%d)\n", cps, __LINE__);
                    cps = sys_vircopy(SELF, D, (vir_bytes) rep->v_re
rx_buf,
                        re_client, D, iovp->iov_addr+s1, s-s1);
                    if (cps != OK) printf("RTL8139: warning, sys_vircopy failed: %d(%d)\n", cps, __LINE__);

```

```

#endif
                                }
                                else
                                {
#if 0
                                cps = sys_abscopy(src_phys+o, dst_phys, s);
                                if (cps != OK) printf("RTL8139: warning, sys_abscopy failed: %d\n", cps);
#else
                                cps = sys_vircopy(SELF, D, (vir_bytes) rep->v_re_
rx_buf+o,
                                re_client, D, iovp->iov_addr, s);
                                if (cps != OK) printf("RTL8139: warning, sys_vircopy failed: %d (%d)\n", cps, __LINE__);
#endif
                                }

                                size += s;
                                if (size == packlen)
                                    break;
                                o += s;
                                }
                                if (size == packlen)
                                    break;
                                }
                                if (size < packlen)
                                {
                                    assert(0);
                                }
                                }
                                else
                                {
                                    assert(0);
                                }
#if 0
                                size= mp->DL_COUNT;
                                if (size < ETH_MIN_PACKET_SIZE || size > ETH_MAX_PACKET_SIZE_TAGGED)
                                    panic("rtl8139", "invalid packet size", size);
                                if (OK != sys_umap(re_client, D, (vir_bytes) mp->DL_ADDR, size, &phys_user
))
                                    panic("rtl8139", "umap_local failed", NO_NUM);

                                p= rep->re_tx[tx_head].ret_buf;
                                cps = sys_abscopy(phys_user, p, size);
                                if (cps != OK) printf("RTL8139: warning, sys_abscopy failed: %d\n", cps);
#endif
                                }

                                if (rep->re_clear_rx)
                                {
                                    /* For some reason the receiver FIFO is not stopped when
                                     * the buffer is full.
                                     */
#if 0
                                    printf("rl_readv: later buffer overflow\n");
#endif
                                    goto suspend; /* Buffer overflow */
                                }

                                rep->re_stat.ets_packetR++;
                                rep->re_read_s= packlen;
                                rep->re_flags= (rep->re_flags & ~REF_READING) | REF_PACK_RECV;

                                /* Avoid overflow in 16-bit computations */
                                l= d_start;
                                l += totlen+4;
                                l= (l+3) & ~3; /* align */
                                if (l >= RX_BUFSIZE)
                                {
                                    l -= RX_BUFSIZE;
                                    assert(l < RX_BUFSIZE);
                                }
                                rl_outw(port, RL_CAPR, l-RL_CAPR_DATA_OFF);

                                if (!from_int)
                                    reply(rep, OK, FALSE);

```

```

        return;

suspend:
    if (from_int)
    {
        assert(rep->re_flags & REF_READING);

        /* No need to store any state */
        return;
    }

    rep->re_rx_mess= *mp;
    assert(!(rep->re_flags & REF_READING));
    rep->re_flags |= REF_READING;

    reply(rep, OK, FALSE);
}

/*=====
 *                               rl_writev                               *
 *=====*/
static void rl_writev(mp, from_int, vectored)
message *mp;
int from_int;
int vectored;
{
    phys_bytes p, iov_src, phys_user;
    int i, j, n, s, port, count, size;
    int tx_head, re_client;
    re_t *rep;
    iovect_t *iovp;
    char *ret;
    int cps;

    port = mp->DL_PORT;
    count = mp->DL_COUNT;
    if (port < 0 || port >= RE_PORT_NR)
        panic("rtl8139", "illegal port", port);
    rep= &re_table[port];
    re_client= mp->DL_PROC;
    rep->re_client= re_client;

    assert(rep->re_mode == REM_ENABLED);
    assert(rep->re_flags & REF_ENABLED);

    if (from_int)
    {
        assert(rep->re_flags & REF_SEND_AVAIL);
        rep->re_flags &= ~REF_SEND_AVAIL;
        rep->re_send_int= FALSE;
        rep->re_tx_alive= TRUE;
    }

    tx_head= rep->re_tx_head;
    if (rep->re_tx[tx_head].ret_busy)
    {
        assert(!(rep->re_flags & REF_SEND_AVAIL));
        rep->re_flags |= REF_SEND_AVAIL;
        if (rep->re_tx[tx_head].ret_busy)
            goto suspend;

        /* Race condition, the interrupt handler may clear re_busy
         * before we got a chance to set REF_SEND_AVAIL. Checking
         * ret_busy twice should be sufficient.
         */
    }

#if 0
    printf("rl_writev: race detected\n");
#endif

    rep->re_flags &= ~REF_SEND_AVAIL;
    rep->re_send_int= FALSE;
}

    assert(!(rep->re_flags & REF_SEND_AVAIL));
    assert(!(rep->re_flags & REF_PACK_SENT));

```

```

        if (vectored)
        {
            int iov_offset = 0;

#if 0
            if (OK != sys_umap(re_client, D, (vir_bytes)mp->DL_ADDR,
                count * sizeof(rep->re_iovec[0]), &iov_src))
                panic("rtl8139", "umap_local failed", NO_NUM);
#endif

            size= 0;

#if 0
            p= rep->re_tx[tx_head].ret_buf;
#else
            ret = rep->re_tx[tx_head].v_ret_buf;
#endif

            for (i= 0; i<count; i += IOVEC_NR,
                iov_src += IOVEC_NR * sizeof(rep->re_iovec[0]),
                iov_offset += IOVEC_NR * sizeof(rep->re_iovec[0]))
            {
                n= IOVEC_NR;
                if (i+n > count)
                    n= count-i;

                cps = sys_physcopy(NONE, PHYS_SEG, iov_src, SELF, D, (vir
_bytes) rep->re_iovec,
                    n * sizeof(rep->re_iovec[0]));
                if (cps != OK) printf("RTL8139: warning, sys_abscopy failed: %d\n", cps);
            #else
                cps = sys_vircopy(re_client, D, ((vir_bytes) mp->DL_ADDR)
+ iov_offset,
                    SELF, D, (vir_bytes) rep->re_iovec,
                    n * sizeof(rep->re_iovec[0]));
                if (cps != OK) printf("RTL8139: warning, sys_vircopy failed: %d\n", cps);
            #endif

            for (j= 0, iovp= rep->re_iovec; j<n; j++, iovp++)
            {
                s= iovp->iov_size;
                if (size + s > ETH_MAX_PACK_SIZE_TAGGED)
                {
                    panic("rtl8139", "invalid packet size",
                        NO_NUM);
                }

                if (OK != sys_umap(re_client, D, iovp->iov_addr, s, &phys
_user))
                    panic("rtl8139", "umap_local failed\n", NO_NUM);

                if (cps != OK) printf("RTL8139: warning, sys_abscopy failed: %d\n", cps);
            #else
                cps = sys_vircopy(re_client, D, iovp->iov_addr,
                    SELF, D, (vir_bytes) ret, s);
                if (cps != OK) printf("RTL8139: warning, sys_vircopy failed: %d\n", cps);
            #endif

                size += s;

                if 0
                {
                    p += s;

                    ret += s;
                }
            }

            if (size < ETH_MIN_PACK_SIZE)
                panic("rtl8139", "invalid packet size", size);
        }
    }
    else
    {
        size= mp->DL_COUNT;
        if (size < ETH_MIN_PACK_SIZE || size > ETH_MAX_PACK_SIZE_TAGGED)
            panic("rtl8139", "invalid packet size", size);
    }
}

```



```

    if (OK != sys_umap(re_client, D, (vir_bytes)mp->DL_ADDR, size, &phys_user
))
        panic("rtl8139", "umap_local failed\n", NO_NUM);

    p = rep->re_tx[tx_head].ret_buf;
    cps = sys_abscopy(phys_user, p, size);
    if (cps != OK) printf("RTL8139: warning, sys_abscopy failed: %d\n", cps);
#else
    ret = rep->re_tx[tx_head].v_ret_buf;
    cps = sys_vircopy(re_client, D, (vir_bytes)mp->DL_ADDR,
        SELF, D, (vir_bytes) ret, size);
    if (cps != OK) printf("RTL8139: warning, sys_abscopy failed: %d\n", cps);
#endif
}

rl_outl(rep->re_base_port, RL_TSD0+tx_head*4,
    rep->re_ertxth | size);
rep->re_tx[tx_head].ret_busy = TRUE;

if (++tx_head == N_TX_BUF)
    tx_head = 0;
assert(tx_head < RL_N_TX);
rep->re_tx_head = tx_head;

rep->re_flags |= REF_PACK_SENT;

/* If the interrupt handler called, don't send a reply. The reply
 * will be sent after all interrupts are handled.
 */
if (from_int)
    return;
reply(rep, OK, FALSE);
return;

suspend:
#if 0
    printf("rl_writev: head %d, tail %d, busy: %d %d %d %d\n",
        tx_head, rep->re_tx_tail,
        rep->re_tx[0].ret_busy, rep->re_tx[1].ret_busy,
        rep->re_tx[2].ret_busy, rep->re_tx[3].ret_busy);
    printf("rl_writev: TSD: 0x%x, 0x%x, 0x%x, 0x%x\n",
        rl_inl(rep->re_base_port, RL_TSD0+0*4),
        rl_inl(rep->re_base_port, RL_TSD0+1*4),
        rl_inl(rep->re_base_port, RL_TSD0+2*4),
        rl_inl(rep->re_base_port, RL_TSD0+3*4));
#endif

    if (from_int)
        panic("rtl8139", "should not be sending\n", NO_NUM);

    rep->re_tx_mess = *mp;
    reply(rep, OK, FALSE);
}

/*=====
 *                               rl_check_ints                               *
 *=====*/
static void rl_check_ints(rep)
re_t *rep;
{
#if 0
10-1f    R/W    TSD[0-3]    Transmit Status of Descriptor [0-3]
        31     R      CRS      Carrier Sense Lost
        30     R      TABT     Transmit Abort
        29     R      OWC      Out of Window Collision
        27-24   R      NCC[3-0] Number of Collision Count
        23-22   reserved
        21-16   R/W    ERTXH[5-0] Early Tx Threshold
        15     R      TOK      Transmit OK
        14     R      TUN      Transmit FIFO Underrun
        13     R/W    OWN      OWN
        12-0    R/W    SIZE     Descriptor Size
3e-3f    R/W    ISR      Interrupt Status Register
        6      R/W    FOVW     Fx FIFO Overflow Interrupt

```

	5	R/W	PUN/LinkChg	Packet Underrun / Link Change Interrupt
	3	R/W	TER	Transmit Error Interrupt
	2	R/W	TOK	Transmit OK Interrupt
3e-3f	R/W	ISR		Interrupt Status Register
	15	R/W	SERR	System Error Interrupt
	14	R/W	TimeOut	Time Out Interrupt
	13	R/W	LenChg	Cable Length Change Interrupt
3e-3f	R/W	ISR		Interrupt Status Register
	4	R/W	RXOVW	Rx Buffer Overflow Interrupt
	1	R/W	RER	Receive Error Interrupt
	0	R/W	ROK	Receive OK Interrupt
4c-4f	R/W	MPC		Missed Packet Counter
60-61	R	TSAD		Transmit Status of All Descriptors
	15-12	R	TOK[3-0]	TOK bit of Descriptor [3-0]
	11-8	R	TUN[3-0]	TUN bit of Descriptor [3-0]
	7-4	R	TABT[3-0]	TABT bit of Descriptor [3-0]
	3-0	R	OWN[3-0]	OWN bit of Descriptor [3-0]
6c-6d	R	DIS		Disconnect Counter
	15-0	R	DCNT	Disconnect Counter
6e-6f	R	FCSC		False Carrier Sense Counter
	15-0	R	FCSCNT	False Carrier event counter
72-73	R	REC		RX_ER Counter
	15-0	R	RXERCNT	Received packet counter

#endif

int re\_flags;

re\_flags= rep-&gt;re\_flags;

if ((re\_flags &amp; REF\_READING) &amp;&amp;

!(rl\_inb(rep-&gt;re\_base\_port, RL\_CR) &amp; RL\_CR\_BUFE))

{

if (rep-&gt;re\_rx\_mess.m\_type == DL\_READV)

{

rl\_readv(&rep->re\_rx\_mess, TRUE /\* from int \*/,  
TRUE /\* vectored \*/);

}

else

{

assert(rep->re\_rx\_mess.m\_type == DL\_READ);  
rl\_readv(&rep->re\_rx\_mess, TRUE /\* from int \*/,  
FALSE /\* !vectored \*/);

}

}

if (rep-&gt;re\_clear\_rx)

rl\_clear\_rx(rep);

if (rep-&gt;re\_need\_reset)

rl\_do\_reset(rep);

if (rep-&gt;re\_send\_int)

{

if (rep-&gt;re\_tx\_mess.m\_type == DL\_WRITEV)

{

rl\_writev(&rep->re\_tx\_mess, TRUE /\* from int \*/,  
TRUE /\* vectored \*/);

}

else

{

assert(rep->re\_tx\_mess.m\_type == DL\_WRITE);  
rl\_writev(&rep->re\_tx\_mess, TRUE /\* from int \*/,  
FALSE /\* !vectored \*/);

}

}

if (rep-&gt;re\_report\_link)

rl\_report\_link(rep);

if (rep-&gt;re\_flags &amp; (REF\_PACK\_SENT | REF\_PACK\_RECV))

reply(rep, OK, TRUE);

}

```

/*=====
*
*                               rl_report_link
*
*=====

```

```

=====*/
static void rl_report_link(rep)
re_t *rep;
{
    port_t port;
    ul6_t mii_ctrl, mii_status, mii_ana, mii_anlpa, mii_ane, mii_extstat;
    u8_t msr;
    int f, link_up;

    rep->re_report_link= FALSE;
    port= rep->re_base_port;
    msr= rl_inb(port, RL_MSR);
    link_up= !(msr & RL_MSR_LINKB);
    rep->re_link_up= link_up;
    if (!link_up)
    {
        printf("%s: link down\n", rep->re_name);
        return;
    }

    mii_ctrl= rl_inw(port, RL_BMCR);
    mii_status= rl_inw(port, RL_BMSR);
    mii_ana= rl_inw(port, RL_ANAR);
    mii_anlpa= rl_inw(port, RL_ANLPA);
    mii_ane= rl_inw(port, RL_ANER);
    mii_extstat= 0;

    if (mii_ctrl & (MII_CTRL_LB|MII_CTRL_PD|MII_CTRL_ISO))
    {
        printf("%s: PHY: ", rep->re_name);
        f= 1;
        if (mii_ctrl & MII_CTRL_LB)
        {
            printf("loopback mode");
            f= 0;
        }
        if (mii_ctrl & MII_CTRL_PD)
        {
            if (!f) printf(", ");
            f= 0;
            printf("powered down");
        }
        if (mii_ctrl & MII_CTRL_ISO)
        {
            if (!f) printf(", ");
            f= 0;
            printf("isolated");
        }
        printf("\n");
        return;
    }
    if (!(mii_ctrl & MII_CTRL_ANE))
    {
        printf("%s: manual config: ", rep->re_name);
        switch(mii_ctrl & (MII_CTRL_SP_LSB|MII_CTRL_SP_MSB))
        {
            case MII_CTRL_SP_10:    printf("10 Mbps"); break;
            case MII_CTRL_SP_100:   printf("100 Mbps"); break;
            case MII_CTRL_SP_1000:  printf("1000 Mbps"); break;
            case MII_CTRL_SP_RES:   printf("reserved speed"); break;
        }
        if (mii_ctrl & MII_CTRL_DM)
            printf(", full duplex");
        else
            printf(", half duplex");
        printf("\n");
        return;
    }

    if (!debug) goto resspeed;

    printf("%s: ", rep->re_name);
    mii_print_stat_speed(mii_status, mii_extstat);
    printf("\n");
}

```

```

if (!(mii_status & MII_STATUS_ANC))
    printf("%s: auto-negotiation not complete\n", rep->re_name);
if (mii_status & MII_STATUS_RF)
    printf("%s: remote fault detected\n", rep->re_name);
if (!(mii_status & MII_STATUS_ANA))
{
    printf("%s: local PHY has no auto-negotiation ability\n",
        rep->re_name);
}
if (!(mii_status & MII_STATUS_LS))
    printf("%s: link down\n", rep->re_name);
if (mii_status & MII_STATUS_JD)
    printf("%s: jabber condition detected\n", rep->re_name);
if (!(mii_status & MII_STATUS_EC))
{
    printf("%s: no extended register set\n", rep->re_name);
    goto resspeed;
}
if (!(mii_status & MII_STATUS_ANC))
    goto resspeed;

printf("%s: local cap.: ", rep->re_name);
mii_print_techab(mii_ana);
printf("\n");

if (mii_ane & MII_ANE_PDF)
    printf("%s: parallel detection fault\n", rep->re_name);
if (!(mii_ane & MII_ANE_LPANA))
{
    printf("%s: link-partner does not support auto-negotiation\n",
        rep->re_name);
    goto resspeed;
}

printf("%s: remote cap.: ", rep->re_name);
mii_print_techab(mii_anlpa);
printf("\n");

```

```

resspeed:
    printf("%s: ", rep->re_name);
    printf("link up at %d Mbps, ", (msr & RL_MSR_SPEED_10) ? 10 : 100);
    printf("%s duplex\n", ((mii_ctrl & MII_CTRL_DM) ? "full" : "half"));

```

```

}

```

```

static void mii_print_techab(techab)
ul6_t techab;
{
    int fs, ft;
    if ((techab & MII_ANA_SEL_M) != MII_ANA_SEL_802_3)
    {
        printf("strange selector 0x%x, value 0x%x",
            techab & MII_ANA_SEL_M,
            (techab & MII_ANA_TAF_M) >> MII_ANA_TAF_S);
        return;
    }
    fs= 1;
    if (techab & (MII_ANA_100T4 | MII_ANA_100TXFD | MII_ANA_100TXHD))
    {
        printf("100 Mbps: ");
        fs= 0;
        ft= 1;
        if (techab & MII_ANA_100T4)
        {
            printf("T4");
            ft= 0;
        }
        if (techab & (MII_ANA_100TXFD | MII_ANA_100TXHD))
        {
            if (!ft)
                printf(",");
            ft= 0;
            printf("TX-");
        }
    }
}

```

```

        switch (techab & (MII_ANA_100TXFD | MII_ANA_100TXHD))
        {
            case MII_ANA_100TXFD:    printf("FD"); break;
            case MII_ANA_100TXHD:    printf("HD"); break;
            default:                  printf("FD/HD"); break;
        }
    }
}
if (techab & (MII_ANA_10TFD | MII_ANA_10THD))
{
    if (!fs)
        printf(",");
    printf("10 Mbps: ");
    fs = 0;
    printf("T-");
    switch (techab & (MII_ANA_10TFD | MII_ANA_10THD))
    {
        case MII_ANA_10TFD:        printf("FD"); break;
        case MII_ANA_10THD:        printf("HD"); break;
        default:                    printf("FD/HD"); break;
    }
}
if (techab & MII_ANA_PAUSE_SYM)
{
    if (!fs)
        printf(",");
    fs = 0;
    printf("pause(SYM)");
}
if (techab & MII_ANA_PAUSE_ASYM)
{
    if (!fs)
        printf(",");
    fs = 0;
    printf("pause(ASYM)");
}
if (techab & MII_ANA_TAF_RES)
{
    if (!fs)
        printf(",");
    fs = 0;
    printf("0x%x", (techab & MII_ANA_TAF_RES) >> MII_ANA_TAF_S);
}
}

static void mii_print_stat_speed(stat, extstat)
ul6_t stat;
ul6_t extstat;
{
    int fs, ft;
    fs = 1;
    if (stat & MII_STATUS_EXT_STAT)
    {
        if (extstat & (MII_ESTAT_1000XFD | MII_ESTAT_1000XHD |
            MII_ESTAT_1000TFD | MII_ESTAT_1000THD))
        {
            printf("1000 Mbps: ");
            fs = 0;
            ft = 1;
            if (extstat & (MII_ESTAT_1000XFD | MII_ESTAT_1000XHD))
            {
                ft = 0;
                printf("X-");
                switch (extstat &
                    (MII_ESTAT_1000XFD | MII_ESTAT_1000XHD))
                {
                    case MII_ESTAT_1000XFD: printf("FD"); break;
                    case MII_ESTAT_1000XHD: printf("HD"); break;
                    default:                  printf("FD/HD"); break;
                }
            }
            if (extstat & (MII_ESTAT_1000TFD | MII_ESTAT_1000THD))
            {
                if (!ft)

```

```

        printf(",");
        ft= 0;
        printf("T-");
        switch(extstat &
            (MII_ESTAT_1000TFD|MII_ESTAT_1000THD))
        {
            case MII_ESTAT_1000TFD: printf("FD"); break;
            case MII_ESTAT_1000THD: printf("HD"); break;
            default:                 printf("FD/HD"); break;
        }
    }
}
if (stat & (MII_STATUS_100T4 |
    MII_STATUS_100XFD | MII_STATUS_100XHD |
    MII_STATUS_100T2FD | MII_STATUS_100T2HD))
{
    if (!fs)
        printf(",");
    fs= 0;
    printf("100 Mbps: ");
    ft= 1;
    if (stat & MII_STATUS_100T4)
    {
        printf("T4");
        ft= 0;
    }
    if (stat & (MII_STATUS_100XFD | MII_STATUS_100XHD))
    {
        if (!ft)
            printf(",");
        ft= 0;
        printf("TX-");
        switch(stat & (MII_STATUS_100XFD|MII_STATUS_100XHD))
        {
            case MII_STATUS_100XFD: printf("FD"); break;
            case MII_STATUS_100XHD: printf("HD"); break;
            default:                 printf("FD/HD"); break;
        }
    }
    if (stat & (MII_STATUS_100T2FD | MII_STATUS_100T2HD))
    {
        if (!ft)
            printf(",");
        ft= 0;
        printf("T2-");
        switch(stat & (MII_STATUS_100T2FD|MII_STATUS_100T2HD))
        {
            case MII_STATUS_100T2FD: printf("FD"); break;
            case MII_STATUS_100T2HD: printf("HD"); break;
            default:                 printf("FD/HD"); break;
        }
    }
}
if (stat & (MII_STATUS_10FD | MII_STATUS_10HD))
{
    if (!fs)
        printf(",");
    printf("10 Mbps: ");
    fs= 0;
    printf("T-");
    switch(stat & (MII_STATUS_10FD|MII_STATUS_10HD))
    {
        case MII_STATUS_10FD: printf("FD"); break;
        case MII_STATUS_10HD: printf("HD"); break;
        default:               printf("FD/HD"); break;
    }
}
}

/*=====
 *                               *
 *                               *
 *=====*/
static void rl_clear_rx(rep)

```

```

re_t *rep;
{
    port_t port;
    u8_t cr;
    int i;
    clock_t t0,t1;

    rep->re_clear_rx= FALSE;
    port= rep->re_base_port;

    /* Reset the receiver */
    cr= rl_inb(port, RL_CR);
    cr &= ~RL_CR_RE;
    rl_outb(port, RL_CR, cr);
    getuptime(&t0);
    do {
        if (!(rl_inb(port, RL_CR) & RL_CR_RE))
            break;
    } while (getuptime(&t1)==OK && (t1-t0) < HZ);
    if (rl_inb(port, RL_CR) & RL_CR_RE)
        panic("rtl8139", "cannot disable receiver", NO_NUM);

#ifdef 0
    printf("RBSTART=0x%08x\n", rl_inl(port, RL_RBSTART));
    printf("CAPR=0x%04x\n", rl_inw(port, RL_CAPR));
    printf("CBR=0x%04x\n", rl_inw(port, RL_CBR));
    printf("RCR=0x%08x\n", rl_inl(port, RL_RCR));
#endif

    rl_outb(port, RL_CR, cr | RL_CR_RE);

    rl_outl(port, RL_RCR, RX_BUFBITS);

    rl_rec_mode(rep);

    rep->re_stat.ets_missedP++;
}

/*=====
 *                               rl_do_reset                               *
 *=====*/
static void rl_do_reset(rep)
re_t *rep;
{
    rep->re_need_reset= FALSE;
    rl_reset_hw(rep);
    rl_rec_mode(rep);

    rep->re_tx_head= 0;
    if (rep->re_flags & REF_SEND_AVAIL)
    {
        rep->re_tx[rep->re_tx_head].ret_busy= FALSE;
        rep->re_send_int= TRUE;
    }
}

/*=====
 *                               rl_getstat                               *
 *=====*/
static void rl_getstat(mp)
message *mp;
{
    int port;
    eth_stat_t stats;
    re_t *rep;

    port = mp->DL_PORT;
    if (port < 0 || port >= RE_PORT_NR)
        panic("rtl8139", "illegal port", port);
    rep= &re_table[port];
    rep->re_client= mp->DL_PROC;

    assert(rep->re_mode == REM_ENABLED);
    assert(rep->re_flags & REF_ENABLED);

```

```

        stats= rep->re_stat;

        put_userdata(mp->DL_PROC, (vir_bytes) mp->DL_ADDR,
                    (vir_bytes) sizeof(stats), &stats);
        reply(rep, OK, FALSE);
}

/*=====
 *                               rl_getname                               *
 *=====*/
static void rl_getname(mp)
message *mp;
{
    int r;

    strncpy(mp->DL_NAME, progname, sizeof(mp->DL_NAME));
    mp->DL_NAME[sizeof(mp->DL_NAME)-1] = '\0';
    mp->m_type= DL_NAME_REPLY;
    r= send(mp->m_source, mp);
    if (r != OK)
        panic("RTL8139", "rl_getname: send failed: %d\n", r);
}

/*=====
 *                               reply                               *
 *=====*/
static void reply(rep, err, may_block)
re_t *rep;
int err;
int may_block;
{
    message reply;
    int status;
    int r;
    clock_t now;

    status = 0;
    if (rep->re_flags & REF_PACK_SENT)
        status |= DL_PACK_SEND;
    if (rep->re_flags & REF_PACK_RECV)
        status |= DL_PACK_RECV;

    reply.m_type = DL_TASK_REPLY;
    reply.DL_PORT = rep - re_table;
    reply.DL_PROC = rep->re_client;
    reply.DL_STAT = status | ((u32_t) err << 16);
    reply.DL_COUNT = rep->re_read_s;
    if (OK != (r = getuptime(&now)))
        panic("rtl8139", "getuptime() failed:", r);
    reply.DL_CLCK = now;

    r= send(rep->re_client, &reply);

    if (r == ELOCKED && may_block)
    {
#if 0
        printW(); printf("send locked\n");
#endif
        return;
    }

    if (r < 0) {
        printf("RTL8139 tried sending to %d, type %d\n", rep->re_client, reply.m_type);
        panic("rtl8139", "send failed:", r);
    }

    rep->re_read_s = 0;
    rep->re_flags &= ~(REF_PACK_SENT | REF_PACK_RECV);
}

/*=====

```



```

*                                     mess_reply                                     *
*=====*/
static void mess_reply(req, reply_mess)
message *req;
message *reply_mess;
{
    if (send(req->m_source, reply_mess) != OK)
        panic("rtl8139", "unable to mess_reply", NO_NUM);
}

/*=====*
*                                     put_userdata                                     *
*=====*/
static void put_userdata(user_proc, user_addr, count, loc_addr)
int user_proc;
vir_bytes user_addr;
vir_bytes count;
void *loc_addr;
{
    int cps;
    cps = sys_datacopy(SELF, (vir_bytes) loc_addr, user_proc, user_addr, count);
    if (cps != OK) printf("RTL8139: warning, scopy failed: %d\n", cps);
}

#if 0
static void dump_phy(rep)
re_t *rep;
{
    port_t port;
    u32_t t;

    port= rep->re_base_port;

    t= rl_inb(port, RL_MSR);
    printf("MSR: 0x%02lx\n", t);
    if (t & RL_MSR_SPEED_10)
        printf("\t10 Mbps\n");
    if (t & RL_MSR_LINKB)
        printf("\tLink failed\n");

    t= rl_inb(port, RL_CONFIG1);
    printf("CONFIG1: 0x%02lx\n", t);

    t= rl_inb(port, RL_CONFIG3);
    printf("CONFIG3: 0x%02lx\n", t);

    t= rl_inb(port, RL_CONFIG4);
    printf("CONFIG4: 0x%02lx\n", t);

    t= rl_inw(port, RL_BMCR);
    printf("BMCR (MII_CTRL): 0x%04lx\n", t);

    t= rl_inw(port, RL_BMSR);
    printf("BMSR:");
    if (t & MII_STATUS_100T4)
        printf(" 100Base-T4");
    if (t & MII_STATUS_100XFD)
        printf(" 100Base-X-FD");
    if (t & MII_STATUS_100XHD)
        printf(" 100Base-X-HD");
    if (t & MII_STATUS_10FD)
        printf(" 10Mbps-FD");
    if (t & MII_STATUS_10HD)
        printf(" 10Mbps-HD");
    if (t & MII_STATUS_100T2FD)
        printf(" 100Base-T2-FD");
    if (t & MII_STATUS_100T2HD)
        printf(" 100Base-T2-HD");
    if (t & MII_STATUS_EXT_STAT)
        printf(" Ext-stat");
    if (t & MII_STATUS_RES)
        printf(" res-0x%lx", t & MII_STATUS_RES);
    if (t & MII_STATUS_MFPS)
        printf(" MFPS");
}

```

```

    if (t & MII_STATUS_ANC)
        printf(" ANC");
    if (t & MII_STATUS_RF)
        printf(" remote-fault");
    if (t & MII_STATUS_ANA)
        printf(" ANA");
    if (t & MII_STATUS_LS)
        printf(" Link");
    if (t & MII_STATUS_JD)
        printf(" Jabber");
    if (t & MII_STATUS_EC)
        printf(" Extended-capability");
    printf("\n");

    t= rl_inw(port, RL_ANAR);
    printf("ANAR (MII_ANA): 0x%04lx\n", t);

    t= rl_inw(port, RL_ANLPAR);
    printf("ANLPAR: 0x%04lx\n", t);

    t= rl_inw(port, RL_ANER);
    printf("ANER (MII_ANE): ");
    if (t & MII_ANE_RES)
        printf(" res-0x%lx", t & MII_ANE_RES);
    if (t & MII_ANE_PDF)
        printf(" Par-Detect-Fault");
    if (t & MII_ANE_LPNPA)
        printf(" LP-Next-Page-Able");
    if (t & MII_ANE_NPA)
        printf(" Loc-Next-Page-Able");
    if (t & MII_ANE_PR)
        printf(" Page-Received");
    if (t & MII_ANE_LPANA)
        printf(" LP-Auto-Neg-Able");
    printf("\n");

    t= rl_inw(port, RL_NWAYTR);
    printf("NWAYTR: 0x%04lx\n", t);
    t= rl_inw(port, RL_CSCR);
    printf("CSCR: 0x%04lx\n", t);

    t= rl_inb(port, RL_CONFIG5);
    printf("CONFIG5: 0x%02lx\n", t);
}
#endif

static int do_hard_int(void)
{
    int i,s;

    for (i=0; i < RE_PORT_NR; i++) {

        /* Run interrupt handler at driver level. */
        rl_handler( &re_table[i]);

        /* Reenable interrupts for this hook. */
        if ((s=sys_irqenable(&re_table[i].re_hook_id)) != OK)
            printf("RTL8139: error, couldn't enable interrupts: %d\n", s);
    }
}

/*=====
 *                               rl_handler                               *
 *=====*/
static int rl_handler(rep)
re_t *rep;
{
    int i, port, tx_head, tx_tail, link_up;
    u16_t isr, tsad;
    u32_t tsd, tcr, ertxth;

    #if 0
    u8_t cr;
    #endif

    clock_t t0,t1;

```

```

int_event_check = FALSE;          /* disable check by default */

port= rep->re_base_port;

/* Ack interrupt */
isr= rl_inw(port, RL_ISR);
rl_outw(port, RL_ISR, isr);

if (isr & RL_IMR_FOVW)
{
    isr &= ~RL_IMR_FOVW;
    /* Should do anything? */

    rep->re_stat.ets_fifoOver++;
}
if (isr & RL_IMR_PUN)
{
    isr &= ~RL_IMR_PUN;

    /* Either the link status changed or there was a TX fifo
     * underrun.
     */
    link_up= !(rl_inb(port, RL_MSR) & RL_MSR_LINKB);
    if (link_up != rep->re_link_up)
    {
        rep->re_report_link= TRUE;
        rep->re_got_int= TRUE;
        int_event_check = TRUE;
    }
}
if (isr & RL_IMR_RXOVW)
{
    isr &= ~RL_IMR_RXOVW;

    /* Clear the receive buffer */
    rep->re_clear_rx= TRUE;
    rep->re_got_int= TRUE;
    int_event_check = TRUE;
}

if (isr & (RL_ISR_RER | RL_ISR_ROK))
{
    isr &= ~(RL_ISR_RER | RL_ISR_ROK);

    if (!rep->re_got_int && (rep->re_flags & REF_READING))
    {
        rep->re_got_int= TRUE;
        int_event_check = TRUE;
    }
}

#if 0
if ((isr & (RL_ISR_TER | RL_ISR_TOK)) &&
    (rep->re_flags & REF_SEND_AVAIL) &&
    (rep->re_tx[0].ret_busy || rep->re_tx[1].ret_busy ||
     rep->re_tx[2].ret_busy || rep->re_tx[3].ret_busy))

{
    printf(
        "rl_handler, SEND_AVAIL: tx_head %d, tx_tail %d, busy: %d %d %d %d\n",
        rep->re_tx_head, rep->re_tx_tail,
        rep->re_tx[0].ret_busy, rep->re_tx[1].ret_busy,
        rep->re_tx[2].ret_busy, rep->re_tx[3].ret_busy);
    printf(
        "rl_handler: TSAD: 0x%04x, TSD: 0x%08x, 0x%08x, 0x%08x, 0x%08x\n",
        rl_inw(port, RL_TSAD),
        rl_inl(port, RL_TSD0+0*4),
        rl_inl(port, RL_TSD0+1*4),
        rl_inl(port, RL_TSD0+2*4),
        rl_inl(port, RL_TSD0+3*4));
}
#endif
if ((isr & (RL_ISR_TER | RL_ISR_TOK)) || 1)
{
    isr &= ~(RL_ISR_TER | RL_ISR_TOK);

```

```

tsad= rl_inw(port, RL_TSAD);
if (tsad & (RL_TSAD_TABT0|RL_TSAD_TABT1|
            RL_TSAD_TABT2|RL_TSAD_TABT3))
{
#if 0
    /* Do we need a watch dog? */
    /* Just reset the whole chip */
    rep->re_need_reset= TRUE;
    rep->re_got_int= TRUE;
    int_event_check = TRUE;

#elif 0
    /* Reset transmitter */
    rep->re_stat.ets_transAb++;

    cr= rl_inb(port, RL_CR);
    cr &= ~RL_CR_TE;
    rl_outb(port, RL_CR, cr);
    getuptime(&t0);
    do {
        if (!(rl_inb(port, RL_CR) & RL_CR_TE))
            break;
    } while (getuptime(&t1)==OK && (t1-t0) < HZ);
    if (rl_inb(port, RL_CR) & RL_CR_TE)
    {
        panic("rtl8139", "cannot disable transmitter",
              NO_NUM);
    }
    rl_outb(port, RL_CR, cr | RL_CR_TE);

    tcr= rl_inl(port, RL_TCR);
    rl_outl(port, RL_TCR, tcr | RL_TCR_IFG_STD);

    printf("rl_handler: reset after abort\n");

    if (rep->re_flags & REF_SEND_AVAIL)
    {
        printf("rl_handler: REF_SEND_AVAIL\n");
        rep->re_send_int= TRUE;
        rep->re_got_int= TRUE;
        int_event_check = TRUE;
    }
    for (i= 0; i< N_TX_BUF; i++)
        rep->re_tx[i].ret_busy= FALSE;
    rep->re_tx_head= 0;

#else

    printf("rl_handler, TABT, tsad = 0x%04x\n",
           tsad);

    /* Find the aborted transmit request */
    for (i= 0; i< N_TX_BUF; i++)
    {
        tsd= rl_inl(port, RL_TSD0+i*4);
        if (tsd & RL_TSD_TABT)
            break;
    }
    if (i >= N_TX_BUF)
    {
        printf(
            "rl_handler: can't find aborted TX req.\n" );
    }
    else
    {
        printf("TSD%d = 0x%04lx\n", i, tsd);

        /* Set head and tail to this buffer */
        rep->re_tx_head= rep->re_tx_tail= i;
    }

    /* Aborted transmission, just kick the device
     * and be done with it.
     */
    rep->re_stat.ets_transAb++;
    tcr= rl_inl(port, RL_TCR);

```

```

                                rl_outl(port, RL_TCR, tcr | RL_TCR_CLRABT);
#endif
    }

    /* Transmit completed */
    tx_head= rep->re_tx_head;
    tx_tail= rep->re_tx_tail;
    for (i= 0; i< 2*N_TX_BUF; i++)
    {
        if (!rep->re_tx[tx_tail].ret_busy)
        {
            /* Strange, this buffer is not in-use.
             * Increment tx_tail until tx_head is
             * reached (or until we find a buffer that
             * is in-use.
             */
            if (tx_tail == tx_head)
                break;
            if (++tx_tail >= N_TX_BUF)
                tx_tail= 0;
            assert(tx_tail < RL_N_TX);
            rep->re_tx_tail= tx_tail;
            continue;
        }
        tsd= rl_inl(port, RL_TSD0+tx_tail*4);
        if (!(tsd & RL_TSD_OWN))
        {
            /* Buffer is not yet ready */
            break;
        }

        /* Should collect statistics */
        if (tsd & RL_TSD_CRS)
            rep->re_stat.ets_carrSense++;
        if (tsd & RL_TSD_TABT)
        {
            printf("rl_handler, TABT, TSD%d = 0x%04lx\n",
                    tx_tail, tsd);
            assert(0); /* CLRABT is not all that
                       * effective, why not?
                       */
            rep->re_stat.ets_transAb++;
            tcr= rl_inl(port, RL_TCR);
            rl_outl(port, RL_TCR, tcr | RL_TCR_CLRABT);
        }
        if (tsd & RL_TSD_OWC)
            rep->re_stat.ets_OWC++;
        if (tsd & RL_TSD_CDH)
            rep->re_stat.ets_CDheartbeat++;

        /* What about collisions? */
        if (tsd & RL_TSD_TOK)
            rep->re_stat.ets_packetT++;
        else
            rep->re_stat.ets_sendErr++;
        if (tsd & RL_TSD_TUN)
        {
            rep->re_stat.ets_fifoUnder++;

            /* Increase ERTXTH */
            ertxth= tsd + (1 << RL_TSD_ERTXTH_S);
            ertxth &= RL_TSD_ERTXTH_M;
            if (debug && ertxth > rep->re_ertxth)
            {
                printf("%s: new ertxth: %ld bytes\n",
                        rep->re_name,
                        (ertxth >> RL_TSD_ERTXTH_S) *
                        32);
                rep->re_ertxth= ertxth;
            }
        }
        rep->re_tx[tx_tail].ret_busy= FALSE;
    }

```

#if 0

```

        if (rep->re_flags & REF_SEND_AVAIL)
        {
            printf("TSD%d: %08lx\n", tx_tail, tsd);
            printf(
                "rl_handler: head %d, tail %d, busy: %d %d %d %d\n",
                tx_head, tx_tail,
                rep->re_tx[0].ret_busy, rep->re_tx[1].ret_busy,
                rep->re_tx[2].ret_busy, rep->re_tx[3].ret_busy);
        }
#endif

        if (++tx_tail >= N_TX_BUF)
            tx_tail = 0;
        assert(tx_tail < RL_N_TX);
        rep->re_tx_tail = tx_tail;

        if (rep->re_flags & REF_SEND_AVAIL)
        {
            printf("rl_handler: REF_SEND_AVAIL\n");

            rep->re_send_int = TRUE;
            if (!rep->re_got_int)
            {
                rep->re_got_int = TRUE;
                int_event_check = TRUE;
            }
        }
        assert(i < 2*N_TX_BUF);
    }
    if (isr)
    {
        printf("rl_handler: unhandled interrupt: isr = 0x%04x\n",
            isr);
    }

    return 1;
}

/*=====
 *                               rl_watchdog_f                               *
 *=====*/
static void rl_watchdog_f(tp)
timer_t *tp;
{
    int i;
    re_t *rep;
    /* Use a synchronous alarm instead of a watchdog timer. */
    sys_setalarm(HZ, 0);

    for (i = 0, rep = &re_table[0]; i < RE_PORT_NR; i++, rep++)
    {
        if (rep->re_mode != REM_ENABLED)
            continue;
        if (!(rep->re_flags & REF_SEND_AVAIL))
        {
            /* Assume that an idle system is alive */
            rep->re_tx_alive = TRUE;
            continue;
        }
        if (rep->re_tx_alive)
        {
            rep->re_tx_alive = FALSE;
            continue;
        }
        printf("rl_watchdog_f: resetting port %d\n", i);
        printf(
            "TSAD: 0x%04x, TSD: 0x%08x, 0x%08x, 0x%08x, 0x%08x\n",
            rl_inw(rep->re_base_port, RL_TSAD),
            rl_inl(rep->re_base_port, RL_TSD0+0*4),
            rl_inl(rep->re_base_port, RL_TSD0+1*4),
            rl_inl(rep->re_base_port, RL_TSD0+2*4),
            rl_inl(rep->re_base_port, RL_TSD0+3*4));
    }
}

```

```

        printf("tx_head %d, tx_tail %d, busy: %d %d %d %d\n",
               rep->re_tx_head, rep->re_tx_tail,
               rep->re_tx[0].ret_busy, rep->re_tx[1].ret_busy,
               rep->re_tx[2].ret_busy, rep->re_tx[3].ret_busy);
        rep->re_need_reset= TRUE;
        rep->re_got_int= TRUE;

        check_int_events();
    }
}

#if 0

_PROTOTYPE( static void rtl_init, (struct dpeth *dep) )
_PROTOTYPE( static u16_t get_ee_word, (dpeth_t *dep, int a) )
_PROTOTYPE( static void ee_wen, (dpeth_t *dep) )
_PROTOTYPE( static void set_ee_word, (dpeth_t *dep, int a, U16_t w) )
_PROTOTYPE( static void ee_wds, (dpeth_t *dep) )

static void rtl_init(dep)
dpeth_t *dep;
{
    u8_t reg_a, reg_b, cr, config0, config2, config3;
    int i;
    char val[128];

    printf("rtl_init called\n");
    ne_init(dep);

    /* ID */
    outb_reg0(dep, DP_CR, CR_PS_P0);
    reg_a = inb_reg0(dep, DP_DUM1);
    reg_b = inb_reg0(dep, DP_DUM2);

    printf("rtl_init: '%c', '%c'\n", reg_a, reg_b);

    outb_reg0(dep, DP_CR, CR_PS_P3);
    config0 = inb_reg3(dep, 3);
    config2 = inb_reg3(dep, 5);
    config3 = inb_reg3(dep, 6);
    outb_reg0(dep, DP_CR, CR_PS_P0);

    printf("rtl_init: config 0/2/3 = %x/%x/%x\n",
           config0, config2, config3);

    if (0 == sys_getkenv("RTL8029FD", 9+1, val, sizeof(val)))
    {
        printf("rtl_init: setting full-duplex mode\n");
        outb_reg0(dep, DP_CR, CR_PS_P3);

        cr= inb_reg3(dep, 1);
        outb_reg3(dep, 1, cr | 0xc0);

        outb_reg3(dep, 6, config3 | 0x40);
        config3 = inb_reg3(dep, 6);

        config2= inb_reg3(dep, 5);
        outb_reg3(dep, 5, config2 | 0x20);
        config2= inb_reg3(dep, 5);

        outb_reg3(dep, 1, cr);

        outb_reg0(dep, DP_CR, CR_PS_P0);

        printf("rtl_init: config 2 = %x\n", config2);
        printf("rtl_init: config 3 = %x\n", config3);
    }

    for (i= 0; i<64; i++)
        printf("%x ", get_ee_word(dep, i));
    printf("\n");

    if (0 == sys_getkenv("RTL8029MN", 9+1, val, sizeof(val)))
    {

```

```

        ee_wen(dep);

        set_ee_word(dep, 0x78/2, 0x10ec);
        set_ee_word(dep, 0x7A/2, 0x8029);
        set_ee_word(dep, 0x7C/2, 0x10ec);
        set_ee_word(dep, 0x7E/2, 0x8029);

        ee_wds(dep);

        assert(get_ee_word(dep, 0x78/2) == 0x10ec);
        assert(get_ee_word(dep, 0x7A/2) == 0x8029);
        assert(get_ee_word(dep, 0x7C/2) == 0x10ec);
        assert(get_ee_word(dep, 0x7E/2) == 0x8029);
    }

    if (0 == sys_getkenv("RTL8029XXX", 10+1, val, sizeof(val)))
    {
        ee_wen(dep);

        set_ee_word(dep, 0x76/2, 0x8029);

        ee_wds(dep);

        assert(get_ee_word(dep, 0x76/2) == 0x8029);
    }
}

static ul6_t get_ee_word(dep, a)
dpeth_t *dep;
int a;
{
    int b, i, cmd;
    ul6_t w;

    outb_reg0(dep, DP_CR, CR_PS_P3);          /* Bank 3 */

    /* Switch to 9346 mode and enable CS */
    outb_reg3(dep, 1, 0x80 | 0x8);

    cmd= 0x180 | (a & 0x3f);          /* 1 1 0 a5 a4 a3 a2 a1 a0 */
    for (i= 8; i >= 0; i--)
    {
        b= (cmd & (1 << i));
        b= (b ? 2 : 0);

        /* Cmd goes out on the rising edge of the clock */
        outb_reg3(dep, 1, 0x80 | 0x8 | b);
        outb_reg3(dep, 1, 0x80 | 0x8 | 0x4 | b);
    }
    outb_reg3(dep, 1, 0x80 | 0x8); /* End of cmd */

    w= 0;
    for (i= 0; i<16; i++)
    {
        w <= 1;

        /* Data is shifted out on the rising edge. Read at the
         * falling edge.
         */
        outb_reg3(dep, 1, 0x80 | 0x8 | 0x4);
        outb_reg3(dep, 1, 0x80 | 0x8 | b);
        b= inb_reg3(dep, 1);
        w |= (b & 1);
    }

    outb_reg3(dep, 1, 0x80);          /* drop CS */
    outb_reg3(dep, 1, 0x00);          /* back to normal */
    outb_reg0(dep, DP_CR, CR_PS_P0); /* back to bank 0 */

    return w;
}

static void ee_wen(dep)
dpeth_t *dep;

```



```

{
    int b, i, cmd;
    ul6_t w;

    outb_reg0(dep, DP_CR, CR_PS_P3);          /* Bank 3 */

    /* Switch to 9346 mode and enable CS */
    outb_reg3(dep, 1, 0x80 | 0x8);

    cmd= 0x130;          /* 1 0 0 1 1 x x x x */
    for (i= 8; i >= 0; i--)
    {
        b= (cmd & (1 << i));
        b= (b ? 2 : 0);

        /* Cmd goes out on the rising edge of the clock */
        outb_reg3(dep, 1, 0x80 | 0x8 | b);
        outb_reg3(dep, 1, 0x80 | 0x8 | 0x4 | b);
    }
    outb_reg3(dep, 1, 0x80 | 0x8); /* End of cmd */
    outb_reg3(dep, 1, 0x80);      /* Drop CS */
    /* micro_delay(1); */          /* Is this required? */
}

static void set_ee_word(dep, a, w)
dpeth_t *dep;
int a;
ul6_t w;
{
    int b, i, cmd;
    clock_t t0, t1;

    outb_reg3(dep, 1, 0x80 | 0x8);          /* Set CS */

    cmd= 0x140 | (a & 0x3f);          /* 1 0 1 a5 a4 a3 a2 a1 a0 */
    for (i= 8; i >= 0; i--)
    {
        b= (cmd & (1 << i));
        b= (b ? 2 : 0);

        /* Cmd goes out on the rising edge of the clock */
        outb_reg3(dep, 1, 0x80 | 0x8 | b);
        outb_reg3(dep, 1, 0x80 | 0x8 | 0x4 | b);
    }
    for (i= 15; i >= 0; i--)
    {
        b= (w & (1 << i));
        b= (b ? 2 : 0);

        /* Cmd goes out on the rising edge of the clock */
        outb_reg3(dep, 1, 0x80 | 0x8 | b);
        outb_reg3(dep, 1, 0x80 | 0x8 | 0x4 | b);
    }
    outb_reg3(dep, 1, 0x80 | 0x8); /* End of data */
    outb_reg3(dep, 1, 0x80);      /* Drop CS */
    /* micro_delay(1); */          /* Is this required? */
    outb_reg3(dep, 1, 0x80 | 0x8); /* Set CS */
    getuptime(&t0);
    do {
        if (inb_reg3(dep, 1) & 1)
            break;
    } while (getuptime(&t1) == OK && (t1 == t0));
    if (!(inb_reg3(dep, 1) & 1))
        panic("set_ee_word", "device remains busy", NO_NUM);
}

static void ee_wds(dep)
dpeth_t *dep;
{
    int b, i, cmd;
    ul6_t w;

    outb_reg0(dep, DP_CR, CR_PS_P3);          /* Bank 3 */

```

```
/* Switch to 9346 mode and enable CS */
outb_reg3(dep, 1, 0x80 | 0x8);

cmd= 0x100; /* 1 0 0 0 0 x x x x */
for (i= 8; i >= 0; i--)
{
    b= (cmd & (1 << i));
    b= (b ? 2 : 0);

    /* Cmd goes out on the rising edge of the clock */
    outb_reg3(dep, 1, 0x80 | 0x8 | b);
    outb_reg3(dep, 1, 0x80 | 0x8 | 0x4 | b);
}
outb_reg3(dep, 1, 0x80 | 0x8); /* End of cmd */
outb_reg3(dep, 1, 0x80); /* Drop CS */
outb_reg3(dep, 1, 0x00); /* back to normal */
outb_reg0(dep, DP_CR, CR_PS_P0); /* back to bank 0 */
}
#endif

/*
 * $PchId: rtl8139.c,v 1.3 2003/09/11 14:15:15 philip Exp $
 */
```

```
/*
ibm/rtl8139.h

Created:      Aug 2003 by Philip Homburg <philip@cs.vu.nl>
*/

#define RL_IDR          0x00      /* Ethernet address
 * Note: RL_9346CR_EEM_CONFIG mode is
 * required the change the ethernet
 * address.
 * Note: 4-byte write access only.
 */

#define RL_N_TX          4        /* Number of transmit buffers */
#define RL_TSD0          0x010    /* Transmit Status of Descriptor 0 */
#define RL_TSD_CRS        0x80000000 /* Carrier Sense Lost */
#define RL_TSD_TABT       0x40000000 /* Transmit Abort */
#define RL_TSD_OWC        0x20000000 /* Out of Window Collision */
#define RL_TSD_CDH        0x10000000 /* CD Heart Beat */
#define RL_TSD_NCC_M      0x0F000000 /* Number of Collision Count */
#define RL_TSD_RES        0x00C00000 /* Reserved */
#define RL_TSD_ERTXTH_M   0x003F0000 /* Early Tx Threshold */
#define RL_TSD_ERTXTH_S   16       /* shift */
#define RL_TSD_ERTXTH_8   0x00000000 /* 8 bytes */
#define RL_TSD_TOK        0x00008000 /* Transmit OK */
#define RL_TSD_TUN        0x00004000 /* Transmit FIFO Underrun */
#define RL_TSD_OWN        0x00002000 /* Controller (does not) Own Buf. */
#define RL_TSD_SIZE       0x00001FFF /* Descriptor Size */
#define RL_TSAD0          0x20     /* Transmit Start Address of Descriptor 0 */
#define RL_RBSTART        0x30     /* Receive Buffer Start Address */
#define RL_CR             0x37     /* Command Register */
#define RL_CR_RES0        0xE0     /* Reserved */
#define RL_CR_RST         0x10     /* Reset */
#define RL_CR_RE          0x08     /* Receiver Enable */
#define RL_CR_TE          0x04     /* Transmitter Enable
 * Note: start with transmit buffer
 * 0 after RL_CR_TE has been reset.
 */
#define RL_CR_RES1        0x02     /* Reserved */
#define RL_CR_BUF0        0x01     /* Receive Buffer Empty */
#define RL_CAPR           0x38     /* Current Address of Packet Read */
#define RL_CAPR_DATA_OFF  0x10     /* Packet Starts at Offset */
#define RL_CBR            0x3A     /* Current Buffer Address */
#define RL_IMR            0x3C     /* Interrupt Mask Register */
#define RL_IMR_SERR       0x8000   /* System Error */
#define RL_IMR_TIMEOUT    0x4000   /* Time Out */
#define RL_IMR_LENCHG     0x2000   /* Cable Length Change */
#define RL_IMR_RES        0x1F80   /* Reserved */
#define RL_IMR_FOVW       0x0040   /* Rx FIFO Overflow */
#define RL_IMR_PUN        0x0020   /* Packet Underrun / Link Change */
#define RL_IMR_RXOVW      0x0010   /* Rx Buffer Overflow */
#define RL_IMR_TER        0x0008   /* Transmit Error */
#define RL_IMR_TOK        0x0004   /* Transmit OK */
#define RL_IMR_RER        0x0002   /* Receive Error */
#define RL_IMR_ROK        0x0001   /* Receive OK */
#define RL_ISR            0x3E     /* Interrupt Status Register */
#define RL_ISR_SERR       0x8000   /* System Error */
#define RL_ISR_TIMEOUT    0x4000   /* Time Out */
#define RL_ISR_LENCHG     0x2000   /* Cable Length Change */
#define RL_ISR_RES        0x1F80   /* Reserved */
#define RL_ISR_FOVW       0x0040   /* Rx FIFO Overflow */
#define RL_ISR_PUN        0x0020   /* Packet Underrun / Link Change */
#define RL_ISR_RXOVW      0x0010   /* Rx Buffer Overflow */
#define RL_ISR_TER        0x0008   /* Transmit Error */
#define RL_ISR_TOK        0x0004   /* Transmit OK */
#define RL_ISR_RER        0x0002   /* Receive Error */
#define RL_ISR_ROK        0x0001   /* Receive OK */
#define RL_TCR            0x40     /* Transmit Configuration Register
 * Note: RL_CR_TE has to be set to
 * set/change RL_TCR.
 */
#define RL_TCR_RES0       0x80000000 /* Reserved */
#define RL_TCR_HWVER_AM   0x7C000000 /* Hardware Version ID A */
#define RL_TCR_IFG_M      0x03000000 /* Interframe Gap Time */
#define RL_TCR_IFG_STD    0x03000000 /* IEEE 802.3 std */
```

```

#if 0
#undef RL_TCR_IFG_STD
#define RL_TCR_IFG_STD 0x00000000
#endif
#define RL_TCR_HWVER_BM 0x00C00000 /* Hardware Version ID B */
#define RL_TCR_HWVER_RTL8139 0x60000000 /* RTL8139 */
#define RL_TCR_HWVER_RTL8139A 0x70000000 /* RTL8139A */
#define RL_TCR_HWVER_RTL8139AG 0x74000000 /* RTL8139A-G */
#define RL_TCR_HWVER_RTL8139B 0x78000000 /* RTL8139B */
#define RL_TCR_HWVER_RTL8130 0x78000000 /* RTL8130 (dup) */
#define RL_TCR_HWVER_RTL8139C 0x74000000 /* RTL8139C (dup) */
#define RL_TCR_HWVER_RTL8100 0x78800000 /* RTL8100 */
#define RL_TCR_HWVER_RTL8100B 0x74400000 /* RTL8100B /
                                           RTL8139D */
#define RL_TCR_HWVER_RTL8139CP 0x74800000 /* RTL8139C+ */
#define RL_TCR_HWVER_RTL8101 0x74C00000 /* RTL8101 */
#define RL_TCR_RES1 0x00380000 /* Reserved */
#define RL_TCR_LBK_M 0x00060000 /* Loopback Test */
#define RL_TCR_LBK_NORMAL 0x00000000 /* Normal */
#define RL_TCR_LBK_LOOKBOCK 0x00060000 /* Loopback Mode */
#define RL_TCR_CRC 0x00010000 /* (Do not) Append CRC */
#define RL_TCR_RES2 0x0000F800 /* Reserved */
#define RL_TCR_MXDMA_M 0x00000700 /* Max DMA Burst Size Tx */
#define RL_TCR_MXDMA_16 0x00000000 /* 16 bytes */
#define RL_TCR_MXDMA_32 0x00000100 /* 32 bytes */
#define RL_TCR_MXDMA_64 0x00000200 /* 64 bytes */
#define RL_TCR_MXDMA_128 0x00000300 /* 128 bytes */
#define RL_TCR_MXDMA_128 0x00000300 /* 128 bytes */
#define RL_TCR_MXDMA_256 0x00000400 /* 256 bytes */
#define RL_TCR_MXDMA_512 0x00000500 /* 512 bytes */
#define RL_TCR_MXDMA_1024 0x00000600 /* 1024 bytes */
#define RL_TCR_MXDMA_2048 0x00000700 /* 2048 bytes */
#define RL_TCR_TXRR_M 0x000000F0 /* Tx Retry Count */
#define RL_TCR_RES3 0x0000000E /* Reserved */
#define RL_TCR_CLRABT 0x00000001 /* Clear Abort */
#define RL_RCR 0x44 /* Receive Configuration Register
                    * Note: RL_CR_RE has to be set to
                    * set/change RL_RCR.
                    */
#define RL_RCR_RES0 0xF0000000 /* Reserved */
#define RL_RCR_ERTH_M 0x0F000000 /* Early Rx Threshold */
#define RL_RCR_ERTH_0 0x00000000 /* No threshold */
#define RL_RCR_ERTH_1 0x01000000 /* 1/16 */
#define RL_RCR_ERTH_2 0x02000000 /* 2/16 */
#define RL_RCR_ERTH_3 0x03000000 /* 3/16 */
#define RL_RCR_ERTH_4 0x04000000 /* 4/16 */
#define RL_RCR_ERTH_5 0x05000000 /* 5/16 */
#define RL_RCR_ERTH_6 0x06000000 /* 6/16 */
#define RL_RCR_ERTH_7 0x07000000 /* 7/16 */
#define RL_RCR_ERTH_8 0x08000000 /* 8/16 */
#define RL_RCR_ERTH_9 0x09000000 /* 9/16 */
#define RL_RCR_ERTH_10 0x0A000000 /* 10/16 */
#define RL_RCR_ERTH_11 0x0B000000 /* 11/16 */
#define RL_RCR_ERTH_12 0x0C000000 /* 12/16 */
#define RL_RCR_ERTH_13 0x0D000000 /* 13/16 */
#define RL_RCR_ERTH_14 0x0E000000 /* 14/16 */
#define RL_RCR_ERTH_15 0x0F000000 /* 15/16 */
#define RL_RCR_RES1 0x00FC0000 /* Reserved */
#define RL_RCR_MULERINT 0x00020000 /* Multiple Early Int Select */
#define RL_RCR_RER8 0x00010000 /* Receive small error packet */
#define RL_RCR_RXFTH_M 0x0000E000 /* Rx FIFO Threshold */
#define RL_RCR_RXFTH_16 0x00000000 /* 16 bytes */
#define RL_RCR_RXFTH_32 0x00002000 /* 32 bytes */
#define RL_RCR_RXFTH_64 0x00004000 /* 64 bytes */
#define RL_RCR_RXFTH_128 0x00006000 /* 128 bytes */
#define RL_RCR_RXFTH_256 0x00008000 /* 256 bytes */
#define RL_RCR_RXFTH_512 0x0000A000 /* 512 bytes */
#define RL_RCR_RXFTH_1024 0x0000C000 /* 1024 bytes */
#define RL_RCR_RXFTH_UNLIM 0x0000E000 /* unlimited */
#define RL_RCR_RBLEM_M 0x00001800 /* Rx Buffer Length */
#define RL_RCR_RBLEM_8K 0x00000000 /* 8KB + 16 bytes */
#define RL_RCR_RBLEM_8K_SIZE (8*1024)
#define RL_RCR_RBLEM_16K 0x00000800 /* 16KB + 16 bytes */
#define RL_RCR_RBLEM_16K_SIZE (16*1024)

```

```
#define RL_RCR_RBLLEN_32K 0x00001000 /* 32KB + 16 bytes */
#define RL_RCR_RBLLEN_32K_SIZE (32*1024)
#define RL_RCR_RBLLEN_64K 0x00001800 /* 64KB + 16 bytes */
#define RL_RCR_RBLLEN_64K_SIZE (64*1024)
/* Note: the documentation for the RTL8139C(L) or
 * for the RTL8139D(L) claims that the buffer should
 * be 16 bytes larger. Multiples of 8KB are the
 * correct values.
 */
#define RL_RCR_MXDMA_M 0x00000700 /* Rx DMA burst size */
#define RL_RCR_MXDMA_16 0x00000000 /* 16 bytes */
#define RL_RCR_MXDMA_32 0x00000100 /* 32 bytes */
#define RL_RCR_MXDMA_64 0x00000200 /* 64 bytes */
#define RL_RCR_MXDMA_128 0x00000300 /* 128 bytes */
#define RL_RCR_MXDMA_256 0x00000400 /* 256 bytes */
#define RL_RCR_MXDMA_512 0x00000500 /* 512 bytes */
#define RL_RCR_MXDMA_1024 0x00000600 /* 1024 bytes */
#define RL_RCR_MXDMA_UNLIM 0x00000700 /* unlimited */
#define RL_RCR_WRAP 0x00000080 /* (Do not) Wrap on receive */
#define RL_RCR_RES2 0x00000040 /* EEPROM type? */
#define RL_RCR_AER 0x00000020 /* Accept Error Packets */
#define RL_RCR_AR 0x00000010 /* Accept Runt Packets */
#define RL_RCR_AB 0x00000008 /* Accept Broadcast Packets */
#define RL_RCR_AM 0x00000004 /* Accept Multicast Packets */
#define RL_RCR_APM 0x00000002 /* Accept Physical Match Packets */
#define RL_RCR_AAP 0x00000001 /* Accept All Packets */
#define RL_MPC 0x4c /* Missed Packet Counter */
#define RL_9346CR 0x50 /* 93C46 Command Register */
#define RL_9346CR_EEM_M 0xC0 /* Operating Mode */
#define RL_9346CR_EEM_NORMAL 0x00 /* Normal Mode */
#define RL_9346CR_EEM_AUTOLOAD 0x40 /* Load from 93C46 */
#define RL_9346CR_EEM_PROG 0x80 /* 93C46 Programming */
#define RL_9346CR_EEM_CONFIG 0xC0 /* Config Write Enable */
#define RL_9346CR_RES 0x30 /* Reserved */
#define RL_9346CR_EECS 0x08 /* EECS Pin */
#define RL_9346CR_EESK 0x04 /* EESK Pin */
#define RL_9346CR_EEDI 0x02 /* EEDI Pin */
#define RL_9346CR_EEDO 0x01 /* EEDO Pin */
#define RL_CONFIG0 0x51 /* Configuration Register 0 */
#define RL_CONFIG1 0x52 /* Configuration Register 1 */
#define RL_MSR 0x58 /* Media Status Register */
#define RL_MSR_TXFCE 0x80 /* Tx Flow Control Enable */
#define RL_MSR_RXFCE 0x40 /* Rx Flow Control Enable */
#define RL_MSR_RES 0x20 /* Reserved */
#define RL_MSR_AUXSTAT 0x10 /* Aux. Power Present */
#define RL_MSR_SPEED_10 0x08 /* In 10 Mbps mode */
#define RL_MSR_LINKB 0x04 /* link Failed */
#define RL_MSR_TXPF 0x02 /* Sent Pause Packet */
#define RL_MSR_RXPF 0x01 /* Received Pause Packet */
#define RL_CONFIG3 0x59 /* Configuration Register 3 */
#define RL_CONFIG4 0x5A /* Configuration Register 4 */
/* 0x5B */ /* Reserved */
#define RL_REVID 0x5E /* PCI Revision ID */
/* 0x5F */ /* Reserved */
#define RL_TSAD 0x60 /* Transmit Status of All Descriptors */
#define RL_TSAD_TOK3 0x8000 /* TOK bit of Descriptor 3 */
#define RL_TSAD_TOK2 0x4000 /* TOK bit of Descriptor 2 */
#define RL_TSAD_TOK1 0x2000 /* TOK bit of Descriptor 1 */
#define RL_TSAD_TOK0 0x1000 /* TOK bit of Descriptor 0 */
#define RL_TSAD_TUN3 0x0800 /* TUN bit of Descriptor 3 */
#define RL_TSAD_TUN2 0x0400 /* TUN bit of Descriptor 2 */
#define RL_TSAD_TUN1 0x0200 /* TUN bit of Descriptor 1 */
#define RL_TSAD_TUN0 0x0100 /* TUN bit of Descriptor 0 */
#define RL_TSAD_TABT3 0x0080 /* TABT bit of Descriptor 3 */
#define RL_TSAD_TABT2 0x0040 /* TABT bit of Descriptor 2 */
#define RL_TSAD_TABT1 0x0020 /* TABT bit of Descriptor 1 */
#define RL_TSAD_TABT0 0x0010 /* TABT bit of Descriptor 0 */
#define RL_TSAD_OWN3 0x0008 /* OWN bit of Descriptor 3 */
#define RL_TSAD_OWN2 0x0004 /* OWN bit of Descriptor 2 */
#define RL_TSAD_OWN1 0x0002 /* OWN bit of Descriptor 1 */
#define RL_TSAD_OWN0 0x0001 /* OWN bit of Descriptor 0 */
#define RL_BMCR 0x62 /* Basic Mode Control Register (MII_CTRL) */
#define RL_BMSR 0x64 /* Basic Mode Status Register (MII_STATUS) */
#define RL_ANAR 0x66 /* Auto-Neg Advertisement Register (MII_ANA) */
```

```

#define RL_ANLPA      0x68    /* Auto-Neg Link Partner Register (MII_ANLPA) */
#define RL_ANER       0x6a    /* Auto-Neg Expansion Register (MII_ANE) */
#define RL_NWAYTR     0x70    /* N-way Test Register */
#define RL_CSCR       0x74    /* CS Configuration Register */
#define RL_CONFIG5    0xD8    /* Configuration Register 5 */

/* Status word in receive buffer */
#define RL_RXS_LEN_M  0xFFFF0000 /* Length Field, Excl. Status word */
#define RL_RXS_LEN_S  16         /* Shift For Length */
#define RL_RXS_MAR     0x00008000 /* Multicast Address Received */
#define RL_RXS_PAR     0x00004000 /* Physical Address Matched */
#define RL_RXS_BAR     0x00002000 /* Broadcast Address Received */
#define RL_RXS_RES_M   0x00001FC0 /* Reserved */
#define RL_RXS_ISE     0x00000020 /* Invalid Symbol Error */
#define RL_RXS_RUNT    0x00000010 /* Runt Packet Received */
#define RL_RXS_LONG    0x00000008 /* Long (>4KB) Packet */
#define RL_RXS_CRC     0x00000004 /* CRC Error */
#define RL_RXS_FAE     0x00000002 /* Frame Alignment Error */
#define RL_RXS_ROK     0x00000001 /* Receive OK */

/* Registers in the Machine Independent Interface (MII) to the PHY.
 * IEEE 802.3 (2000 Edition) Clause 22.
 */
#define MII_CTRL      0x0      /* Control Register (basic) */
#define MII_CTRL_RST   0x8000 /* Reset PHY */
#define MII_CTRL_LB    0x4000 /* Enable Loopback Mode */
#define MII_CTRL_SP_LSB 0x2000 /* Speed Selection (LSB) */
#define MII_CTRL_ANE   0x1000 /* Auto Negotiation Enable */
#define MII_CTRL_PD    0x0800 /* Power Down */
#define MII_CTRL_ISO   0x0400 /* Isolate */
#define MII_CTRL_RAN    0x0200 /* Restart Auto-Negotiation Process */
#define MII_CTRL_DM     0x0100 /* Full Duplex */
#define MII_CTRL_CT     0x0080 /* Enable COL Signal Test */
#define MII_CTRL_SP_MSB 0x0040 /* Speed Selection (MSB) */
#define MII_CTRL_SP_10 0x0000 /* 10 Mb/s */
#define MII_CTRL_SP_100 0x2000 /* 100 Mb/s */
#define MII_CTRL_SP_1000 0x0040 /* 1000 Mb/s */
#define MII_CTRL_SP_RES 0x2040 /* Reserved */
#define MII_CTRL_RES   0x003F /* Reserved */
#define MII_STATUS     0x1      /* Status Register (basic) */
#define MII_STATUS_100T4 0x8000 /* 100Base-T4 support */
#define MII_STATUS_100XFD 0x4000 /* 100Base-X FD support */
#define MII_STATUS_100XHD 0x2000 /* 100Base-X HD support */
#define MII_STATUS_10FD  0x1000 /* 10 Mb/s FD support */
#define MII_STATUS_10HD  0x0800 /* 10 Mb/s HD support */
#define MII_STATUS_100T2FD 0x0400 /* 100Base-T2 FD support */
#define MII_STATUS_100T2HD 0x0200 /* 100Base-T2 HD support */
#define MII_STATUS_EXT_STAT 0x0100 /* Supports MII_EXT_STATUS */
#define MII_STATUS_RES   0x0080 /* Reserved */
#define MII_STATUS_MFPS  0x0040 /* MF Preamble Suppression */
#define MII_STATUS_ANC   0x0020 /* Auto-Negotiation Completed */
#define MII_STATUS_RF    0x0010 /* Remote Fault Detected */
#define MII_STATUS_ANA   0x0008 /* Auto-Negotiation Ability */
#define MII_STATUS_LS    0x0004 /* Link Up */
#define MII_STATUS_JD    0x0002 /* Jabber Condition Detected */
#define MII_STATUS_EC    0x0001 /* Ext Register Capabilities */
#define MII_PHYID_H     0x2      /* PHY ID (high) */
#define MII_PHYID_L     0x3      /* PHY ID (low) */
#define MII_ANA         0x4      /* Auto-Negotiation Advertisement */
#define MII_ANA_NP      0x8000 /* Next PAge */
#define MII_ANA_RES     0x4000 /* Reserved */
#define MII_ANA_RF      0x2000 /* Remote Fault */
#define MII_ANA_TAF_M   0x1FE0 /* Technology Ability Field */
#define MII_ANA_TAF_S   5        /* Shift */
#define MII_ANA_TAF_RES 0x1000 /* Reserved */
#define MII_ANA_PAUSE_ASYM 0x0800 /* Asym. Pause */
#define MII_ANA_PAUSE_SYM 0x0400 /* Sym. Pause */
#define MII_ANA_100T4   0x0200 /* 100Base-T4 */
#define MII_ANA_100TXFD 0x0100 /* 100Base-TX FD */
#define MII_ANA_100TXHD 0x0080 /* 100Base-TX HD */
#define MII_ANA_10TFD   0x0040 /* 10Base-T FD */
#define MII_ANA_10THD   0x0020 /* 10Base-T HD */
#define MII_ANA_SEL_M   0x001F /* Selector Field */
#define MII_ANA_SEL_802_3 0x0001 /* 802.3 */

```

```

#define MII_ANLPA      0x5      /* Auto-Neg Link Partner Ability Register */
#define MII_ANLPA_NP   0x8000 /* Next Page */
#define MII_ANLPA_ACK   0x4000 /* Acknowledge */
#define MII_ANLPA_RF    0x2000 /* Remote Fault */
#define MII_ANLPA_TAF_M 0x1FC0 /* Technology Ability Field */
#define MII_ANLPA_SEL_M 0x001F /* Selector Field */
#define MII_ANE        0x6      /* Auto-Negotiation Expansion */
#define MII_ANE_RES     0xFFE0 /* Reserved */
#define MII_ANE_PDF     0x0010 /* Parallel Detection Fault */
#define MII_ANE_LPNPA   0x0008 /* Link Partner is Next Page Able */
#define MII_ANE_NPA     0x0002 /* Local Device is Next Page Able */
#define MII_ANE_PR      0x0002 /* New Page has been received */
#define MII_ANE_LPANA   0x0001 /* Link Partner is Auto-Neg.able */
#define MII_ANNPT       0x7      /* Auto-Negotiation Next Page Transmit */
#define MII_ANLPRNP     0x8      /* Auto-Neg Link Partner Received Next Page */
#define MII_MS_CTRL     0x9      /* MASTER-SLAVE Control Register */
#define MII_MS_STATUS   0xA      /* MASTER-SLAVE Status Register */
/* 0xB ... 0xE */
#define MII_EXT_STATUS  0xF      /* Extended Status */
#define MII_ESTAT_1000XFD 0x8000 /* 1000Base-X Full Duplex */
#define MII_ESTAT_1000XHD 0x4000 /* 1000Base-X Half Duplex */
#define MII_ESTAT_1000TFD 0x2000 /* 1000Base-T Full Duplex */
#define MII_ESTAT_1000THD 0x1000 /* 1000Base-T Half Duplex */
#define MII_ESTAT_RES    0x0FFF /* Reserved */
/* 0x10 ... 0x1F */
/* Vendor Specific */

#if 0
34-35  R      ERBCR      Early Receive (Rx) Byte Count Register
36     R      ERSR      Early Rx Status Register
      7-4
      3       R      ERGood Early Rx Good packet
      2       R      ERBad  Early Rx Bad packet
      1       R      EROVW  Early Rx OverWrite
      0       R      EROK   Early Rx OK
51     R/W    CONFIG0    Configuration Register 0
      7       R      SCR    Scrambler Mode
      6       R      PCS    PCS Mode
      5       R      T10    10 Mbps Mode
      4-3     R      PL[1-0] Select 10 Mbps medium type
      2-0     R      BS[2-0] Select Boot ROM size
52     R/W    CONFIG1    Configuration Register 1
      7-6     R/W    LEDS[1-0] LED PIN
      5       R/W    DVRLOAD Driver Load
      4       R/W    LWACT   LWAKE active mode
      3       R      MEMMAP  Memory Mapping
      2       R      IOMAP   I/O Mapping
      1       R/W    VPD     Set to enable Vital Product Data
      0       R/W    PMEn    Power Management Enable
59     R/W    CONFIG3    Configuration Register 3
      7       R      GNTSel  Gnt Select
      6       R/W    PARM_En Parameter Enable
      5       R/W    Magic   Magic Packet
      4       R/W    LinkUp  Link Up
      3       reserved
      2       R      CLKRUN_En CLKRUN Enable
      1       reserved
      0       R      FBtBEn  Fast Back to Back Enable
5a     R/W    CONFIG4    Configuration Register 4
      7       R/W    RxFIFOAutoClr Auto Clear the Rx FIFO on overflow
      6       R/W    AnaOff  Analog Power Off
      5       R/W    LongWF  Long Wake-up Frame
      4       R/W    LWPME   LANWAKE vs PMEB
      3       reserved
      2       R/W    LWPTN   LWAKE pattern
      1       reserved
      0       R/W    PBWakeup Pre-Boot Wakeup
5c-5d  R/W    MULINT     Multiple Interrupt Select
      15-12   reserved
      11-0    R/W    MISR[11-0] Multiple Interrupt Select
68-69  R      ANLPA      Auto-Negotiation Link Partnet Register
      15      R      NP    Next Page bit
      14      R      ACK   acknowledge received from link partner
      13      R/W    RF    received remote fault detection capability
      12-11   reserved

```

```

10      R      Pause      Flow control is supported
9       R      T4         100Base-T4 is supported
8       R/W    TXFD       100Base-TX full duplex is supported
7       R/W    TX         100Base-TX is supported
6       R/W    10FD       10Base-T full duplex is supported
5       R/W    10         10Base-T is supported
4-0     R/W    Selector   Binary encoded selector
6a-6b   R      ANER       Auto-Negotiation Expansion Register
15-5    reserved
4       R      MLF        Multiple link fault occurred
3       R      LP_NP_ABLE Link partner supports Next Page
2       R      NP_ABLE    Local node is able to send add. Next Pages
1       R      PAGE_RX    Link Code Word Page received
0       R      LP_NW_ABLE Link partner supports NWay auto-negotiation
70-71   R/W    NWAYTR     N-way Test Register
15-8    reserved
7       R/W    NWLPBK     NWay loopback mode
6-4     reserved
3       R      ENNWLE     LED0 pin indicates linkpulse
2       R      FLAGABD    Auto-neg experienced ability detect state
1       R      FLAGPDF    Auto-neg exp. par. detection fault state
0       R      FLAGLSC    Auto-neg experienced link status check state
74-75   R/W    CSCR       CS Configuration Register
15      W      Testfun    Auto-neg speeds up internal timer
14-10   reserved
9       R/W    LD         Active low TPI link disable signal
8       R/W    HEARTBEAT  HEART BEAT enable
7       R/W    JBEN       Enable jabber function
6       R/W    F_LINK_100 Force 100 Mbps
5       R/W    F_Conect   Bypass disconnect function
4       reserved
3       R      Con_status Connected link detected
2       R/W    Con_status_En Configures LED1 to indicate conn. stat.
1       reserved
0       R/W    PASS_SCR   Bypass scramble
76-77   reserved
78-7b   R/W    PHY1_PARM  PHY parameter 1
7c-7f   R/W    TW_PARM    Twister parameter
80      R/W    PHY2_PARM  PHY parameter 2
81-83   reserved
84-8b   R/W    CRC[0-7]   Power Management CRC reg.[0-7] for frame[0-7]
8c-cb   R/W    Wakeup[0-7] Power Management wakeup frame[0-7] (64 bit)
cc-d3   R/W    LSB_CRC[0-7] LSB of the mask byte of makeup frame[0-7]
d4-d7   reserved
d8      R/W    Config5    Configuration register 5
7       reserved
6       R/W    BWF        Broadcast Wakeup Frame
5       R/W    MWF        Multicast Wakeup Frame
4       R/W    UWF        Unicast Wakeup Frame
3       R/W    FifoAddrPtr FIFO Address Pointer
2       R/W    LDPS       Link Down Power Saving mode
1       R/W    LANWake    LANWake Signal
0       R/W    PME_STS    PME_Status bit
d9-ff   reserved
#endif

/*
 * $PchId: rtl8139.h,v 1.1 2003/09/05 10:58:50 philip Exp $
 */

```



```
# Makefile for the Sound Blaster 16 driver (SB16)

# directories
u = /usr
i = $u/include
s = $i/sys
m = $i/minix
b = $i/ibm
d = ..

# programs, flags, etc.
CC =      exec cc
CFLAGS = -I$i
LDFLAGS = -i
LIBS = -lsys -lsysutil

# build local binary
all build:      sb16_dsp sb16_mixer
sb16_dsp:      sb16.o sb16_dsp.o
                $(CC) -o $@ $(LDFLAGS) sb16.o sb16_dsp.o $(LIBS)
sb16_mixer:    sb16.o sb16_mixer.o
                $(CC) -o $@ $(LDFLAGS) sb16.o sb16_mixer.o $(LIBS)

# install with other drivers
install:       /usr/sbin/sb16_dsp /usr/sbin/sb16_mixer
/usr/sbin/sb16_dsp:      sb16_dsp
                    install -o root -c $? $@
/usr/sbin/sb16_mixer:    sb16_mixer
                    install -o root -c $? $@

# clean up local files
clean:
    rm -f *.o *.bak sb16 sb16_dsp sb16_mixer

depend:
    /usr/bin/mkdep "$(CC) -E $(CPPFLAGS)" *.c > .depend

# Include generated dependencies.
include .depend
```

Sound Blaster 16 ISA driver for Minix 3

Note: supports audio playback and volume control (mixer),  
recording is not yet supported

Installation instructions SB16 driver Minix >= 3.0.7

```
- set IRQ and I/O address in sb16.h  
(default 7 and 220)  
- make install  
- MAKEDEV /dev/audio (if /dev/audio doesn't already exist)  
- service up /usr/sbin/sb16_dsp -dev /dev/audio  
- service up /usr/sbin/sb16_mixer -dev /dev/mixer  
done... (you can include the last 2 lines in /usr/etc/rc)
```

Peter Boonstoppel - September 2005  
paboont@cs.vu.nl

```
#include "sb16.h"

/*=====
 *
 * mixer_set
 *=====*/
PUBLIC int mixer_set(reg, data)
int reg;
int data;
{
    int i;

    sb16_outb(MIXER_REG, reg);
    for(i = 0; i < 100; i++);
    sb16_outb(MIXER_DATA, data);

    return OK;
}

/*=====
 *
 * sb16_inb
 *=====*/
PUBLIC int sb16_inb(port)
int port;
{
    int s;
    unsigned long value;

    if ((s=sys_inb(port, &value)) != OK)
        panic("SB16DSP", "sys_inb() failed", s);

    return value;
}

/*=====
 *
 * sb16_outb
 *=====*/
PUBLIC void sb16_outb(port, value)
int port;
int value;
{
    int s;

    if ((s=sys_outb(port, value)) != OK)
        panic("SB16DSP", "sys_outb() failed", s);
}
```

```

#ifndef SB16_H
#define SB16_H

#include "../drivers.h"
#include <sys/ioc_sound.h>
#include <minix/sound.h>

#define SB_TIMEOUT          32000    /* timeout count */

/* IRQ, base address and DMA channels */
#define SB_IRQ              7
#define SB_BASE_ADDR        0x220    /* 0x210, 0x220, 0x230, 0x240,
   * 0x250, 0x260, 0x280
   */
#define SB_DMA_8             1        /* 0, 1, 3 */
#define SB_DMA_16           5        /* 5, 6, 7 */
#if _WORD_SIZE == 2
#define DMA_SIZE             8192    /* Dma buffer MUST BE MULTIPLE OF 2 */
#else
#define DMA_SIZE             (64 * 1024) /* Dma buffer MUST BE MULTIPLE OF 2 */
#endif

/* Some defaults for the DSP */
#define DEFAULT_SPEED        22050    /* Sample rate */
#define DEFAULT_BITS         8        /* Nr. of bits */
#define DEFAULT_SIGN         0        /* 0 = unsigned, 1 = signed */
#define DEFAULT_STEREO       0        /* 0 = mono, 1 = stereo */

/* DMA port addresses */
#define DMA8_ADDR            ((SB_DMA_8 & 3) << 1) + 0x00
#define DMA8_COUNT          ((SB_DMA_8 & 3) << 1) + 0x01
#define DMA8_MASK            0x0A
#define DMA8_MODE            0x0B
#define DMA8_CLEAR           0x0C

/* If after this preprocessing stuff DMA8_PAGE is not defined
 * the 8-bit DMA channel specified is not valid
 */
#if SB_DMA_8 == 0
# define DMA8_PAGE          0x87
#else
# if SB_DMA_8 == 1
#   define DMA8_PAGE        0x83
# else
#   if SB_DMA_8 == 3
#     define DMA8_PAGE      0x82
#   endif
# endif
#endif

#define DMA16_ADDR           ((SB_DMA_16 & 3) << 2) + 0xC0
#define DMA16_COUNT          ((SB_DMA_16 & 3) << 2) + 0xC2
#define DMA16_MASK           0xD4
#define DMA16_MODE           0xD6
#define DMA16_CLEAR          0xD8

/* If after this preprocessing stuff DMA16_PAGE is not defined
 * the 16-bit DMA channel specified is not valid
 */
#if SB_DMA_16 == 5
# define DMA16_PAGE          0x8B
#else
# if SB_DMA_16 == 6
#   define DMA16_PAGE        0x89
# else
#   if SB_DMA_16 == 7
#     define DMA16_PAGE      0x8A
#   endif
# endif
#endif

```

```

/* DMA modes */
#define DMA16_AUTO_PLAY      0x58 + (SB_DMA_16 & 3)
#define DMA16_AUTO_REC       0x54 + (SB_DMA_16 & 3)
#define DMA8_AUTO_PLAY      0x58 + SB_DMA_8
#define DMA8_AUTO_REC       0x54 + SB_DMA_8

/* IO ports for soundblaster */
#define DSP_RESET            0x6 + SB_BASE_ADDR
#define DSP_READ              0xA + SB_BASE_ADDR
#define DSP_WRITE            0xC + SB_BASE_ADDR
#define DSP_COMMAND          0xC + SB_BASE_ADDR
#define DSP_STATUS           0xC + SB_BASE_ADDR
#define DSP_DATA_AVL         0xE + SB_BASE_ADDR
#define DSP_DATA16_AVL       0xF + SB_BASE_ADDR
#define MIXER_REG             0x4 + SB_BASE_ADDR
#define MIXER_DATA           0x5 + SB_BASE_ADDR
#define OPL3_LEFT            0x0 + SB_BASE_ADDR
#define OPL3_RIGHT           0x2 + SB_BASE_ADDR
#define OPL3_BOTH            0x8 + SB_BASE_ADDR

/* DSP Commands */
#define DSP_INPUT_RATE        0x42 /* set input sample rate */
#define DSP_OUTPUT_RATE      0x41 /* set output sample rate */
#define DSP_CMD_SPKON         0xD1 /* set speaker on */
#define DSP_CMD_SPKOFF        0xD3 /* set speaker off */
#define DSP_CMD_DMA8HALT      0xD0 /* halt DMA 8-bit operation */
#define DSP_CMD_DMA8CONT      0xD4 /* continue DMA 8-bit operation */
#define DSP_CMD_DMA16HALT     0xD5 /* halt DMA 16-bit operation */
#define DSP_CMD_DMA16CONT     0xD6 /* continue DMA 16-bit operation */
#define DSP_GET_VERSION       0xE1 /* get version number of DSP */
#define DSP_CMD_8BITAUTO_IN    0xCE /* 8 bit auto-initialized input */
#define DSP_CMD_8BITAUTO_OUT   0xC6 /* 8 bit auto-initialized output */
#define DSP_CMD_16BITAUTO_IN   0xBE /* 16 bit auto-initialized input */
#define DSP_CMD_16BITAUTO_OUT  0xB6 /* 16 bit auto-initialized output */
#define DSP_CMD_IRQREQ8        0xF2 /* Interrupt request 8 bit */
#define DSP_CMD_IRQREQ16       0xF3 /* Interrupt request 16 bit */

/* DSP Modes */
#define DSP_MODE_MONO_US      0x00 /* Mono unsigned */
#define DSP_MODE_MONO_S       0x10 /* Mono signed */
#define DSP_MODE_STEREO_US    0x20 /* Stereo unsigned */
#define DSP_MODE_STEREO_S     0x30 /* Stereo signed */

/* MIXER commands */
#define MIXER_RESET           0x00 /* Reset */
#define MIXER_DAC_LEVEL       0x04 /* Used for detection only */
#define MIXER_MASTER_LEFT     0x30 /* Master volume left */
#define MIXER_MASTER_RIGHT    0x31 /* Master volume right */
#define MIXER_DAC_LEFT        0x32 /* Dac level left */
#define MIXER_DAC_RIGHT       0x33 /* Dac level right */
#define MIXER_FM_LEFT         0x34 /* Fm level left */
#define MIXER_FM_RIGHT        0x35 /* Fm level right */
#define MIXER_CD_LEFT         0x36 /* Cd audio level left */
#define MIXER_CD_RIGHT        0x37 /* Cd audio level right */
#define MIXER_LINE_LEFT       0x38 /* Line in level left */
#define MIXER_LINE_RIGHT      0x39 /* Line in level right */
#define MIXER_MIC_LEVEL       0x3A /* Microphone level */
#define MIXER_PC_LEVEL        0x3B /* Pc speaker level */
#define MIXER_OUTPUT_CTRL     0x3C /* Output control */
#define MIXER_IN_LEFT         0x3D /* Input control left */
#define MIXER_IN_RIGHT        0x3E /* Input control right */
#define MIXER_GAIN_IN_LEFT    0x3F /* Input gain control left */
#define MIXER_GAIN_IN_RIGHT    0x40 /* Input gain control right */
#define MIXER_GAIN_OUT_LEFT    0x41 /* Output gain control left */
#define MIXER_GAIN_OUT_RIGHT   0x42 /* Output gain control right */
#define MIXER_AGC              0x43 /* Automatic gain control */
#define MIXER_TREBLE_LEFT     0x44 /* Treble left */
#define MIXER_TREBLE_RIGHT    0x45 /* Treble right */

```

```
#define MIXER_BASS_LEFT      0x46  /* Bass left */
#define MIXER_BASS_RIGHT    0x47  /* Bass right */
#define MIXER_SET_IRQ       0x80  /* Set irq number */
#define MIXER_SET_DMA       0x81  /* Set DMA channels */
#define MIXER_IRQ_STATUS    0x82  /* Irq status */

/* Mixer constants */
#define MIC                  0x01  /* Microphone */
#define CD_RIGHT             0x02
#define CD_LEFT             0x04
#define LINE_RIGHT          0x08
#define LINE_LEFT           0x10
#define FM_RIGHT            0x20
#define FM_LEFT             0x40

/* DSP constants */
#define DMA_NR_OF_BUFFERS   2
#define DSP_MAX_SPEED       44100  /* Max sample speed in KHz */
#define DSP_MIN_SPEED       4000   /* Min sample speed in KHz */
#define DSP_MAX_FRAGMENT_SIZE DMA_SIZE / DMA_NR_OF_BUFFERS /* Maximum fragment size */
#define DSP_MIN_FRAGMENT_SIZE 1024  /* Minimum fragment size */
#define DSP_NR_OF_BUFFERS   8

/* Number of bytes you can DMA before hitting a 64K boundary: */
#define dma_bytes_left(phys) \
    ((unsigned) (sizeof(int) == 2 ? 0 : 0x10000) - (unsigned) ((phys) & 0xFFFF))

_PROTOTYPE(int mixer_set, (int reg, int data));
_PROTOTYPE( int sb16_inb, (int port) );
_PROTOTYPE( void sb16_outb, (int port, int value) );

#endif /* SB16_H */
```

```

/* This file contains the driver for a DSP (Digital Sound Processor) on
 * a SoundBlaster 16 soundcard.
 *
 * The driver supports the following operations (using message format m2):
 *
 *      m_type      DEVICE      IO_ENDPT      COUNT      POSITION      ADDRESS
 * -----
 * | DEV_OPEN | device | proc nr |          |          |          |
 * |-----|
 * | DEV_CLOSE | device | proc nr |          |          |          |
 * |-----|
 * | DEV_READ  | device | proc nr | bytes   |          | buf ptr |
 * |-----|
 * | DEV_WRITE | device | proc nr | bytes   |          | buf ptr |
 * |-----|
 * | DEV_IOCTL | device | proc nr | func code |          | buf ptr |
 * |-----|
 *
 * The file contains one entry point:
 *
 *      main:          main entry when driver is brought up
 *
 *      August 24 2005          Ported driver to user space (only audio playback) (Peter
Boonstoppel)
 *      May 20 1995           Author: Michel R. Prevenier
 */

```

```

#include "sb16.h"

```

```

_PROTOTYPE(void main, (void));
FORWARD _PROTOTYPE( int dsp_open, (void) );
FORWARD _PROTOTYPE( int dsp_close, (void) );
FORWARD _PROTOTYPE( int dsp_ioctl, (message *m_ptr) );
FORWARD _PROTOTYPE( void dsp_write, (message *m_ptr) );
FORWARD _PROTOTYPE( void dsp_hardware_msg, (void) );
FORWARD _PROTOTYPE( void dsp_status, (message *m_ptr) );

FORWARD _PROTOTYPE( void reply, (int code, int replyee, int process, int status) );
FORWARD _PROTOTYPE( void init_buffer, (void) );
FORWARD _PROTOTYPE( int dsp_init, (void) );
FORWARD _PROTOTYPE( int dsp_reset, (void) );
FORWARD _PROTOTYPE( int dsp_command, (int value) );
FORWARD _PROTOTYPE( int dsp_set_size, (unsigned int size) );
FORWARD _PROTOTYPE( int dsp_set_speed, (unsigned int speed) );
FORWARD _PROTOTYPE( int dsp_set_stereo, (unsigned int stereo) );
FORWARD _PROTOTYPE( int dsp_set_bits, (unsigned int bits) );
FORWARD _PROTOTYPE( int dsp_set_sign, (unsigned int sign) );
FORWARD _PROTOTYPE( void dsp_dma_setup, (phys_bytes address, int count) );
FORWARD _PROTOTYPE( void dsp_setup, (void) );

PRIVATE int irq_hook_id;          /* id of irq hook at the kernel */

PRIVATE char DmaBuffer[DMA_SIZE + 64 * 1024];
PRIVATE char* DmaPtr;
PRIVATE phys_bytes DmaPhys;

PRIVATE char Buffer[DSP_MAX_FRAGMENT_SIZE * DSP_NR_OF_BUFFERS];

PRIVATE int DspVersion[2];
PRIVATE unsigned int DspStereo = DEFAULT_STEREO;
PRIVATE unsigned int DspSpeed = DEFAULT_SPEED;
PRIVATE unsigned int DspBits = DEFAULT_BITS;
PRIVATE unsigned int DspSign = DEFAULT_SIGN;
PRIVATE unsigned int DspFragmentSize = DSP_MAX_FRAGMENT_SIZE;
PRIVATE int DspAvail = 0;
PRIVATE int DspBusy = 0;
PRIVATE int DmaMode = 0;
PRIVATE int DmaBusy = -1;
PRIVATE int DmaFillNext = 0;
PRIVATE int BufReadNext = -1;
PRIVATE int BufFillNext = 0;

PRIVATE int revivePending = 0;

```

```

PRIVATE int reviveStatus;
PRIVATE int reviveProcNr;

#define dprint (void)

/*=====
 *                               main
 *=====*/
PUBLIC void main()
{
    int r, caller, proc_nr, s;
    message mess;

    dprint("sb16_dsp.c: main()\n");

    /* Get a DMA buffer. */
    init_buffer();

    while(TRUE) {
        /* Wait for an incoming message */
        receive(ANY, &mess);

        caller = mess.m_source;
        proc_nr = mess.IO_ENDPT;

        /* Now carry out the work. */
        switch(mess.m_type) {
            case DEV_OPEN:      r = dsp_open(); break;
            case DEV_CLOSE:     r = dsp_close(); break;
            case DEV_IOCTL:     r = dsp_ioctl(&mess); break;

            case DEV_READ:      r = EINVAL; break; /* Not yet implemented
*/
            case DEV_WRITE:     dsp_write(&mess); continue; /* don't repl
y */
            case DEV_STATUS:    dsp_status(&mess); continue; /* don't rep
ly */
            case HARD_INT:      dsp_hardware_msg(); continue; /* don't re
ply */
            case SYS_SIG:      continue; /* don't reply */
            default:            r = EINVAL;
        }

        /* Finally, prepare and send the reply message. */
        reply(TASK_REPLY, caller, proc_nr, r);
    }
}

/*=====
 *                               dsp_open
 *=====*/
PRIVATE int dsp_open()
{
    dprint("sb16_dsp.c: dsp_open()\n");

    /* try to detect SoundBlaster card */
    if(!DspAvail && dsp_init() != OK) return EIO;

    /* Only one open at a time with soundcards */
    if(DspBusy) return EBUSY;

    /* Start with a clean DSP */
    if(dsp_reset() != OK) return EIO;

    /* Setup default values */
    DspStereo = DEFAULT_STEREO;
    DspSpeed = DEFAULT_SPEED;
    DspBits = DEFAULT_BITS;
    DspSign = DEFAULT_SIGN;
    DspFragmentSize = DMA_SIZE / 2;

```



```

        DspBusy = 1;

        return OK;
}

/*=====
 *
 *                      dsp_close
 *=====*/
PRIVATE int dsp_close()
{
    dprint("sb16_dsp.c: dsp_close()\n");

    DspBusy = 0;                      /* soundcard available again */

    return OK;
}

/*=====
 *
 *                      dsp_ioctl
 *=====*/
PRIVATE int dsp_ioctl(m_ptr)
message *m_ptr;
{
    int status;
    phys_bytes user_phys;
    unsigned int val;

    dprint("sb16_dsp.c: dsp_ioctl()\n");

    /* Cannot change parameters during play or recording */
    if(DmaBusy >= 0) return EBUSY;

    /* Get user data */
    if(m_ptr->REQUEST != DSPIORESET) {
        sys_vircopy(m_ptr->IO_ENDPT, D, (vir_bytes)m_ptr->ADDRESS, SELF, D, (vir_
bytes)&val, sizeof(val));
    }

    dprint("dsp_ioctl: got ioctl %d, argument: %d\n", m_ptr->REQUEST, val);

    switch(m_ptr->REQUEST) {
        case DSPIORATE:          status = dsp_set_speed(val); break;
        case DSPIOSTEREO:       status = dsp_set_stereo(val); break;
        case DSPIOBITS:         status = dsp_set_bits(val); break;
        case DSPIOSIZE:         status = dsp_set_size(val); break;
        case DSPIOSIGN:         status = dsp_set_sign(val); break;
        case DSPIOMAX:
            val = DSP_MAX_FRAGMENT_SIZE;
            sys_vircopy(SELF, D, (vir_bytes)&val, m_ptr->IO_ENDPT, D, (vir_by
tes)m_ptr->ADDRESS, sizeof(val));
            status = OK;
            break;
        case DSPIORESET:       status = dsp_reset(); break;
        default:               status = ENOTTY; break;
    }

    return status;
}

/*=====
 *
 *                      dsp_write
 *=====*/
PRIVATE void dsp_write(m_ptr)
message *m_ptr;
{
    int s;
    message mess;

    dprint("sb16_dsp.c: dsp_write()\n");

```

```

    if(m_ptr->COUNT != DspFragmentSize) {
        reply(TASK_REPLY, m_ptr->m_source, m_ptr->IO_ENDPT, EINVAL);
        return;
    }
    if(m_ptr->m_type != DmaMode && DmaBusy >= 0) {
        reply(TASK_REPLY, m_ptr->m_source, m_ptr->IO_ENDPT, EBUSY);
        return;
    }

    reply(TASK_REPLY, m_ptr->m_source, m_ptr->IO_ENDPT, SUSPEND);

    if(DmaBusy < 0) { /* Dma tranfer not yet started */

        DmaMode = DEV_WRITE; /* Dma mode is writing */
        sys_datacopy(m_ptr->IO_ENDPT, (vir_bytes)m_ptr->ADDRESS, SELF, (vir_bytes
)DmaPtr, (phys_bytes)DspFragmentSize);
        dsp_dma_setup(DmaPhys, DspFragmentSize * DMA_NR_OF_BUFFERS);
        dsp_setup();
        DmaBusy = 0; /* Dma is busy */
        dprint(" filled dma[0]\n");
        DmaFillNext = 1;

    } else if(DmaBusy != DmaFillNext) { /* Dma transfer started, but Dma buffer not y
et full */

        sys_datacopy(m_ptr->IO_ENDPT, (vir_bytes)m_ptr->ADDRESS, SELF, (vir_bytes
)DmaPtr + DmaFillNext * DspFragmentSize, (phys_bytes)DspFragmentSize);
        dprint(" filled dma[%d]\n", DmaFillNext);
        DmaFillNext = (DmaFillNext + 1) % DMA_NR_OF_BUFFERS;

    } else if(BufReadNext < 0) { /* Dma buffer full, fill first element of second buf
fer */

        sys_datacopy(m_ptr->IO_ENDPT, (vir_bytes)m_ptr->ADDRESS, SELF, (vir_bytes
)Buffer, (phys_bytes)DspFragmentSize);
        dprint(" filled buf[0]\n");
        BufReadNext = 0;
        BufFillNext = 1;

    } else { /* Dma buffer is full, filling second buffer */

        while(BufReadNext == BufFillNext) { /* Second buffer also full, wait for
space to become available */
            receive(HARDWARE, &mess);
            dsp_hardware_msg();
        }
        sys_datacopy(m_ptr->IO_ENDPT, (vir_bytes)m_ptr->ADDRESS, SELF, (vir_bytes
)Buffer + BufFillNext * DspFragmentSize, (phys_bytes)DspFragmentSize);
        dprint(" filled buf[%d]\n", BufFillNext);
        BufFillNext = (BufFillNext + 1) % DSP_NR_OF_BUFFERS;

    }

    revivePending = 1;
    reviveStatus = DspFragmentSize;
    reviveProcNr = m_ptr->IO_ENDPT;
    notify(m_ptr->m_source);
}

/*=====
 *                               dsp_hardware_msg
 *=====*/
PRIVATE void dsp_hardware_msg()
{
    dprint("Interrupt: ");
    if(DmaBusy >= 0) { /* Dma transfer was actually busy */
        dprint("Finished playing dma[%d]; ", DmaBusy);
        DmaBusy = (DmaBusy + 1) % DMA_NR_OF_BUFFERS;
        if(DmaBusy == DmaFillNext) { /* Dma buffer empty, stop Dma transfer */

            dsp_command((DspBits == 8 ? DSP_CMD_DMA8HALT : DSP_CMD_DMA16HALT)

);
            dprint("No more work...!\n");

```

```

        DmaBusy = -1;

    } else if (BufReadNext >= 0) { /* Data in second buffer, copy one fragment
to Dma buffer */

        /* Acknowledge the interrupt on the DSP */
        sb16_inb((DspBits == 8 ? DSP_DATA_AVL : DSP_DATA16_AVL));

        memcpy(DmaPtr + DmaFillNext * DspFragmentSize, Buffer + BufReadNe
xt * DspFragmentSize, DspFragmentSize);
        dprint("copy buf[%d] -> dma[%d]; ", BufReadNext, DmaFillNext);
        BufReadNext = (BufReadNext + 1) % DSP_NR_OF_BUFFERS;
        DmaFillNext = (DmaFillNext + 1) % DMA_NR_OF_BUFFERS;
        if (BufReadNext == BufFillNext) {
            BufReadNext = -1;
        }
        dprint("Starting dma[%d]\n", DmaBusy);

        return;

    } else { /* Second buffer empty, still data in Dma buffer, continue playb
ack */

        dprint("Starting dma[%d]\n", DmaBusy);

    }

    /* Acknowledge the interrupt on the DSP */
    sb16_inb((DspBits == 8 ? DSP_DATA_AVL : DSP_DATA16_AVL));
}

/*=====
 *                                dsp_status                                *
 *=====*/
PRIVATE void dsp_status(m_ptr)
message *m_ptr; /* pointer to the newly arrived message */
{
    if (revivePending) {
        m_ptr->m_type = DEV_REVIVE; /* build message */
        m_ptr->REP_ENDPT = reviveProcNr;
        m_ptr->REP_STATUS = reviveStatus;

        revivePending = 0; /* unmark event */
    } else {
        m_ptr->m_type = DEV_NO_STATUS;
    }

    send(m_ptr->m_source, m_ptr); /* send the message */
}

/*=====
 *                                reply                                *
 *=====*/
PRIVATE void reply(code, replyee, process, status)
int code;
int replyee;
int process;
int status;
{
    message m;

    m.m_type = code; /* TASK_REPLY or REVIVE */
    m.REP_STATUS = status; /* result of device operation */
    m.REP_ENDPT = process; /* which user made the request */

    send(replyee, &m);
}

/*=====
 *                                init_buffer                                *
 *=====*/

```

```

PRIVATE void init_buffer()
{
    /* Select a buffer that can safely be used for dma transfers.
     * Its absolute address is 'DmaPhys', the normal address is 'DmaPtr'.
     */

    #if (CHIP == INTEL)
        unsigned left;

        DmaPtr = DmaBuffer;
        sys_umap(SELF, D, (vir_bytes)DmaBuffer, (phys_bytes)sizeof(DmaBuffer), &DmaPhys);

        if((left = dma_bytes_left(DmaPhys)) < DMA_SIZE) {
            /* First half of buffer crosses a 64K boundary, can't DMA into that */
            DmaPtr += left;
            DmaPhys += left;
        }
    #else /* CHIP != INTEL */
        panic("SB16DSP", "init_buffer() failed, CHIP != INTEL", 0);
    #endif /* CHIP == INTEL */
}

/*=====
 *
 *                      dsp_init
 *=====*/
PRIVATE int dsp_init()
{
    int i, s;

    if(dsp_reset () != OK) {
        dprint("sb16: No SoundBlaster card detected\n");
        return -1;
    }

    DspVersion[0] = DspVersion[1] = 0;
    dsp_command(DSP_GET_VERSION); /* Get DSP version bytes */

    for(i = 1000; i; i--) {
        if(sb16_inb(DSP_DATA_AVL) & 0x80) {
            if(DspVersion[0] == 0) {
                DspVersion[0] = sb16_inb(DSP_READ);
            } else {
                DspVersion[1] = sb16_inb(DSP_READ);
                break;
            }
        }
    }

    if(DspVersion[0] < 4) {
        dprint("sb16: No SoundBlaster 16 compatible card detected\n");
        return -1;
    }

    dprint("sb16: SoundBlaster DSP version %d.%d detected\n", DspVersion[0], DspVersion[1]);

    /* set SB to use our IRQ and DMA channels */
    mixer_set(MIXER_SET_IRQ, (1 << (SB_IRQ / 2 - 1)));
    mixer_set(MIXER_SET_DMA, (1 << SB_DMA_8 | 1 << SB_DMA_16));

    /* register interrupt vector and enable irq */
    if ((s=sys_irqsetpolicy(SB_IRQ, IRQ_REENABLE, &irq_hook_id)) != OK)
        panic("SB16DSP", "Couldn't set IRQ policy", s);
    if ((s=sys_irgenable(&irq_hook_id)) != OK)
        panic("SB16DSP", "Couldn't enable IRQ", s);

    DspAvail = 1;
    return OK;
}

/*=====
 *
 *                      dsp_reset
 *=====*/

```

```

PRIVATE int dsp_reset()
{
    int i;

    sb16_outb(DSP_RESET, 1);
    for(i = 0; i < 1000; i++); /* wait a while */
    sb16_outb(DSP_RESET, 0);

    for(i = 0; i < 1000 && !(sb16_inb(DSP_DATA_AVL) & 0x80); i++);

    if(sb16_inb(DSP_READ) != 0xAA) return EIO; /* No SoundBlaster */

    DmaBusy = -1;

    return OK;
}

/*=====
 *                               dsp_command
 *=====*/
PRIVATE int dsp_command(value)
int value;
{
    int i, status;

    for (i = 0; i < SB_TIMEOUT; i++) {
        if((sb16_inb(DSP_STATUS) & 0x80) == 0) {
            sb16_outb(DSP_COMMAND, value);
            return OK;
        }
    }

    dprint("sb16: SoundBlaster: DSP Command(%x) timeout\n", value);
    return -1;
}

/*=====
 *                               dsp_set_size
 *=====*/
static int dsp_set_size(size)
unsigned int size;
{
    dprint("dsp_set_size(): set fragment size to %u\n", size);

    /* Sanity checks */
    if(size < DSP_MIN_FRAGMENT_SIZE || size > DSP_MAX_FRAGMENT_SIZE || size % 2 != 0)
    {
        return EINVAL;
    }

    DspFragmentSize = size;

    return OK;
}

/*=====
 *                               dsp_set_speed
 *=====*/
static int dsp_set_speed(speed)
unsigned int speed;
{
    dprint("sb16: setting speed to %u, stereo = %d\n", speed, DspStereo);

    if(speed < DSP_MIN_SPEED || speed > DSP_MAX_SPEED) {
        return EPERM;
    }

    /* Soundblaster 16 can be programmed with real sample rates
     * instead of time constants
     *
     * Since you cannot sample and play at the same time

```

```

    * we set in- and output rate to the same value
    */

    dsp_command(DSP_INPUT_RATE);          /* set input rate */
    dsp_command(speed >> 8);              /* high byte of speed */
    dsp_command(speed);                   /* low byte of speed */
    dsp_command(DSP_OUTPUT_RATE);         /* same for output rate */
    dsp_command(speed >> 8);
    dsp_command(speed);

    DspSpeed = speed;

    return OK;
}

/*=====
 *                      dsp_set_stereo
 *=====*/
static int dsp_set_stereo(stereo)
unsigned int stereo;
{
    if(stereo) {
        DspStereo = 1;
    } else {
        DspStereo = 0;
    }

    return OK;
}

/*=====
 *                      dsp_set_bits
 *=====*/
static int dsp_set_bits(bits)
unsigned int bits;
{
    /* Sanity checks */
    if(bits != 8 && bits != 16) {
        return EINVAL;
    }

    DspBits = bits;

    return OK;
}

/*=====
 *                      dsp_set_sign
 *=====*/
static int dsp_set_sign(sign)
unsigned int sign;
{
    dprint("sb16: set sign to %u\n", sign);

    DspSign = (sign > 0 ? 1 : 0);

    return OK;
}

/*=====
 *                      dsp_dma_setup
 *=====*/
PRIVATE void dsp_dma_setup(address, count)
phys_bytes address;
int count;
{
    pvb_pair_t pvb[9];

    dprint("Setting up %d bit DMA\n", DspBits);

```

```

        if(DspBits == 8) { /* 8 bit sound */
            count--;

            pv_set(pvb[0], DMA8_MASK, SB_DMA_8 | 0x04); /* Disable DMA channel */
            pv_set(pvb[1], DMA8_CLEAR, 0x00); /* Clear flip flop */

            /* set DMA mode */
            pv_set(pvb[2], DMA8_MODE, (DmaMode == DEV_WRITE ? DMA8_AUTO_PLAY : DMA8_A
UTO_REC));

            pv_set(pvb[3], DMA8_ADDR, address >> 0); /* Low_byte of address */
            pv_set(pvb[4], DMA8_ADDR, address >> 8); /* High byte of address */

            pv_set(pvb[5], DMA8_PAGE, address >> 16); /* 64K page number */
            pv_set(pvb[6], DMA8_COUNT, count >> 0); /* Low byte of count */
            pv_set(pvb[7], DMA8_COUNT, count >> 8); /* High byte of count */
            pv_set(pvb[8], DMA8_MASK, SB_DMA_8); /* Enable DMA channel */

            sys_voutb(pvb, 9);
        } else { /* 16 bit sound */
            count-= 2;

            pv_set(pvb[0], DMA16_MASK, (SB_DMA_16 & 3) | 0x04); /* Disable DMA ch
annel */

            pv_set(pvb[1], DMA16_CLEAR, 0x00); /* Clear flip flop */

            /* Set dma mode */
            pv_set(pvb[2], DMA16_MODE, (DmaMode == DEV_WRITE ? DMA16_AUTO_PLAY : DMA1
6_AUTO_REC));

            pv_set(pvb[3], DMA16_ADDR, (address >> 1) & 0xFF); /* Low_byte of addres
s */
            pv_set(pvb[4], DMA16_ADDR, (address >> 9) & 0xFF); /* High byte of addre
ss */
            pv_set(pvb[5], DMA16_PAGE, (address >> 16) & 0xFE); /* 128K page number */
            pv_set(pvb[6], DMA16_COUNT, count >> 1); /* Low byte of count */
            pv_set(pvb[7], DMA16_COUNT, count >> 9); /* High byte of count */
            pv_set(pvb[8], DMA16_MASK, SB_DMA_16 & 3); /* Enable DMA channel */

            sys_voutb(pvb, 9);
        }
    }

/*=====
 *
 *                      dsp_setup()
 *=====*/
PRIVATE void dsp_setup()
{
    /* Set current sample speed */
    dsp_set_speed(DspSpeed);

    /* Put the speaker on */
    if(DmaMode == DEV_WRITE) {
        dsp_command (DSP_CMD_SPKON); /* put speaker on */

        /* Program DSP with dma mode */
        dsp_command((DspBits == 8 ? DSP_CMD_8BITAUTO_OUT : DSP_CMD_16BITAUTO_OUT)
);
    } else {
        dsp_command (DSP_CMD_SPKOFF); /* put speaker off */

        /* Program DSP with dma mode */
        dsp_command((DspBits == 8 ? DSP_CMD_8BITAUTO_IN : DSP_CMD_16BITAUTO_IN));
    }
}

```

```
    }

    /* Program DSP with transfer mode */
    if (!DspSign) {
        dsp_command((DspStereo == 1 ? DSP_MODE_STEREO_US : DSP_MODE_MONO_US));
    } else {
        dsp_command((DspStereo == 1 ? DSP_MODE_STEREO_S : DSP_MODE_MONO_S));
    }

    /* Give length of fragment to DSP */
    if (DspBits == 8) { /* 8 bit transfer */
        /* #bytes - 1 */
        dsp_command((DspFragmentSize - 1) >> 0);
        dsp_command((DspFragmentSize - 1) >> 8);
    } else { /* 16 bit transfer */
        /* #words - 1 */
        dsp_command((DspFragmentSize - 1) >> 1);
        dsp_command((DspFragmentSize - 1) >> 9);
    }
}
```



```

/* This file contains the driver for the mixer on
 * a SoundBlaster 16 soundcard.
 *
 * The driver supports the following operations (using message format m2):
 *
 *      m_type      DEVICE      IO_ENDPT      COUNT      POSITION      ADDRESS
 * -----
 * | DEV_OPEN | device | proc_nr |          |          |          |
 * |-----|
 * | DEV_CLOSE | device | proc_nr |          |          |          |
 * |-----|
 * | DEV_IOCTL | device | proc_nr | func code |          | buf_ptr |
 * |-----|
 *
 * The file contains one entry point:
 *
 *      sb16mixer_task:  main entry when system is brought up
 *
 *      August 24 2005      Ported driver to user space (Peter Boonstoppel)
 *      May 20 1995         Author: Michel R. Prevenier
 */

```

```
#include "sb16.h"
```

```

_PROTOTYPE(void main, (void));
FORWARD _PROTOTYPE( int mixer_init, (void));
FORWARD _PROTOTYPE( int mixer_open, (message *m_ptr));
FORWARD _PROTOTYPE( int mixer_close, (message *m_ptr));
FORWARD _PROTOTYPE( int mixer_ioctl, (message *m_ptr));
FORWARD _PROTOTYPE( int mixer_get, (int reg));
FORWARD _PROTOTYPE( int get_set_volume, (message *m_ptr, int flag));
FORWARD _PROTOTYPE( int get_set_input, (message *m_ptr, int flag, int channel));
FORWARD _PROTOTYPE( int get_set_output, (message *m_ptr, int flag));

```

```
PRIVATE int mixer_avail = 0;    /* Mixer exists? */
```

```
#define dprint (void)
```

```

/*=====
 *
 *                               main
 *=====*/

```

```

PUBLIC void main() {
message mess;
    int err, caller, proc_nr;

    /* Here is the main loop of the mixer task. It waits for a message, carries
     * it out, and sends a reply.
     */
    while (TRUE) {
        receive(ANY, &mess);

        caller = mess.m_source;
        proc_nr = mess.IO_ENDPT;

        switch (caller) {
            case HARDWARE: /* Leftover interrupt. */
                continue;
            case FS_PROC_NR: /* The only legitimate caller. */
                break;
            default:
                dprint("sb16: got message from %d\n", caller);
                continue;
        }

        /* Now carry out the work. */
        switch(mess.m_type) {
            case DEV_OPEN:      err = mixer_open(&mess); break;
            case DEV_CLOSE:    err = mixer_close(&mess); break;
            case DEV_IOCTL:    err = mixer_ioctl(&mess); break;

```

```

        default:                err = EINVAL; break;
    }

    /* Finally, prepare and send the reply message. */
    mess.m_type = TASK_REPLY;
    mess.REP_ENDPT = proc_nr;

    dprint("%d%d", err, OK);

    mess.REP_STATUS = err; /* error code */
    send(caller, &mess); /* send reply to caller */
}

/*=====
 *                               mixer_open
 *=====*/
PRIVATE int mixer_open(m_ptr)
message *m_ptr;
{
    dprint("mixer_open\n");

    /* try to detect the mixer type */
    if (!mixer_avail && mixer_init() != OK) return EIO;

    return OK;
}

/*=====
 *                               mixer_close
 *=====*/
PRIVATE int mixer_close(m_ptr)
message *m_ptr;
{
    dprint("mixer_close\n");

    return OK;
}

/*=====
 *                               mixer_ioctl
 *=====*/
PRIVATE int mixer_ioctl(m_ptr)
message *m_ptr;
{
    int status;

    dprint("mixer: got ioctl %d\n", m_ptr->REQUEST);

    switch(m_ptr->REQUEST) {
        case MIXIOGETVOLUME:      status = get_set_volume(m_ptr, 0); break;
        case MIXIOSETVOLUME:      status = get_set_volume(m_ptr, 1); break;
        case MIXIOGETINPUTLEFT:   status = get_set_input(m_ptr, 0, 0); break;
        case MIXIOGETINPUTRIGHT:  status = get_set_input(m_ptr, 0, 1); break;
        case MIXIOGETOUTPUT:      status = get_set_output(m_ptr, 0); break;
        case MIXIOSETINPUTLEFT:   status = get_set_input(m_ptr, 1, 0); break;
        case MIXIOSETINPUTRIGHT:  status = get_set_input(m_ptr, 1, 1); break;
        case MIXIOSETOUTPUT:      status = get_set_output(m_ptr, 1); break;
        default:                  status = ENOTTY;
    }

    return status;
}

/*=====
 *                               mixer_init
 *=====*/
PRIVATE int mixer_init()
{

```

```

/* Try to detect the mixer by writing to MIXER_DAC_LEVEL if the
 * value written can be read back the mixer is there
 */

mixer_set(MIXER_DAC_LEVEL, 0x10); /* write something to it */
if(mixer_get(MIXER_DAC_LEVEL) != 0x10) {
    dprint("sb16: Mixer not detected\n");
    return EIO;
}

/* Enable Automatic Gain Control */
mixer_set(MIXER_AGC, 0x01);

dprint("Mixer detected\n");

mixer_avail = 1;
return OK;
}

/*=====
 *                               mixer_get
 *=====*/
PRIVATE int mixer_get(reg)
int reg;
{
    int i;

    sb16_outb(MIXER_REG, reg);
    for(i = 0; i < 100; i++);
    return sb16_inb(MIXER_DATA) & 0xff;
}

/*=====
 *                               get_set_volume
 *=====*/
PRIVATE int get_set_volume(m_ptr, flag)
message *m_ptr;
int flag; /* 0 = get, 1 = set */
{
    phys_bytes user_phys;
    struct volume_level level;
    int cmd_left, cmd_right, shift, max_level;

    sys_datacopy(m_ptr->IO_ENDPT, (vir_bytes)m_ptr->ADDRESS, SELF, (vir_bytes)&level,
        (phys_bytes)sizeof(level));

    shift = 3;
    max_level = 0x1F;

    switch(level.device) {
        case Master:
            cmd_left = MIXER_MASTER_LEFT;
            cmd_right = MIXER_MASTER_RIGHT;
            break;
        case Dac:
            cmd_left = MIXER_DAC_LEFT;
            cmd_right = MIXER_DAC_RIGHT;
            break;
        case Fm:
            cmd_left = MIXER_FM_LEFT;
            cmd_right = MIXER_FM_RIGHT;
            break;
        case Cd:
            cmd_left = MIXER_CD_LEFT;
            cmd_right = MIXER_CD_RIGHT;
            break;
        case Line:
            cmd_left = MIXER_LINE_LEFT;
            cmd_right = MIXER_LINE_RIGHT;
            break;
        case Mic:
            cmd_left = cmd_right = MIXER_MIC_LEVEL;
    }
}

```

```

        break;
    case Speaker:
        cmd_left = cmd_right = MIXER_PC_LEVEL;
        shift = 6;
        max_level = 0x03;
        break;
    case Treble:
        cmd_left = MIXER_TREBLE_LEFT;
        cmd_right = MIXER_TREBLE_RIGHT;
        shift = 4;
        max_level = 0x0F;
        break;
    case Bass:
        cmd_left = MIXER_BASS_LEFT;
        cmd_right = MIXER_BASS_RIGHT;
        shift = 4;
        max_level = 0x0F;
        break;
    default:
        return EINVAL;
}

if(flag) { /* Set volume level */
    if(level.right < 0) level.right = 0;
    else if(level.right > max_level) level.right = max_level;
    if(level.left < 0) level.left = 0;
    else if(level.left > max_level) level.left = max_level;

    mixer_set(cmd_right, (level.right << shift));
    mixer_set(cmd_left, (level.left << shift));
} else { /* Get volume level */
    level.left = mixer_get(cmd_left);
    level.right = mixer_get(cmd_right);

    level.left >>= shift;
    level.right >>= shift;

    /* Copy back to user */
    sys_datacopy(SELFS, (vir_bytes)&level, m_ptr->IO_ENDPT, (vir_bytes)m_ptr->
ADDRESS, (phys_bytes)sizeof(level));
}

return OK;
}

/*=====
 *                               get_set_input                               *
 *=====*/
PRIVATE int get_set_input(m_ptr, flag, channel)
message *m_ptr;
int flag; /* 0 = get, 1 = set */
int channel; /* 0 = left, 1 = right */
{
    phys_bytes user_phys;
    struct inout_ctrl input;
    int input_cmd, input_mask, mask, del_mask, shift;

    sys_datacopy(m_ptr->IO_ENDPT, (vir_bytes)m_ptr->ADDRESS, SELFS, (vir_bytes)&input,
(phys_bytes)sizeof(input));

    input_cmd = (channel == 0 ? MIXER_IN_LEFT : MIXER_IN_RIGHT);

    mask = mixer_get(input_cmd);

    switch (input.device) {
        case Fm:
            shift = 5;
            del_mask = 0x1F;
            break;
        case Cd:
            shift = 1;
            del_mask = 0x79;
            break;
    }

```

```

        case Line:
            shift = 3;
            del_mask = 0x67;
            break;
        case Mic:
            shift = 0;
            del_mask = 0x7E;
            break;
        default:
            return EINVAL;
    }

    if (flag) { /* Set input */
        input_mask = ((input.left == ON ? 1 : 0) << 1) | (input.right == ON ? 1 :
0);

        if (shift > 0) input_mask <=< shift;
        else input_mask >=> 1;

        mask &= del_mask;
        mask |= input_mask;

        mixer_set(input_cmd, mask);
    } else { /* Get input */
        if (shift > 0) {
            input.left = ((mask >> (shift+1)) & 1 == 1 ? ON : OFF);
            input.right = ((mask >> shift) & 1 == 1 ? ON : OFF);
        } else {
            input.left = ((mask & 1) == 1 ? ON : OFF);
        }

        /* Copy back to user */
        sys_datacopy(SELFS, (vir_bytes)&input, m_ptr->IO_ENDPT, (vir_bytes)m_ptr->
ADDRESS, (phys_bytes)sizeof(input));
    }

    return OK;
}

/*=====
 *                               get_set_output                               *
 *=====*/
PRIVATE int get_set_output(m_ptr, flag)
message *m_ptr;
int flag; /* 0 = get, 1 = set */
{
    phys_bytes user_phys;
    struct inout_ctrl output;
    int output_mask, mask, del_mask, shift;

    sys_datacopy(m_ptr->IO_ENDPT, (vir_bytes)m_ptr->ADDRESS, SELFS, (vir_bytes)&output
, (phys_bytes)sizeof(output));

    mask = mixer_get(MIXER_OUTPUT_CTRL);

    switch (output.device) {
        case Cd:
            shift = 1;
            del_mask = 0x79;
            break;
        case Line:
            shift = 3;
            del_mask = 0x67;
            break;
        case Mic:
            shift = 0;
            del_mask = 0x7E;
            break;
        default:
            return EINVAL;
    }

    if (flag) { /* Set input */

```

```
        output_mask = ((output.left == ON ? 1 : 0) << 1) | (output.right == ON ?
1 : 0);

        if (shift > 0) output_mask <= shift;
        else output_mask >= 1;

        mask &= del_mask;
        mask |= output_mask;

        mixer_set(MIXER_OUTPUT_CTRL, mask);
    } else {        /* Get input */
        if (shift > 0) {
            output.left = ((mask >> (shift+1)) & 1 == 1 ? ON : OFF);
            output.right = ((mask >> shift) & 1 == 1 ? ON : OFF);
        } else {
            output.left = ((mask & 1) == 1 ? ON : OFF);
        }

        /* Copy back to user */
        sys_datacopy(SELF, (vir_bytes)&output, m_ptr->IO_ENDPT, (vir_bytes)m_ptr-
>ADDRESS, (phys_bytes)sizeof(output));
    }

    return OK;
}
```

```
# Makefile for the Texas Instruments PCI1225 PC Card controller driver (ti1225)
DRIVER = ti1225

# directories
u = /usr
i = $u/include
s = $i/sys
m = $i/minix
b = $i/ibm
d = ..

# programs, flags, etc.
CC =      exec cc
CFLAGS = -I$i
LDFLAGS = -i
LIBS = -lsys -lsysutil -ltimers

OBJ = ti1225.o

# build local binary
all build:      $(DRIVER)
$(DRIVER):      $(OBJ)
                $(CC) -o $@ $(LDFLAGS) $(OBJ) $(LIBS)
                install -S 4096 $(DRIVER)

# install with other drivers
install:      /usr/sbin/$(DRIVER)
/usr/sbin/$(DRIVER):      $(DRIVER)
                install -o root -cs $? $@

# clean up local files
clean:
                rm -f *.o *.bak $(DRIVER)

depend:
                /usr/bin/mkdep "$(CC) -E $(CPPFLAGS)" *.c > .depend

# Include generated dependencies.
include .depend
```

```

/*
i82365.h

Created:      May 1995 by Philip Homburg <philip@cs.vu.nl>
*/

#ifndef I82365_H
#define I82365_H

/* The default I/O ports used by a i82365 are the following: */
#define I365_INDEX      0x3E0
#define I365_DATA       0x3E1

/* The index register is used to select one of the following registers: */
#define I365_REVISION    0x00    /* IDREG */
#define I365R_ID_MASK    0xC0
#define I365R_ID_IO      0x00
#define I365R_ID_MEM     0x40
#define I365R_ID_MEM_IO  0x80
#define I365R_RES_MASK   0x30
#define I365R_REV_MASK   0x0F

#define I365_IF_STAT     0x01    /* ISTAT */
#define I365IS_GPI       0x80
#define CL6722IS_VPPVALID 0x80
#define I365IS_POWER     0x40
#define I365IS_READY     0x20
#define I365IS_WRTPROT   0x10
#define I365IS_CARD_MASK 0x0C
#define I365IS_CARD_ABSENT 0x00
#define I365IS_CARD_PART_0 0x04
#define I365IS_CARD_PART_1 0x08
#define I365IS_CARD_PRESENT 0x0C
#define I365IS_BAT_MASK  0x03
#define I365IS_BAT_LOST_0 0x00
#define I365IS_BAT_LOW   0x01
#define I365IS_BAT_LOST_1 0x02
#define I365IS_BAT_OKAY  0x03

#define I365_PWR_CTL     0x02    /* PCTRL */
#define I365PC_CARD_EN   0x80
#define I365PC_NORESET   0x40
#define CL6722PC_COMPAT_0 0x40
#define I365PC_AUTO_PWR  0x20
#define I365PC_Vcc_MASK  0x18
#define I365PC_Vcc_NC     0x00
#define I365PC_Vcc_Reserved 0x08
#define I365PC_Vcc_5      0x10
#define I365PC_Vcc_33     0x18
#define CL6722PC_Vcc_PWR  0x10
#define CL6722PC_COMPAT_1 0x08
#define I365PC_Vpp_MASK   0x03
#define I365PC_Vpp_NC     0x00
#define CL6722PC_Vpp_ZERO_0 0x00
#define I365PC_Vpp_5      0x01
#define CL6722PC_Vpp_Vcc  0x01
#define I365PC_Vpp_12     0x02
#define I365PC_Vpp_Reserved 0x03
#define CL6722PC_Vpp_ZERO_1 0x03

#define I365_INT_GEN_CTL 0x03
#define I365IGC_RING_IND 0x80
#define I365IGC_RESET    0x40
#define I365IGC_CARD_IS_IO 0x20
#define I365IGC_EN_MNG_INT 0x10
#define I365IGC_IRQ_MASK 0x0F

#define I365_CRD_STAT_CHG 0x04    /* CSTCH */
#define I365CSC_GPI       0x10
#define I365CSC_CARD_DETECT 0x08
#define I365CSC_READY     0x04
#define I365CSC_BAT_WARN  0x02
#define I365CSC_BAT_DEAD  0x01

```



```
#define I365_MNG_INT_CONF      0x05
#define      I365MIC_IRQ_MASK      0xF0
#define      I365MIC_CARD_DETECT    0x08
#define      I365MIC_READY          0x04
#define      I365MIC_BAT_WARN       0x02
#define      I365MIC_BAT_DEAD       0x01

#define I365_MAP_ENABLE        0x06      /* ADWEN */
#define      I365ME_IO_MAP_0        0x40
#define      I365ME_MEM_MAP_0       0x01

#define I365_IO_WND_CTL        0x07
#define      I365IWC_AUTO_1         0x80
#define      CL6722IWC_TIMING_1     0x80
#define      I365IWC_OWS_1         0x40
#define      I365IWC_AUTO_SIZE_1    0x20
#define      I365IWC_IO_SIZE_1     0x10
#define      I365IWC_WAIT_0         0x08
#define      I365IWC_OWS_0          0x04
#define      CL6722IWC_TIMING_0     0x08
#define      I365IWC_AUTO_SIZE_0    0x02
#define      I365IWC_IO_SIZE_0     0x01

#define I365_IO_0_START_LOW    0x08
#define I365_IO_0_START_HIGH   0x09
#define I365_IO_0_END_LOW      0x0A
#define I365_IO_0_END_HIGH     0x0B
#define I365_IO_1_START_LOW    0x0C
#define I365_IO_1_START_HIGH   0x0D
#define I365_IO_1_END_LOW      0x0E
#define I365_IO_1_END_HIGH     0x0F

#define I365_MEM_0_START_LOW    0x10
#define I365_MEM_0_START_HIGH   0x11
#define I365_MEM_0_END_LOW      0x12
#define I365_MEM_0_END_HIGH     0x13
#define I365_MEM_0_OFF_LOW      0x14
#define I365_MEM_0_OFF_HIGH     0x15

#define CL6722_MISC_CTL_1       0x16
#define CL6722_FIFO_CTL         0x17

#define I365_MEM_1_START_LOW    0x18
#define I365_MEM_1_START_HIGH   0x19
#define I365_MEM_1_END_LOW      0x1A
#define I365_MEM_1_END_HIGH     0x1B
#define I365_MEM_1_OFF_LOW      0x1C
#define I365_MEM_1_OFF_HIGH     0x1D

#define CL6722_MISC_CTL_2       0x1E
#define CL6722_CHIP_INFO        0x1F
#define      CL6722CI_ID_MASK      0xC0

#define I365_MEM_2_START_LOW    0x20
#define I365_MEM_2_START_HIGH   0x21
#define I365_MEM_2_END_LOW      0x22
#define I365_MEM_2_END_HIGH     0x23
#define I365_MEM_2_OFF_LOW      0x24
#define I365_MEM_2_OFF_HIGH     0x25

#define CL6722_ATA_CONTROL       0x26      /* CPAGE */
#define I365_RESERVED           0x27

#define I365_MEM_3_START_LOW    0x28
#define I365_MEM_3_START_HIGH   0x29
#define I365_MEM_3_END_LOW      0x2A
#define I365_MEM_3_END_HIGH     0x2B
#define I365_MEM_3_OFF_LOW      0x2C
#define I365_MEM_3_OFF_HIGH     0x2D

#define CL6722_EXT_INDEX         0x2E      /* CSCTRL */
#define CL6722_EXT_DATA          0x2F

#define I365_MEM_4_START_LOW    0x30
```

```
#define I365_MEM_4_START_HIGH    0x31
#define I365_MEM_4_END_LOW      0x32
#define I365_MEM_4_END_HIGH     0x33
#define I365_MEM_4_OFF_LOW      0x34
#define I365_MEM_4_OFF_HIGH     0x35

#define CL6722_IO_0_OFF_LOW      0x36
#define CL6722_IO_0_OFF_HIGH    0x37
#define CL6722_IO_1_OFF_LOW     0x38
#define CL6722_IO_1_OFF_HIGH    0x39

#define I365_SETUP_TIM_0         0x3A
#define I365_CMD_TIM_0           0x3B
#define I365_RECOV_TIM_0         0x3C
#define I365_SETUP_TIM_1         0x3D
#define I365_CMD_TIM_1           0x3E
#define I365_RECOV_TIM_1         0x3F

#endif /* I82365_H */
```

```

/*
ti1225.c

Created:      Dec 2005 by Philip Homburg
*/

#include "../drivers.h"
#include <ibm/pci.h>
#include <sys/vm.h>

#include "ti1225.h"
#include "i82365.h"

/* The use of interrupts is not yet ready for prime time */
#define USE_INTS      0

#define MICROS_TO_TICKS(m)  (((m)*HZ/1000000)+1)

#define NR_PORTS 2

PRIVATE struct port
{
    unsigned p_flags;
    int p_devind;
    u8_t p_cb_busr;
    ul6_t p_exca_port;
#if USE_INTS
    int p_irq;
    int p_hook;
#endif
    char *base_ptr;
    volatile struct csr *csr_ptr;

    char buffer[2*PAGE_SIZE];
} ports[NR_PORTS];

#define PF_PRESENT      1

struct pcitab
{
    ul6_t vid;
    ul6_t did;
    int checkclass;
};

PRIVATE struct pcitab pcitab_ti[]=
{
    { 0x104C, 0xAC1C, 0 },          /* TI PCI1225 */
    { 0x0000, 0x0000, 0 }
};

PRIVATE char *progrname;
PRIVATE int debug;

FORWARD _PROTOTYPE( void init, (void) ) ;
FORWARD _PROTOTYPE( void hw_init, (struct port *pp) ) ;
FORWARD _PROTOTYPE( void map_regs, (struct port *pp, u32_t base) ) ;
FORWARD _PROTOTYPE( void do_int, (struct port *pp) ) ;
FORWARD _PROTOTYPE( u8_t read_exca, (struct port *pp, int socket, int reg) );
FORWARD _PROTOTYPE( void do_outb, (port_t port, u8_t value) ) ;
FORWARD _PROTOTYPE( u8_t do_inb, (port_t port) ) ;
FORWARD _PROTOTYPE( void micro_delay, (unsigned long usecs) ) ;

int main(int argc, char *argv[])
{
    int c, r;
    message m;

    (progrname=strrchr(argv[0], '/')) ? progrname++ : (progrname=argv[0]);

    debug= 0;
    while (c= getopt(argc, argv, "d?"), c != -1)
    {
        switch(c)

```

```

        {
            case '?': panic("ti1225", "Usage: ti1225 [-d]", NO_NUM);
            case 'd': debug++; break;
            default: panic("ti1225", "getopt failed", NO_NUM);
        }
    }

    init();

    for (;;)
    {
        r= receive(ANY, &m);
        if (r != OK)
            panic("ti1225", "receive failed", r);
        printf("ti1225: got message %u from %d\n",
            m.m_type, m.m_source);
    }
    return 0;
}

PRIVATE void init()
{
    int i, r, first, devind, port;
    ul6_t vid, did;

    pci_init1(progname);

    first= 1;
    port= 0;
    for (;;)
    {
        if (first)
        {
            first= 0;
            r= pci_first_dev(&devind, &vid, &did);
        }
        else
            r= pci_next_dev(&devind, &vid, &did);
        if (r != 1)
            break;

        for (i= 0; pcitab_ti[i].vid != 0; i++)
        {
            if (pcitab_ti[i].vid != vid)
                continue;
            if (pcitab_ti[i].did != did)
                continue;
            if (pcitab_ti[i].checkclass)
            {
                panic("ti1225",
                    "fxp_probe: class check not implemented",
                    NO_NUM);
            }
            break;
        }
        if (pcitab_ti[i].vid == 0)
            continue;

        pci_reserve(devind);

        if (debug)
            printf("ti1225: found device %04x/%04x\n", vid, did);
        ports[port].p_devind= devind;
        ports[port].p_flags |= PF_PRESENT;
        port++;
        if (port >= NR_PORTS)
            break;
    }

    for (i= 0; i<NR_PORTS; i++)
    {
        if (!(ports[i].p_flags & PF_PRESENT))
            continue;
        hw_init(&ports[i]);
    }
}

```

```

    }
}

PRIVATE void hw_init(pp)
struct port *pp;
{
    int i, r, devind, irq, socket;
    u8_t v8;
    u16_t v16;
    u32_t v32;

    devind= pp->p_devind;
    if (debug)
        printf("hw_init: devind = %d\n", devind);

    if (debug)
    {
        v16= pci_attr_r16(devind, PCI_CR);
        printf("ti1225: command register 0x%x\n", v16);
    }

    v32= pci_attr_r32(devind, TI_CB_BASEADDR);
    if (debug)
        printf("ti1225: Cardbus/ExCA base address 0x%x\n", v32);
    map_regs(pp, v32);
    pp->csr_ptr= (struct csr *)pp->base_ptr;

    if (debug)
    {
        v8= pci_attr_r8(devind, TI_PCI_BUS_NR);
        printf("ti1225: PCI bus number %d\n", v8);
    }
    v8= pci_attr_r8(devind, TI_CB_BUS_NR);
    pp->p_cb_busr= v8;
    if (debug)
    {
        printf("ti1225: CardBus bus number %d\n", v8);
        v8= pci_attr_r8(devind, TI_SO_BUS_NR);
        printf("ti1225: Subordinate bus number %d\n", v8);
    }

#ifdef USE_INTS
    irq= pci_attr_r8(devind, PCI_ILR);
    pp->p_irq= irq;
    printf("ti1225 using IRQ %d\n", irq);
#endif

    v32= pci_attr_r32(devind, TI_LEGACY_BA);
    v32 &= ~1;
    if (debug)
    {
        printf("ti1225: PC Card 16-bit legacy-mode base address 0x%x\n",
            v32);
    }

    if (v32 == 0)
        panic("ti1225", "bad lagacy-mode base address 0x%x\n", v32);
    pp->p_exca_port= v32;

    if (debug)
    {
        v32= pci_attr_r32(devind, TI_MF_ROUTE);
        printf("ti1225: Multifunction routing 0x%08x\n", v32);
    }

#ifdef USE_INTS
    pp->p_hook = pp->p_irq;
    r= sys_irqsetpolicy(pp->p_irq, 0, &pp->p_hook);
    if (r != OK)
        panic("ti1225", "sys_irqsetpolicy failed", r);
#endif

    /* Clear CBB_BC_INTXCA */
    v16= pci_attr_r16(devind, CBB_BRIDGECTRL);

```

```

    if (debug)
        printf("ti1225: Bridge control 0x%04x\n", v16);
    v16 &= ~CBB_BC_INTEXCA;
    pci_attr_w16(devind, CBB_BRIDGECTRL, v16);

    if (debug)
    {
        v32= pci_attr_r32(devind, TI_SYSCCTRL);
        printf("ti1225: System Control Register 0x%08x\n", v32);

        v8= pci_attr_r8(devind, TI_CARD_CTRL);
        printf("ti1225: Card Control 0x%02x\n", v8);

        v8= pci_attr_r8(devind, TI_DEV_CTRL);
        printf("ti1225: Device Control 0x%02x\n", v8);
    }

    /* Enable socket interrupts */
    pp->csr_ptr->csr_mask |= CM_PWRMASK | CM_CDMASK | CM_CSTSMASK;

    do_int(pp);

#if USE_INTS
    r= sys_irgenable(&pp->p_hook);
    if (r != OK)
        panic("ti1225", "unable enable interrupts", r);
#endif
}

PRIVATE void map_regs(pp, base)
struct port *pp;
u32_t base;
{
    int r;
    vir_bytes buf_base;

    buf_base= (vir_bytes)pp->buffer;
    if (buf_base % PAGE_SIZE)
        buf_base += PAGE_SIZE-(buf_base % PAGE_SIZE);
    pp->base_ptr= (char *)buf_base;
    if (debug)
    {
        printf("ti1225: map_regs: using %p for %p\n",
            pp->base_ptr, pp->buffer);
    }

    /* Clear low order bits in base */
    base &= ~(u32_t)0xF;

    r= sys_vm_map(SELF, 1 /* map */, (vir_bytes)pp->base_ptr,
        PAGE_SIZE, (phys_bytes)base);
    if (r != OK)
        panic("ti1225", "map_regs: sys_vm_map failed", r);
}

PRIVATE void do_int(pp)
struct port *pp;
{
    int i, r, devind, vcc_5v, vcc_3v, vcc_Xv, vcc_Yv,
        socket_5v, socket_3v, socket_Xv, socket_Yv;
    clock_t t0, t1;
    u32_t csr_event, csr_present, csr_control;
    u8_t v8;
    u16_t v16;

    devind= pp->p_devind;
    v8= pci_attr_r8(devind, TI_CARD_CTRL);
    if (v8 & TI_CCR_IFG)
    {
        printf("ti1225: got functional interrupt\n", v8);
        pci_attr_w8(devind, TI_CARD_CTRL, v8);
    }

    if (debug)

```

```

{
    printf("Socket event: 0x%x\n", pp->csr_ptr->csr_event);
    printf("Socket mask: 0x%x\n", pp->csr_ptr->csr_mask);
}

csr_present= pp->csr_ptr->csr_present;
csr_control= pp->csr_ptr->csr_control;

if ((csr_present & (CP_CDETECT1|CP_CDETECT2)) != 0)
{
    if (debug)
        printf("do_int: no card present\n");
    return;
}
if (csr_present & CP_BADVCCREQ)
{
    printf("do_int: Bad Vcc request\n");
    /* return; */
}
if (csr_present & CP_DATALOST)
{
    /* Do we care? */
    if (debug)
        printf("do_int: Data lost\n");
    /* return; */
}
if (csr_present & CP_NOTACARD)
{
    printf("do_int: Not a card\n");
    return;
}
if (debug)
{
    if (csr_present & CP_CBCARD)
        printf("do_int: Cardbus card detected\n");
    if (csr_present & CP_16BITCARD)
        printf("do_int: 16-bit card detected\n");
}
if (csr_present & CP_PWRCYCLE)
{
    if (debug)
        printf("do_int: powered up\n");
    return;
}
vcc_5v= !(csr_present & CP_5VCARD);
vcc_3v= !(csr_present & CP_3VCARD);
vcc_Xv= !(csr_present & CP_XVCARD);
vcc_Yv= !(csr_present & CP_YVCARD);
if (debug)
{
    printf("do_int: card supports:%s%s%s%s\n",
        vcc_5v ? "5V" : "", vcc_3v ? "3V" : "",
        vcc_Xv ? "X.X V" : "", vcc_Yv ? "Y.Y V" : "");
}
socket_5v= !(csr_present & CP_5VSOCKET);
socket_3v= !(csr_present & CP_3VSOCKET);
socket_Xv= !(csr_present & CP_XVSOCKET);
socket_Yv= !(csr_present & CP_YVSOCKET);
if (debug)
{
    printf("do_int: socket supports:%s%s%s%s\n",
        socket_5v ? "5V" : "", socket_3v ? "3V" : "",
        socket_Xv ? "X.X V" : "", socket_Yv ? "Y.Y V" : "");
}
if (vcc_5v && socket_5v)
{
    csr_control= (csr_control & ~CC_VCCCTRL) | CC_VCC_5V;
    pp->csr_ptr->csr_control= csr_control;
    if (debug)
        printf("do_int: applying 5V\n");
}
else if (vcc_3v && socket_3v)
{
    csr_control= (csr_control & ~CC_VCCCTRL) | CC_VCC_3V;

```

```

        pp->csr_ptr->csr_control= csr_control;
        if (debug)
            printf("do_int: applying 3V\n");
    }
    else if (vcc_Xv && socket_Xv)
    {
        csr_control= (csr_control & ~CC_VCCCTRL) | CC_VCC_XV;
        pp->csr_ptr->csr_control= csr_control;
        printf("do_int: applying X.X V\n");
    }
    else if (vcc_Yv && socket_Yv)
    {
        csr_control= (csr_control & ~CC_VCCCTRL) | CC_VCC_YV;
        pp->csr_ptr->csr_control= csr_control;
        printf("do_int: applying Y.Y V\n");
    }
    else
    {
        printf("do_int: socket and card are not compatible\n");
        return;
    }

    csr_event= pp->csr_ptr->csr_event;
    if (csr_event)
    {
        if (debug)
            printf("clearing socket event\n");
        pp->csr_ptr->csr_event= csr_event;
        if (debug)
        {
            printf("Socket event (cleared): 0x%x\n",
                    pp->csr_ptr->csr_event);
        }
    }

    devind= pp->p_devind;
    v8= pci_attr_r8(devind, TI_CARD_CTRL);
    if (v8 & TI_CCR_IFG)
    {
        printf("ti1225: got functional interrupt\n", v8);
        pci_attr_w8(devind, TI_CARD_CTRL, v8);
    }

    if (debug)
    {
        v8= pci_attr_r8(devind, TI_CARD_CTRL);
        printf("TI_CARD_CTRL: 0x%02x\n", v8);
    }

    getuptime(&t0);
    do {
        csr_present= pp->csr_ptr->csr_present;
        if (csr_present & CP_PWRCYCLE)
            break;
    } while (getuptime(&t1)==OK && (t1-t0) < MICROS_TO_TICKS(100000));

    if (!(csr_present & CP_PWRCYCLE))
    {
        printf("do_int: not powered up?\n");
        return;
    }

    /* Reset device */
    v16= pci_attr_r16(devind, CBB_BRIDGECTRL);
    v16 |= CBB_BC_CRST;
    pci_attr_w16(devind, CBB_BRIDGECTRL, v16);

    /* Wait one microsecond. Is this correct? What are the specs? */
    micro_delay(1);

    /* Clear CBB_BC_CRST */
    v16= pci_attr_r16(devind, CBB_BRIDGECTRL);
    v16 &= ~CBB_BC_CRST;
    pci_attr_w16(devind, CBB_BRIDGECTRL, v16);

```



```
    /* Wait one microsecond after clearing the reset line. Is this
     * correct? What are the specs?
     */
    micro_delay(1);

    pci_rescan_bus(pp->p_cb_busr);

#if USE_INTS
    r= sys_irgenable(&pp->p_hook);
    if (r != OK)
        panic("ti1225", "unable enable interrupts", r);
#endif
}

PRIVATE u8_t read_exca(pp, socket, reg)
struct port *pp;
int socket;
int reg;
{
    u16_t port;

    port= pp->p_exca_port;
    if (port == 0)
        panic("ti1225", "read_exca: bad port", NO_NUM);
    do_outb(port, socket * 0x40 + reg);
    return do_inb(port+1);
}

PRIVATE u8_t do_inb(port_t port)
{
    int r;
    u32_t value;

    r= sys_inb(port, &value);
    if (r != OK)
        panic("ti1225", "sys_inb failed", r);
    return value;
}

PRIVATE void do_outb(port_t port, u8_t value)
{
    int r;

    r= sys_outb(port, value);
    if (r != OK)
        panic("ti1225", "sys_outb failed", r);
}

PRIVATE void micro_delay(unsigned long usecs)
{
    tickdelay(MICROS_TO_TICKS(usecs));
}
```

```
/*
ti1225.h

Created:      Dec 2005 by Philip Homburg
*/

/* PCI attribute space registers */
#define TI_CB_BASEADDR    0x10
#define TI_PCI_BUS_NR     0x18
#define TI_CB_BUS_NR      0x19
#define TI_SO_BUS_NR      0x1A
#define TI_LEGACY_BA       0x44
#define TI_SYSCTRL         0x80
#define TI_MF_ROUTE        0x8C
#define TI_CARD_CTRL       0x91
#define TI_CCR_IFG          0x01
#define TI_DEV_CTRL        0x92

/* CardBus Socket Registers */
struct csr
{
/*00*/  u32_t  csr_event;
/*04*/  u32_t  csr_mask;
/*08*/  u32_t  csr_present;
/*0C*/  u32_t  csr_force_event;
/*10*/  u32_t  csr_control;
/*14*/  u32_t  csr_res0;
/*18*/  u32_t  csr_res1;
/*1C*/  u32_t  csr_res2;
/*20*/  u32_t  csr_power;
};

/* csr_mask */
#define CM_PWRMASK        0x00000008
#define CM_CDMASK         0x00000006
#define CM_CSTSMASK       0x00000001

/* csr_present */
#define CP_YVSOCKET        0x80000000
#define CP_XVSOCKET        0x40000000
#define CP_3VSOCKET        0x20000000
#define CP_5VSOCKET        0x10000000
#define CP_YVCARD          0x00002000
#define CP_XVCARD          0x00001000
#define CP_3VCARD          0x00000800
#define CP_5VCARD          0x00000400
#define CP_BADVCCREQ       0x00000200
#define CP_DATA_LOST       0x00000100
#define CP_NOTACARD        0x00000080
#define CP_IREQCINT        0x00000040
#define CP_CBCARD          0x00000020
#define CP_16BITCARD       0x00000010
#define CP_PWRCYCLE         0x00000008
#define CP_CDETECT2        0x00000004
#define CP_CDETECT1        0x00000002
#define CP_CARDSTS         0x00000001

/* csr_control */
#define CC_VCCCTRL          0x00000070
#define CC_VCC_OFF         0x00000000
#define CC_VCC_5V          0x00000020
#define CC_VCC_3V          0x00000030
#define CC_VCC_XV          0x00000040
#define CC_VCC_YV          0x00000050
#define CC_VPPCTRL         0x00000007
#define CC_VPP_OFF         0x00000000
#define CC_VPP_12V         0x00000001
#define CC_VPP_5V          0x00000002
#define CC_VPP_3V          0x00000003
#define CC_VPP_XV          0x00000004
#define CC_VPP_YV          0x00000005
```

```
# Makefile for terminal driver (TTY)
DRIVER = tty

# directories
u = /usr
i = $u/include
s = $i/sys
m = $i/minix
b = $i/ibm
d = ..

# programs, flags, etc.
MAKE = exec make
CC = exec cc
CFLAGS = -I$i
LDFLAGS = -i
LIBS = -lsys -lsysutil -ltimers

OBJ = tty.o console.o vidcopy.o keyboard.o pty.o rs232.o

# build local binary
all build: $(DRIVER)
$(DRIVER): $(OBJ)
    $(CC) -o $@ $(LDFLAGS) $(OBJ) $(LIBS)
    install -S 256w $(DRIVER)

# install with other drivers
install:
    cd keymaps && $(MAKE) -$(MAKEFLAGS) install

# /sbin/$(DRIVER): $(DRIVER)
# install -o root -cs $? $@

# clean up local files
clean:
    cd keymaps && $(MAKE) -$(MAKEFLAGS) $@
    rm -f $(DRIVER) *.o *.bak

depend:
    /usr/bin/mkdep "$(CC) -E $(CPPFLAGS)" *.c > .depend

# Include generated dependencies.
include .depend
```

```

/* Code and data for the IBM console driver.
 *
 * The 6845 video controller used by the IBM PC shares its video memory with
 * the CPU somewhere in the 0xB0000 memory bank. To the 6845 this memory
 * consists of 16-bit words. Each word has a character code in the low byte
 * and a so-called attribute byte in the high byte. The CPU directly modifies
 * video memory to display characters, and sets two registers on the 6845 that
 * specify the video origin and the cursor position. The video origin is the
 * place in video memory where the first character (upper left corner) can
 * be found. Moving the origin is a fast way to scroll the screen. Some
 * video adapters wrap around the top of video memory, so the origin can
 * move without bounds. For other adapters screen memory must sometimes be
 * moved to reset the origin. All computations on video memory use character
 * (word) addresses for simplicity and assume there is no wrapping. The
 * assembly support functions translate the word addresses to byte addresses
 * and the scrolling function worries about wrapping.
 */

#include "../drivers.h"
#include <termios.h>
#include <sys/ioctl.h>
#include <sys/vm.h>
#include <minix/callnr.h>
#include <minix/com.h>
#include "tty.h"

#include "../kernel/const.h"
#include "../kernel/config.h"
#include "../kernel/type.h"

/* Definitions used by the console driver. */
#define MONO_BASE 0xB0000L /* base of mono video memory */
#define COLOR_BASE 0xB8000L /* base of color video memory */
#define MONO_SIZE 0x1000 /* 4K mono video memory */
#define COLOR_SIZE 0x4000 /* 16K color video memory */
#define EGA_SIZE 0x8000 /* EGA & VGA have at least 32K */
#define BLANK_COLOR 0x0700 /* determines cursor color on blank screen */
#define SCROLL_UP 0 /* scroll forward */
#define SCROLL_DOWN 1 /* scroll backward */
#define BLANK_MEM ((u16_t *) 0) /* tells mem_vid_copy() to blank the screen */
#define CONS_RAM_WORDS 80 /* video ram buffer size */
#define MAX_ESC_PARMS 4 /* number of escape sequence params allowed */

/* Constants relating to the controller chips. */
#define M_6845 0x3B4 /* port for 6845 mono */
#define C_6845 0x3D4 /* port for 6845 color */
#define INDEX 0 /* 6845's index register */
#define DATA 1 /* 6845's data register */
#define STATUS 6 /* 6845's status register */
#define VID_ORG 12 /* 6845's origin register */
#define CURSOR 14 /* 6845's cursor register */

/* The clock task should provide an interface for this */
#define TIMER_FREQ 1193182L /* clock frequency for timer in PC and AT */

/* Beeper. */
#define BEEP_FREQ 0x0533 /* value to put into timer to set beep freq */
#define B_TIME 3 /* length of CTRL-G beep is ticks */

/* definitions used for font management */
#define GA_SEQUENCER_INDEX 0x3C4
#define GA_SEQUENCER_DATA 0x3C5
#define GA_GRAPHICS_INDEX 0x3CE
#define GA_GRAPHICS_DATA 0x3CF
#define GA_VIDEO_ADDRESS 0xA0000L
#define GA_FONT_SIZE 8192

/* Global variables used by the console driver and assembly support. */
PUBLIC int vid_index; /* index of video segment in remote mem map */
PUBLIC u16_t vid_seg;
PUBLIC vir_bytes vid_off; /* video ram is found at vid_seg:vid_off */
PUBLIC unsigned vid_size; /* 0x2000 for color or 0x0800 for mono */
PUBLIC unsigned vid_mask; /* 0x1FFF for color or 0x07FF for mono */
PUBLIC unsigned blank_color = BLANK_COLOR; /* display code for blank */

```

```

/* Private variables used by the console driver. */
PRIVATE int vid_port;          /* I/O port for accessing 6845 */
PRIVATE int wrap;              /* hardware can wrap? */
PRIVATE int softscroll;        /* 1 = software scrolling, 0 = hardware */
PRIVATE int beeping;           /* speaker is beeping? */
PRIVATE unsigned font_lines;   /* font lines per character */
PRIVATE unsigned scr_width;    /* # characters on a line */
PRIVATE unsigned scr_lines;    /* # lines on the screen */
PRIVATE unsigned scr_size;     /* # characters on the screen */

/* Per console data. */
typedef struct console {
    tty_t *c_tty;              /* associated TTY struct */
    int c_column;              /* current column number (0-origin) */
    int c_row;                 /* current row (0 at top of screen) */
    int c_rwords;              /* number of WORDS (not bytes) in outqueue */
    unsigned c_start;          /* start of video memory of this console */
    unsigned c_limit;          /* limit of this console's video memory */
    unsigned c_org;            /* location in RAM where 6845 base points */
    unsigned c_cur;            /* current position of cursor in video RAM */
    unsigned c_attr;           /* character attribute */
    unsigned c_blank;          /* blank attribute */
    char c_reverse;            /* reverse video */
    char c_esc_state;          /* 0=normal, 1=ESC, 2=ESC[ */
    char c_esc_intro;          /* Distinguishing character following ESC */
    int *c_esc_parmp;          /* pointer to current escape parameter */
    int c_esc_parmv[MAX_ESC_PARMS]; /* list of escape parameters */
    ul6_t c_ramqueue[CONS_RAM_WORDS]; /* buffer for video RAM */
} console_t;

PRIVATE int nr_cons = 1;       /* actual number of consoles */
PRIVATE console_t cons_table[NR_CONS];
PRIVATE console_t *curcons;    /* currently visible */

/* Color if using a color controller. */
#define color (vid_port == C_6845)

/* Map from ANSI colors to the attributes used by the PC */
PRIVATE int ansi_colors[8] = {0, 4, 2, 6, 1, 5, 3, 7};

/* Structure used for font management */
struct sequence {
    unsigned short index;
    unsigned char port;
    unsigned char value;
};

FORWARD _PROTOTYPE( int cons_write, (struct tty *tp, int try) );
FORWARD _PROTOTYPE( void cons_echo, (tty_t *tp, int c) );
FORWARD _PROTOTYPE( void out_char, (console_t *cons, int c) );
FORWARD _PROTOTYPE( void cons_putk, (int c) );
FORWARD _PROTOTYPE( void beep, (void) );
FORWARD _PROTOTYPE( void do_escape, (console_t *cons, int c) );
FORWARD _PROTOTYPE( void flush, (console_t *cons) );
FORWARD _PROTOTYPE( void parse_escape, (console_t *cons, int c) );
FORWARD _PROTOTYPE( void scroll_screen, (console_t *cons, int dir) );
FORWARD _PROTOTYPE( void set_6845, (int reg, unsigned val) );
FORWARD _PROTOTYPE( void get_6845, (int reg, unsigned *val) );
FORWARD _PROTOTYPE( void stop_beep, (timer_t *tmrp) );
FORWARD _PROTOTYPE( void cons_org0, (void) );
FORWARD _PROTOTYPE( int ga_program, (struct sequence *seq) );
FORWARD _PROTOTYPE( int cons_ioctl, (tty_t *tp, int) );
PRIVATE _PROTOTYPE( void ser_putc, (char c) );

/*=====
 *
 * cons_write
 *=====*/
PRIVATE int cons_write(tp, try)
register struct tty *tp;      /* tells which terminal is to be used */
int try;
{
    /* Copy as much data as possible to the output queue, then start I/O. On
     * memory-mapped terminals, such as the IBM console, the I/O will also be

```

```

* finished, and the counts updated. Keep repeating until all I/O done.
*/

int count;
int result;
register char *tbuf;
char buf[64];
console_t *cons = tp->tty_priv;

if (try) return 1; /* we can always write to console */

/* Check quickly for nothing to do, so this can be called often without
 * unmodular tests elsewhere.
 */
if ((count = tp->tty_outleft) == 0 || tp->tty_inhibited) return;

/* Copy the user bytes to buf[] for decent addressing. Loop over the
 * copies, since the user buffer may be much larger than buf[].
 */
do {
    if (count > sizeof(buf)) count = sizeof(buf);
    if ((result = sys_vircopy(tp->tty_outproc, D, tp->tty_out_vir,
                             SELF, D, (vir_bytes) buf, (vir_bytes) count)) != OK)
        break;
    tbuf = buf;

    /* Update terminal data structure. */
    tp->tty_out_vir += count;
    tp->tty_outcum += count;
    tp->tty_outleft -= count;

    /* Output each byte of the copy to the screen. Avoid calling
     * out_char() for the "easy" characters, put them into the buffer
     * directly.
     */
    do {
        if ((unsigned) *tbuf < ' ' || cons->c_esc_state > 0
            || cons->c_column >= scr_width
            || cons->c_rwords >= buflen(cons->c_ramqueue))
        {
            out_char(cons, *tbuf++);
        } else {
            cons->c_ramqueue[cons->c_rwords++] =
                cons->c_attr | (*tbuf++ & BYTE);
            cons->c_column++;
        }
    } while (--count != 0);
} while ((count = tp->tty_outleft) != 0 && !tp->tty_inhibited);

flush(cons); /* transfer anything buffered to the screen */

/* Reply to the writer if all output is finished or if an error occurred. */
if (tp->tty_outleft == 0 || result != OK) {
    /* REVIVE is not possible. I/O on memory mapped consoles finishes. */
    tty_reply(tp->tty_outrepcode, tp->tty_outcaller, tp->tty_outproc,
              tp->tty_outcum);
    tp->tty_outcum = 0;
}
}

/*=====
 *                               cons_echo                               *
 *=====*/
PRIVATE void cons_echo(tp, c)
register tty_t *tp; /* pointer to tty struct */
int c; /* character to be echoed */
{
    /* Echo keyboard input (print & flush). */
    console_t *cons = tp->tty_priv;

    out_char(cons, c);
    flush(cons);
}

```

```

/*=====
 *                               out_char                               *
 *=====*/
PRIVATE void out_char(cons, c)
register console_t *cons;      /* pointer to console struct */
int c;                        /* character to be output */
{
    /* Output a character on the console. Check for escape sequences first. */
    if (cons->c_esc_state > 0) {
        parse_escape(cons, c);
        return;
    }

    switch(c) {
        case 000:                /* null is typically used for padding */
            return;              /* better not do anything */

        case 007:                /* ring the bell */
            flush(cons);         /* print any chars queued for output */
            beep();
            return;

        case '\b':               /* backspace */
            if (--cons->c_column < 0) {
                if (--cons->c_row >= 0) cons->c_column += scr_width;
            }
            flush(cons);
            return;

        case '\n':               /* line feed */
            if ((cons->c_tty->tty_termios.c_oflag & (OPOST|ONLCR))
                == (OPOST|ONLCR)) {
                cons->c_column = 0;
            }
            /*FALL THROUGH*/

        case 013:                /* CTRL-K */
        case 014:                /* CTRL-L */
            if (cons->c_row == scr_lines-1) {
                scroll_screen(cons, SCROLL_UP);
            } else {
                cons->c_row++;
            }
            flush(cons);
            return;

        case '\r':               /* carriage return */
            cons->c_column = 0;
            flush(cons);
            return;

        case '\t':               /* tab */
            cons->c_column = (cons->c_column + TAB_SIZE) & ~TAB_MASK;
            if (cons->c_column > scr_width) {
                cons->c_column -= scr_width;
                if (cons->c_row == scr_lines-1) {
                    scroll_screen(cons, SCROLL_UP);
                } else {
                    cons->c_row++;
                }
            }
            flush(cons);
            return;

        case 033:                /* ESC - start of an escape sequence */
            flush(cons);         /* print any chars queued for output */
            cons->c_esc_state = 1; /* mark ESC as seen */
            return;

        default:                 /* printable chars are stored in ramqueue */
            if (cons->c_column >= scr_width) {
                if (!LINEWRAP) return;
                if (cons->c_row == scr_lines-1) {
                    scroll_screen(cons, SCROLL_UP);
                } else {

```

```

        cons->c_row++;
    }
    cons->c_column = 0;
    flush(cons);
}
if (cons->c_rwords == buflen(cons->c_ramqueue)) flush(cons);
cons->c_ramqueue[cons->c_rwords++] = cons->c_attr | (c & BYTE);
cons->c_column++; /* next column */
return;
}
}

/*=====
 *                               scroll_screen                               *
 *=====*/
PRIVATE void scroll_screen(cons, dir)
register console_t *cons; /* pointer to console struct */
int dir; /* SCROLL_UP or SCROLL_DOWN */
{
    unsigned new_line, new_org, chars;

    flush(cons);
    chars = scr_size - scr_width; /* one screen minus one line */

    /* Scrolling the screen is a real nuisance due to the various incompatible
     * video cards. This driver supports software scrolling (Hercules?),
     * hardware scrolling (mono and CGA cards) and hardware scrolling without
     * wrapping (EGA cards). In the latter case we must make sure that
     * c_start <= c_org && c_org + scr_size <= c_limit
     * holds, because EGA doesn't wrap around the end of video memory.
     */
    if (dir == SCROLL_UP) {
        /* Scroll one line up in 3 ways: soft, avoid wrap, use origin. */
        if (softscroll) {
            vid_vid_copy(cons->c_start + scr_width, cons->c_start, chars);
        } else
        if (!wrap && cons->c_org + scr_size + scr_width >= cons->c_limit) {
            vid_vid_copy(cons->c_org + scr_width, cons->c_start, chars);
            cons->c_org = cons->c_start;
        } else {
            cons->c_org = (cons->c_org + scr_width) & vid_mask;
        }
        new_line = (cons->c_org + chars) & vid_mask;
    } else {
        /* Scroll one line down in 3 ways: soft, avoid wrap, use origin. */
        if (softscroll) {
            vid_vid_copy(cons->c_start, cons->c_start + scr_width, chars);
        } else
        if (!wrap && cons->c_org < cons->c_start + scr_width) {
            new_org = cons->c_limit - scr_size;
            vid_vid_copy(cons->c_org, new_org + scr_width, chars);
            cons->c_org = new_org;
        } else {
            cons->c_org = (cons->c_org - scr_width) & vid_mask;
        }
        new_line = cons->c_org;
    }
    /* Blank the new line at top or bottom. */
    blank_color = cons->c_blank;
    mem_vid_copy(BLANK_MEM, new_line, scr_width);

    /* Set the new video origin. */
    if (cons == curcons) set_6845(VID_ORG, cons->c_org);
    flush(cons);
}

/*=====
 *                               flush                               *
 *=====*/
PRIVATE void flush(cons)
register console_t *cons; /* pointer to console struct */
{
    /* Send characters buffered in 'ramqueue' to screen memory, check the new
     * cursor position, compute the new hardware cursor position and set it.

```



```

*/
unsigned cur;
tty_t *tp = cons->c_tty;

/* Have the characters in 'ramqueue' transferred to the screen. */
if (cons->c_rwords > 0) {
    mem_vid_copy(cons->c_ramqueue, cons->c_cur, cons->c_rwords);
    cons->c_rwords = 0;

    /* TTY likes to know the current column and if echoing messed up. */
    tp->tty_position = cons->c_column;
    tp->tty_reprint = TRUE;
}

/* Check and update the cursor position. */
if (cons->c_column < 0) cons->c_column = 0;
if (cons->c_column > scr_width) cons->c_column = scr_width;
if (cons->c_row < 0) cons->c_row = 0;
if (cons->c_row >= scr_lines) cons->c_row = scr_lines - 1;
cur = cons->c_org + cons->c_row * scr_width + cons->c_column;
if (cur != cons->c_cur) {
    if (cons == curcons) set_6845(CURSORM, cur);
    cons->c_cur = cur;
}
}

/*=====
*                                     parse_escape
*=====*/
PRIVATE void parse_escape(cons, c)
register console_t *cons;    /* pointer to console struct */
char c;                     /* next character in escape sequence */
{
    /* The following ANSI escape sequences are currently supported.
    * If n and/or m are omitted, they default to 1.
    * ESC [nA moves up n lines
    * ESC [nB moves down n lines
    * ESC [nC moves right n spaces
    * ESC [nD moves left n spaces
    * ESC [m;nH" moves cursor to (m,n)
    * ESC [J clears screen from cursor
    * ESC [K clears line from cursor
    * ESC [nL inserts n lines at cursor
    * ESC [nM deletes n lines at cursor
    * ESC [nP deletes n chars at cursor
    * ESC [n@ inserts n chars at cursor
    * ESC [nm enables rendition n (0=normal, 4=bold, 5=blinking, 7=reverse)
    * ESC M scrolls the screen backwards if the cursor is on the top line
    */

    switch (cons->c_esc_state) {
        case 1: /* ESC seen */
            cons->c_esc_intro = '\0';
            cons->c_esc_parmp = bufend(cons->c_esc_parmv);
            do {
                *--cons->c_esc_parmp = 0;
            } while (cons->c_esc_parmp > cons->c_esc_parmv);
            switch (c) {
                case '[': /* Control Sequence Introducer */
                    cons->c_esc_intro = c;
                    cons->c_esc_state = 2;
                    break;
                case 'M': /* Reverse Index */
                    do_escape(cons, c);
                    break;
                default:
                    cons->c_esc_state = 0;
            }
            break;

        case 2: /* ESC [ seen */
            if (c >= '0' && c <= '9') {
                if (cons->c_esc_parmp < bufend(cons->c_esc_parmv))
                    *cons->c_esc_parmp = *cons->c_esc_parmp * 10 + (c-'0');
            }
    }
}

```

```

    } else
    if (c == ';' ) {
        if (cons->c_esc_parmp < bufend(cons->c_esc_parmv))
            cons->c_esc_parmp++;
    } else {
        do_escape(cons, c);
    }
    break;
}
}

/*=====
 *                               do_escape                               *
 *=====*/
PRIVATE void do_escape(cons, c)
register console_t *cons;    /* pointer to console struct */
char c;                    /* next character in escape sequence */
{
    int value, n;
    unsigned src, dst, count;
    int *parmp;

    /* Some of these things hack on screen RAM, so it had better be up to date */
    flush(cons);

    if (cons->c_esc_intro == '\0') {
        /* Handle a sequence beginning with just ESC */
        switch (c) {
            case 'M':          /* Reverse Index */
                if (cons->c_row == 0) {
                    scroll_screen(cons, SCROLL_DOWN);
                } else {
                    cons->c_row--;
                }
                flush(cons);
                break;

            default: break;
        }
    } else
    if (cons->c_esc_intro == '[') {
        /* Handle a sequence beginning with ESC [ and parameters */
        value = cons->c_esc_parmv[0];
        switch (c) {
            case 'A':          /* ESC [nA moves up n lines */
                n = (value == 0 ? 1 : value);
                cons->c_row -= n;
                flush(cons);
                break;

            case 'B':          /* ESC [nB moves down n lines */
                n = (value == 0 ? 1 : value);
                cons->c_row += n;
                flush(cons);
                break;

            case 'C':          /* ESC [nC moves right n spaces */
                n = (value == 0 ? 1 : value);
                cons->c_column += n;
                flush(cons);
                break;

            case 'D':          /* ESC [nD moves left n spaces */
                n = (value == 0 ? 1 : value);
                cons->c_column -= n;
                flush(cons);
                break;

            case 'H':          /* ESC [m;nH" moves cursor to (m,n) */
                cons->c_row = cons->c_esc_parmv[0] - 1;
                cons->c_column = cons->c_esc_parmv[1] - 1;
                flush(cons);
                break;
        }
    }
}

```

```

case 'J':          /* ESC [sJ clears in display */
    switch (value) {
        case 0:      /* Clear from cursor to end of screen */
            count = scr_size - (cons->c_cur - cons->c_org);
            dst = cons->c_cur;
            break;
        case 1:      /* Clear from start of screen to cursor */
            count = cons->c_cur - cons->c_org;
            dst = cons->c_org;
            break;
        case 2:      /* Clear entire screen */
            count = scr_size;
            dst = cons->c_org;
            break;
        default:      /* Do nothing */
            count = 0;
            dst = cons->c_org;
    }
    blank_color = cons->c_blank;
    mem_vid_copy(BLANK_MEM, dst, count);
    break;

case 'K':          /* ESC [sK clears line from cursor */
    switch (value) {
        case 0:      /* Clear from cursor to end of line */
            count = scr_width - cons->c_column;
            dst = cons->c_cur;
            break;
        case 1:      /* Clear from beginning of line to cursor */
            count = cons->c_column;
            dst = cons->c_cur - cons->c_column;
            break;
        case 2:      /* Clear entire line */
            count = scr_width;
            dst = cons->c_cur - cons->c_column;
            break;
        default:      /* Do nothing */
            count = 0;
            dst = cons->c_cur;
    }
    blank_color = cons->c_blank;
    mem_vid_copy(BLANK_MEM, dst, count);
    break;

case 'L':          /* ESC [nL inserts n lines at cursor */
    n = value;
    if (n < 1) n = 1;
    if (n > (scr_lines - cons->c_row))
        n = scr_lines - cons->c_row;

    src = cons->c_org + cons->c_row * scr_width;
    dst = src + n * scr_width;
    count = (scr_lines - cons->c_row - n) * scr_width;
    vid_vid_copy(src, dst, count);
    blank_color = cons->c_blank;
    mem_vid_copy(BLANK_MEM, src, n * scr_width);
    break;

case 'M':          /* ESC [nM deletes n lines at cursor */
    n = value;
    if (n < 1) n = 1;
    if (n > (scr_lines - cons->c_row))
        n = scr_lines - cons->c_row;

    dst = cons->c_org + cons->c_row * scr_width;
    src = dst + n * scr_width;
    count = (scr_lines - cons->c_row - n) * scr_width;
    vid_vid_copy(src, dst, count);
    blank_color = cons->c_blank;
    mem_vid_copy(BLANK_MEM, dst + count, n * scr_width);
    break;

case '@':          /* ESC [n@ inserts n chars at cursor */
    n = value;

```

```

    if (n < 1) n = 1;
    if (n > (scr_width - cons->c_column))
        n = scr_width - cons->c_column;

    src = cons->c_cur;
    dst = src + n;
    count = scr_width - cons->c_column - n;
    vid_vid_copy(src, dst, count);
    blank_color = cons->c_blank;
    mem_vid_copy(BLANK_MEM, src, n);
    break;

case 'P':          /* ESC [nP deletes n chars at cursor */
    n = value;
    if (n < 1) n = 1;
    if (n > (scr_width - cons->c_column))
        n = scr_width - cons->c_column;

    dst = cons->c_cur;
    src = dst + n;
    count = scr_width - cons->c_column - n;
    vid_vid_copy(src, dst, count);
    blank_color = cons->c_blank;
    mem_vid_copy(BLANK_MEM, dst + count, n);
    break;

case 'm':          /* ESC [nm enables rendition n */
    for (parmp = cons->c_esc_parmv; parmp <= cons->c_esc_parmp
        && parmp < bufend(cons->c_esc_parmv); parmp++) {
        if (cons->c_reverse) {
            /* Unswap fg and bg colors */
            cons->c_attr = ((cons->c_attr & 0x7000) >> 4) |
                           ((cons->c_attr & 0x0700) << 4) |
                           ((cons->c_attr & 0x8800));
        }
        switch (n = *parmp) {
            case 0:      /* NORMAL */
                cons->c_attr = cons->c_blank = BLANK_COLOR;
                cons->c_reverse = FALSE;
                break;

            case 1:      /* BOLD */
                /* Set intensity bit */
                cons->c_attr |= 0x0800;
                break;

            case 4:      /* UNDERLINE */
                if (color) {
                    /* Change white to cyan, i.e. lose red */
                    /*
                     */
                    cons->c_attr = (cons->c_attr & 0xBBFF);
                } else {
                    /* Set underline attribute */
                    cons->c_attr = (cons->c_attr & 0x99FF);
                }
                break;

            case 5:      /* BLINKING */
                /* Set the blink bit */
                cons->c_attr |= 0x8000;
                break;

            case 7:      /* REVERSE */
                cons->c_reverse = TRUE;
                break;

            default:      /* COLOR */
                if (n == 39) n = 37;      /* set default color */
                if (n == 49) n = 40;

                if (!color) {
                    /* Don't mess up a monochrome screen */
                } else
                    if (30 <= n && n <= 37) {

```

```

        /* Foreground color */
        cons->c_attr =
            (cons->c_attr & 0xF8FF) |
            (ansi_colors[(n - 30)] << 8);
        cons->c_blank =
            (cons->c_blank & 0xF8FF) |
            (ansi_colors[(n - 30)] << 8);
    } else
    if (40 <= n && n <= 47) {
        /* Background color */
        cons->c_attr =
            (cons->c_attr & 0x8FFF) |
            (ansi_colors[(n - 40)] << 12);
        cons->c_blank =
            (cons->c_blank & 0x8FFF) |
            (ansi_colors[(n - 40)] << 12);
    }
}
if (cons->c_reverse) {
    /* Swap fg and bg colors */
    cons->c_attr = ((cons->c_attr & 0x7000) >> 4) |
                  ((cons->c_attr & 0x0700) << 4) |
                  ((cons->c_attr & 0x8800));
}
break;
}
}
cons->c_esc_state = 0;
}

/*=====
 *                               set_6845                               *
 *=====*/
PRIVATE void set_6845(reg, val)
int reg;                /* which register pair to set */
unsigned val;           /* 16-bit value to set it to */
{
    /* Set a register pair inside the 6845.
     * Registers 12-13 tell the 6845 where in video ram to start
     * Registers 14-15 tell the 6845 where to put the cursor
     */
    pvb_pair_t char_out[4];
    pv_set(char_out[0], vid_port + INDEX, reg); /* set index register */
    pv_set(char_out[1], vid_port + DATA, (val>>8) & BYTE); /* high byte */
    pv_set(char_out[2], vid_port + INDEX, reg + 1); /* again */
    pv_set(char_out[3], vid_port + DATA, val&BYTE); /* low byte */
    sys_voutb(char_out, 4); /* do actual output */
}

/*=====
 *                               get_6845                               *
 *=====*/
PRIVATE void get_6845(reg, val)
int reg;                /* which register pair to set */
unsigned *val;           /* 16-bit value to set it to */
{
    char v1, v2;
    unsigned long v;
    /* Get a register pair inside the 6845. */
    sys_outb(vid_port + INDEX, reg);
    sys_inb(vid_port + DATA, &v);
    v1 = v;
    sys_outb(vid_port + INDEX, reg+1);
    sys_inb(vid_port + DATA, &v);
    v2 = v;
    *val = (v1 << 8) | v2;
}

/*=====
 *                               beep                               *
 *=====*/
PRIVATE void beep()
{

```

```

/* Making a beeping sound on the speaker (output for CTRL-G).
 * This routine works by turning on the bits 0 and 1 in port B of the 8255
 * chip that drive the speaker.
 */
static timer_t tmr_stop_beep;
pvb_pair_t char_out[3];
clock_t now;
unsigned long port_b_val;
int s;

/* Fetch current time in advance to prevent beeping delay. */
if ((s=getuptime(&now)) != OK)
    panic("TTY", "Console couldn't get clock's uptime.", s);
if (!beeping) {
    /* Set timer channel 2, square wave, with given frequency. */
    pv_set(char_out[0], TIMER_MODE, 0xB6);
    pv_set(char_out[1], TIMER2, (BEEP_FREQ >> 0) & BYTE);
    pv_set(char_out[2], TIMER2, (BEEP_FREQ >> 8) & BYTE);
    if (sys_voutb(char_out, 3) == OK) {
        if (sys_inb(PORT_B, &port_b_val) == OK &&
            sys_outb(PORT_B, (port_b_val|3)) == OK)
            beeping = TRUE;
    }
}
/* Add a timer to the timers list. Possibly reschedule the alarm. */
tmrs_settimer(&tty_timers, &tmr_stop_beep, now+B_TIME, stop_beep, NULL);
if (tty_timers->tmr_exp_time != tty_next_timeout) {
    tty_next_timeout = tty_timers->tmr_exp_time;
    if ((s=sys_setalarm(tty_next_timeout, 1)) != OK)
        panic("TTY", "Console couldn't set alarm.", s);
}
}

/*=====
 * do_video
 *=====*/
PUBLIC void do_video(message *m)
{
    int i, n, r, ops, watch;
    unsigned char c;

    /* Execute the requested device driver function. */
    r = EINVAL; /* just in case */
    switch (m->m_type) {
        case DEV_OPEN:
            /* Should grant IOPL */
            r = OK;
            break;
        case DEV_CLOSE:
            r = OK;
            break;
        case DEV_IOCTL:
            if (m->TTY_REQUEST == MIOCMAP || m->TTY_REQUEST == MIOCUNMAP)
            {
                int r, do_map;
                struct mapreq mapreq;

                do_map = (m->REQUEST == MIOCMAP); /* else unmap */

                /* Get request structure */
                r = sys_vircopy(m->IO_ENDPT, D,
                    (vir_bytes)m->ADDRESS,
                    SELF, D, (vir_bytes)&mapreq, sizeof(mapreq));
                if (r != OK)
                {
                    tty_reply(TASK_REPLY, m->m_source, m->IO_ENDPT,
                        r);
                    return;
                }
                r = sys_vm_map(m->IO_ENDPT, do_map,
                    (phys_bytes)mapreq.base, mapreq.size,
                    mapreq.offset);
                tty_reply(TASK_REPLY, m->m_source, m->IO_ENDPT, r);
            }
    }
}

```

```

        return;
    }
    r= ENOTTY;
    break;

default:
    printf(
        "Warning, TTY(video) got unexpected request %d from %d\n",
        m->m_type, m->m_source);
    r= EINVAL;
}
tty_reply(TASK_REPLY, m->m_source, m->IO_ENDPT, r);
}

/*=====
*
*                      beep_x
*=====*/
PUBLIC void beep_x(freq, dur)
unsigned freq;
clock_t dur;
{
    /* Making a beeping sound on the speaker.
     * This routine works by turning on the bits 0 and 1 in port B of the 8255
     * chip that drive the speaker.
     */
    static timer_t tmr_stop_beep;
    pvb_pair_t char_out[3];
    clock_t now;
    unsigned long port_b_val;
    int s;

    unsigned long ival= TIMER_FREQ / freq;
    if (ival == 0 || ival > 0xffff)
        return; /* Frequency out of range */

    /* Fetch current time in advance to prevent beeping delay. */
    if ((s=getuptime(&now)) != OK)
        panic("TTY", "Console couldn't get clock's uptime.", s);
    if (!beeping) {
        /* Set timer channel 2, square wave, with given frequency. */
        pv_set(char_out[0], TIMER_MODE, 0xB6);
        pv_set(char_out[1], TIMER2, (ival >> 0) & BYTE);
        pv_set(char_out[2], TIMER2, (ival >> 8) & BYTE);
        if (sys_voutb(char_out, 3)==OK) {
            if (sys_inb(PORT_B, &port_b_val)==OK &&
                sys_outb(PORT_B, (port_b_val|3))==OK)
                beeping = TRUE;
        }
    }
    /* Add a timer to the timers list. Possibly reschedule the alarm. */
    tmrs_settimer(&tty_timers, &tmr_stop_beep, now+dur, stop_beep, NULL);
    if (tty_timers->tmr_exp_time != tty_next_timeout) {
        tty_next_timeout = tty_timers->tmr_exp_time;
        if ((s=sys_setalarm(tty_next_timeout, 1)) != OK)
            panic("TTY", "Console couldn't set alarm.", s);
    }
}

/*=====
*
*                      stop_beep
*=====*/
PRIVATE void stop_beep(tmrp)
timer_t *tmrp;
{
    /* Turn off the beeper by turning off bits 0 and 1 in PORT_B. */
    unsigned long port_b_val;
    if (sys_inb(PORT_B, &port_b_val)==OK &&
        sys_outb(PORT_B, (port_b_val & ~3))==OK)
        beeping = FALSE;
}

/*=====
*
*                      scr_init
*=====*/

```

```

*=====*/
PUBLIC void scr_init(tp)
tty_t *tp;
{
/* Initialize the screen driver. */
console_t *cons;
phys_bytes vid_base;
u16_t bios_columns, bios_crtbase, bios_fontlines;
u8_t bios_rows;
int line;
int s;
static int vdu_initialized = 0;
unsigned page_size;

/* Associate console and TTY. */
line = tp - &tty_table[0];
if (line >= nr_cons) return;
cons = &cons_table[line];
cons->c_tty = tp;
tp->tty_priv = cons;

/* Fill in TTY function hooks. */
tp->tty_devwrite = cons_write;
tp->tty_echo = cons_echo;
tp->tty_ioctl = cons_ioctl;

/* Get the BIOS parameters that describe the VDU. */
if (! vdu_initialized++) {

    /* How about error checking? What to do on failure??? */
    s=sys_vircopy(SELF, BIOS_SEG, (vir_bytes) VDU_SCREEN_COLS_ADDR,
        SELF, D, (vir_bytes) &bios_columns, VDU_SCREEN_COLS_SIZE);
    s=sys_vircopy(SELF, BIOS_SEG, (vir_bytes) VDU_CRT_BASE_ADDR,
        SELF, D, (vir_bytes) &bios_crtbase, VDU_CRT_BASE_SIZE);
    s=sys_vircopy(SELF, BIOS_SEG, (vir_bytes) VDU_SCREEN_ROWS_ADDR,
        SELF, D, (vir_bytes) &bios_rows, VDU_SCREEN_ROWS_SIZE);
    s=sys_vircopy(SELF, BIOS_SEG, (vir_bytes) VDU_FONTLINES_ADDR,
        SELF, D, (vir_bytes) &bios_fontlines, VDU_FONTLINES_SIZE);

    vid_port = bios_crtbase;
    scr_width = bios_columns;
    font_lines = bios_fontlines;
    scr_lines = machine.vdu_ega ? bios_rows+1 : 25;

    if (color) {
        vid_base = COLOR_BASE;
        vid_size = COLOR_SIZE;
    } else {
        vid_base = MONO_BASE;
        vid_size = MONO_SIZE;
    }
    if (machine.vdu_ega) vid_size = EGA_SIZE;
    wrap = ! machine.vdu_ega;

    s = sys_segctl(&vid_index, &vid_seg, &vid_off, vid_base, vid_size);

    vid_size >>= 1;          /* word count */
    vid_mask = vid_size - 1;

    /* Size of the screen (number of displayed characters.) */
    scr_size = scr_lines * scr_width;

    /* There can be as many consoles as video memory allows. */
    nr_cons = vid_size / scr_size;
    if (nr_cons > NR_CONS) nr_cons = NR_CONS;
    if (nr_cons > 1) wrap = 0;
    page_size = vid_size / nr_cons;
}

cons->c_start = line * page_size;
cons->c_limit = cons->c_start + page_size;
cons->c_cur = cons->c_org = cons->c_start;
cons->c_attr = cons->c_blank = BLANK_COLOR;

```



```

if (line != 0) {
    /* Clear the non-console vtys. */
    blank_color = BLANK_COLOR;
    mem_vid_copy(BLANK_MEM, cons->c_start, scr_size);
} else {
    int i, n;
    /* Set the cursor of the console vty at the bottom. c_cur
     * is updated automatically later.
     */
    scroll_screen(cons, SCROLL_UP);
    cons->c_row = scr_lines - 1;
    cons->c_column = 0;
}
select_console(0);
cons_ioctl(tp, 0);
}

/*=====
 *                               kputc                               *
 *=====*/
PUBLIC void kputc(c)
int c;
{
    /* Accumulate a single character for a kernel message. Send a notification
     * the to output driver if an END_OF_KMESS is encountered.
     */
    #if 0
        ser_putc(c);
        return;
    #endif

    #if 0
        if (panicing)
    #endif
        cons_putk(c);
    if (c != 0) {
        kmess.km_buf[kmess.km_next] = c; /* put normal char in buffer */
        if (kmess.km_size < KMESS_BUF_SIZE)
            kmess.km_size += 1;
        kmess.km_next = (kmess.km_next + 1) % KMESS_BUF_SIZE;
    } else {
        notify(LOG_PROC_NR);
    }
}

/*=====
 *                               do_new_kmess                       *
 *=====*/
PUBLIC void do_new_kmess(m)
message *m;
{
    /* Notification for a new kernel message. */
    struct kmessages kmess; /* kmessages structure */
    static int prev_next = 0; /* previous next seen */
    int size, next;
    int bytes;
    int r;

    /* Try to get a fresh copy of the buffer with kernel messages. */
    #if DEAD_CODE
        /* During shutdown, the reply is garbled because new notifications arrive
         * while the system task makes a copy of the kernel messages buffer.
         * Hence, don't check the return value.
         */
        if ((r=sys_getkmessages(&kmess)) != OK) {
            printf("TTY: couldn't get copy of kmessages: %d, 0x%x\n", r, r);
            return;
        }
    #endif
    sys_getkmessages(&kmess);

    /* Print only the new part. Determine how many new bytes there are with
     * help of the current and previous 'next' index. Note that the kernel
     * buffer is circular. This works fine if less then KMESS_BUF_SIZE bytes

```

```

    * is new data; else we miss % KMESS_BUF_SIZE here.
    * Check for size being positive, the buffer might as well be emptied!
    */
    if (kmess.km_size > 0) {
        bytes = ((kmess.km_next + KMESS_BUF_SIZE) - prev_next) % KMESS_BUF_SIZE;
        r = prev_next; /* start at previous old */
        while (bytes > 0) {
            cons_putk( kmess.km_buf[(r%KMESS_BUF_SIZE)] );
            bytes--;
            r++;
        }
        cons_putk(0); /* terminate to flush output */
    }

    /* Almost done, store 'next' so that we can determine what part of the
     * kernel messages buffer to print next time a notification arrives.
     */
    prev_next = kmess.km_next;
}

/*=====
 * do_diagnostics
 *=====*/
PUBLIC void do_diagnostics(m_ptr)
message *m_ptr; /* pointer to request message */
{
    /* Print a string for a server. */
    char c;
    vir_bytes src;
    int count;
    int result = OK;
    int proc_nr = m_ptr->DIAG_ENDPT;
    if (proc_nr == SELF) proc_nr = m_ptr->m_source;

    src = (vir_bytes) m_ptr->DIAG_PRINT_BUF;
    for (count = m_ptr->DIAG_BUF_COUNT; count > 0; count--) {
        if (sys_vircopy(proc_nr, D, src++, SELF, D, (vir_bytes) &c, 1) != OK) {
            result = EFAULT;
            break;
        }
        cons_putk(c);
    }
    cons_putk(0); /* always terminate, even with EFAULT */
    m_ptr->m_type = result;
    send(m_ptr->m_source, m_ptr);
}

/*=====
 * do_get_kmess
 *=====*/
PUBLIC void do_get_kmess(m_ptr)
message *m_ptr; /* pointer to request message */
{
    /* Provide the log device with debug output */
    vir_bytes dst;
    int r;

    dst = (vir_bytes) m_ptr->GETKM_PTR;
    r = OK;
    if (sys_vircopy(SELF, D, (vir_bytes)&kmess, m_ptr->m_source, D,
        dst, sizeof(kmess)) != OK) {
        r = EFAULT;
    }
    m_ptr->m_type = r;
    send(m_ptr->m_source, m_ptr);
}

/*=====
 * cons_putk
 *=====*/
PRIVATE void cons_putk(c)
int c; /* character to print */
{
    /* This procedure is used to print a character on the console.

```

```

*/
    if (c != 0) {
        if (c == '\n') cons_putk('\r');
        out_char(&cons_table[0], (int) c);
    } else {
        flush(&cons_table[0]);
    }
}

/*=====
*                                     toggle_scroll                                     *
*=====*/
PUBLIC void toggle_scroll()
{
    /* Toggle between hardware and software scroll. */

    cons_org0();
    softscroll = !softscroll;
    printf("%sware scrolling enabled.\n", softscroll ? "Soft" : "Hard");
}

/*=====
*                                     cons_stop                                     *
*=====*/
PUBLIC void cons_stop()
{
    /* Prepare for halt or reboot. */
    cons_org0();
    softscroll = 1;
    select_console(0);
    cons_table[0].c_attr = cons_table[0].c_blank = BLANK_COLOR;
}

/*=====
*                                     cons_org0                                     *
*=====*/
PRIVATE void cons_org0()
{
    /* Scroll video memory back to put the origin at 0. */
    int cons_line;
    console_t *cons;
    unsigned n;

    for (cons_line = 0; cons_line < nr_cons; cons_line++) {
        cons = &cons_table[cons_line];
        while (cons->c_org > cons->c_start) {
            n = vid_size - scr_size; /* amount of unused memory */
            if (n > cons->c_org - cons->c_start)
                n = cons->c_org - cons->c_start;
            vid_vid_copy(cons->c_org, cons->c_org - n, scr_size);
            cons->c_org -= n;
        }
        flush(cons);
    }
    select_console(ccurrent);
}

/*=====
*                                     select_console                                     *
*=====*/
PUBLIC void select_console(int cons_line)
{
    /* Set the current console to console number 'cons_line'. */

    if (cons_line < 0 || cons_line >= nr_cons) return;
    ccurrent = cons_line;
    curcons = &cons_table[cons_line];
    set_6845(VID_ORG, curcons->c_org);
    set_6845(CURSOR, curcons->c_cur);
}

/*=====
*                                     con_loadfont                                     *
*=====*/

```

```

PUBLIC int con_loadfont(m)
message *m;
{
    /* Load a font into the EGA or VGA adapter. */
    int result;
    static struct sequence seq1[7] = {
        { GA_SEQUENCER_INDEX, 0x00, 0x01 },
        { GA_SEQUENCER_INDEX, 0x02, 0x04 },
        { GA_SEQUENCER_INDEX, 0x04, 0x07 },
        { GA_SEQUENCER_INDEX, 0x00, 0x03 },
        { GA_GRAPHICS_INDEX, 0x04, 0x02 },
        { GA_GRAPHICS_INDEX, 0x05, 0x00 },
        { GA_GRAPHICS_INDEX, 0x06, 0x00 },
    };
    static struct sequence seq2[7] = {
        { GA_SEQUENCER_INDEX, 0x00, 0x01 },
        { GA_SEQUENCER_INDEX, 0x02, 0x03 },
        { GA_SEQUENCER_INDEX, 0x04, 0x03 },
        { GA_SEQUENCER_INDEX, 0x00, 0x03 },
        { GA_GRAPHICS_INDEX, 0x04, 0x00 },
        { GA_GRAPHICS_INDEX, 0x05, 0x10 },
        { GA_GRAPHICS_INDEX, 0x06, 0 },
    };

    seq2[6].value= color ? 0x0E : 0x0A;

    if (!machine.vdu_ega) return(ENOTTY);
    result = ga_program(seq1); /* bring font memory into view */

    result = sys_physcopy(m->IO_ENDPT, D, (vir_bytes) m->ADDRESS,
        NONE, PHYS_SEG, (phys_bytes) GA_VIDEO_ADDRESS, (phys_bytes)GA_FONT_SIZE);

    result = ga_program(seq2); /* restore */

    return(result);
}

/*=====
 *                               ga_program                               *
 *=====*/
PRIVATE int ga_program(seq)
struct sequence *seq;
{
    pvb_pair_t char_out[14];
    int i;
    for (i=0; i<7; i++) {
        pv_set(char_out[2*i], seq->index, seq->port);
        pv_set(char_out[2*i+1], seq->index+1, seq->value);
        seq++;
    }
    return sys_voutb(char_out, 14);
}

/*=====
 *                               cons_ioctl                               *
 *=====*/
PRIVATE int cons_ioctl(tp, try)
tty_t *tp;
int try;
{
    /* Set the screen dimensions. */

    tp->tty_winsize.ws_row= scr_lines;
    tp->tty_winsize.ws_col= scr_width;
    tp->tty_winsize.ws_xpixel= scr_width * 8;
    tp->tty_winsize.ws_ypixel= scr_lines * font_lines;
}

#define COM1_BASE          0x3F8
#define COM1_THR            (COM1_BASE + 0)
#define LSR_THRE           0x20
#define COM1_LSR            (COM1_BASE + 5)

PRIVATE void ser_putc(char c)

```

```
{
    unsigned long b;
    int i;
    int lsr, thr;

    lsr= COM1_LSR;
    thr= COM1_THR;
    for (i= 0; i<100; i++)
    {
        sys_inb(lsr, &b);
        if (b & LSR_THRE)
            break;
    }
    sys_outb(thr, c);
}
```

```

/* Keyboard driver for PC's and AT's.
 *
 * Changes:
 *   Jul 13, 2004    processes can observe function keys (Jorrit N. Herder)
 *   Jun 15, 2004    removed wreboot(), except panic dumps (Jorrit N. Herder)
 *   Feb 04, 1994    loadable keymaps (Marcus Hampel)
 */

#include "../drivers.h"
#include <sys/ioctl.h>
#include <sys/kbdio.h>
#include <sys/time.h>
#include <sys/select.h>
#include <termios.h>
#include <signal.h>
#include <unistd.h>
#include <minix/callnr.h>
#include <minix/com.h>
#include <minix/keymap.h>
#include "tty.h"
#include "keymaps/us-std.src"
#include "../kernel/const.h"
#include "../kernel/config.h"
#include "../kernel/type.h"
#include "../kernel/proc.h"

int irq_hook_id = -1;
int aux_irq_hook_id = -1;

/* Standard and AT keyboard. (PS/2 MCA implies AT throughout.) */
#define KEYBD 0x60 /* I/O port for keyboard data */

/* AT keyboard. */
#define KB_COMMAND 0x64 /* I/O port for commands on AT */
#define KB_STATUS 0x64 /* I/O port for status on AT */
#define KB_ACK 0xFA /* keyboard ack response */
#define KB_AUX_BYTE 0x20 /* Auxiliary Device Output Buffer Full */
#define KB_OUT_FULL 0x01 /* status bit set when keypress char pending */
#define KB_IN_FULL 0x02 /* status bit set when not ready to receive */
#define KBC_RD_RAM_CCB 0x20 /* Read controller command byte */
#define KBC_WR_RAM_CCB 0x60 /* Write controller command byte */
#define KBC_DI_AUX 0xA7 /* Disable Auxiliary Device */
#define KBC_EN_AUX 0xA8 /* Enable Auxiliary Device */
#define KBC_DI_KBD 0xAD /* Disable Keyboard Interface */
#define KBC_EN_KBD 0xAE /* Enable Keyboard Interface */
#define KBC_WRITE_AUX 0xD4 /* Write to Auxiliary Device */
#define LED_CODE 0xED /* command to keyboard to set LEDs */
#define MAX_KB_ACK_RETRIES 0x1000 /* max #times to wait for kb ack */
#define MAX_KB_BUSY_RETRIES 0x1000 /* max #times to loop while kb busy */
#define KBIT 0x80 /* bit used to ack characters to keyboard */

#define KBC_IN_DELAY 7 /* wait 7 microseconds when polling */

/* Miscellaneous. */
#define ESC_SCAN 0x01 /* reboot key when panicking */
#define SLASH_SCAN 0x35 /* to recognize numeric slash */
#define RSHIFT_SCAN 0x36 /* to distinguish left and right shift */
#define HOME_SCAN 0x47 /* first key on the numeric keypad */
#define INS_SCAN 0x52 /* INS for use in CTRL-ALT-INS reboot */
#define DEL_SCAN 0x53 /* DEL for use in CTRL-ALT-DEL reboot */

#define KBD_BUFSZ 1024 /* Buffer size for raw scan codes */
#define KBD_OUT_BUFSZ 16 /* Output buffer to sending data to the
 * keyboard.
 */

#define MICROS_TO_TICKS(m) (((m)*HZ/1000000)+1)

#define CONSOLE 0 /* line number for console */
#define KB_IN_BYTES 32 /* size of keyboard input buffer */
PRIVATE char ibuf[KB_IN_BYTES]; /* input buffer */
PRIVATE char *ihead = ibuf; /* next free spot in input buffer */
PRIVATE char *itail = ibuf; /* scan code to return to TTY */
PRIVATE int icount; /* # codes in buffer */

```

```

PRIVATE int esc; /* escape scan code detected? */
PRIVATE int alt_l; /* left alt key state */
PRIVATE int alt_r; /* right alt key state */
PRIVATE int alt; /* either alt key */
PRIVATE int ctrl_l; /* left control key state */
PRIVATE int ctrl_r; /* right control key state */
PRIVATE int ctrl; /* either control key */
PRIVATE int shift_l; /* left shift key state */
PRIVATE int shift_r; /* right shift key state */
PRIVATE int shift; /* either shift key */
PRIVATE int num_down; /* num lock key depressed */
PRIVATE int caps_down; /* caps lock key depressed */
PRIVATE int scroll_down; /* scroll lock key depressed */
PRIVATE int locks[NR_CONS]; /* per console lock keys state */

/* Lock key active bits. Chosen to be equal to the keyboard LED bits. */
#define SCROLL_LOCK 0x01
#define NUM_LOCK 0x02
#define CAPS_LOCK 0x04

PRIVATE char numpad_map[] =
    {'H', 'Y', 'A', 'B', 'D', 'C', 'V', 'U', 'G', 'S', 'T', '@'};

/* Variables and definition for observed function keys. */
typedef struct observer { int proc_nr; int events; } obs_t;
PRIVATE obs_t fkey_obs[12]; /* observers for F1-F12 */
PRIVATE obs_t sfkey_obs[12]; /* observers for SHIFT F1-F12 */

PRIVATE struct kbd
{
    int minor;
    int nr_open;
    char buf[KBD_BUFSZ];
    int offset;
    int avail;
    int req_size;
    int req_proc;
    vir_bytes req_addr;
    int incaller;
    int select_ops;
    int select_proc;
} kbd, kbdaux;

/* Data that is to be sent to the keyboard. Each byte is ACKed by the
 * keyboard.
 */
PRIVATE struct kbd_outack
{
    unsigned char buf[KBD_OUT_BUFSZ];
    int offset;
    int avail;
    int expect_ack;
} kbdaux;

PRIVATE int kbd_watchdog_set= 0;
PRIVATE int kbd_alive= 1;
PRIVATE timer_t tmr_kbd_wd;

FORWARD _PROTOTYPE( void handle_req, (struct kbd *kbp, message *m) );
FORWARD _PROTOTYPE( int handle_status, (struct kbd *kbp, message *m) );
FORWARD _PROTOTYPE( void kbc_cmd0, (int cmd) );
FORWARD _PROTOTYPE( void kbc_cmd1, (int cmd, int data) );
FORWARD _PROTOTYPE( int kbc_read, (void) );
FORWARD _PROTOTYPE( void kbd_send, (void) );
FORWARD _PROTOTYPE( int kb_ack, (void) );
FORWARD _PROTOTYPE( int kb_wait, (void) );
FORWARD _PROTOTYPE( int func_key, (int scode) );
FORWARD _PROTOTYPE( int scan_keyboard, (unsigned char *bp, int *isauxp) );
FORWARD _PROTOTYPE( unsigned make_break, (int scode) );
FORWARD _PROTOTYPE( void set_leds, (void) );
FORWARD _PROTOTYPE( void show_key_mappings, (void) );
FORWARD _PROTOTYPE( int kb_read, (struct tty *tp, int try) );
FORWARD _PROTOTYPE( unsigned map_key, (int scode) );

```

```

FORWARD _PROTOTYPE( void micro_delay, (unsigned long usecs) ) ;
FORWARD _PROTOTYPE( void kbd_watchdog, (timer_t *tmrp) ) ;

/*=====
 *
 * do_kbd
 *=====*/
PUBLIC void do_kbd(message *m)
{
    handle_req(&kbd, m);
}

/*=====
 *
 * kbd_status
 *=====*/
PUBLIC int kbd_status(message *m)
{
    int r;

    r= handle_status(&kbd, m);
    if (r)
        return r;
    return handle_status(&kbdaux, m);
}

/*=====
 *
 * do_kbdaux
 *=====*/
PUBLIC void do_kbdaux(message *m)
{
    handle_req(&kbdaux, m);
}

/*=====
 *
 * handle_req
 *=====*/
PRIVATE void handle_req(kbdp, m)
struct kbd *kbdp;
message *m;
{
    int i, n, r, ops, watch;
    unsigned char c;

    /* Execute the requested device driver function. */
    r= EINVAL; /* just in case */
    switch (m->m_type) {
        case DEV_OPEN:
            kbdp->nr_open++;
            r= OK;
            break;
        case DEV_CLOSE:
            kbdp->nr_open--;
            if (kbdp->nr_open < 0)
            {
                printf("TTY(kbd): open count is negative\n" );
                kbdp->nr_open= 0;
            }
            if (kbdp->nr_open == 0)
                kbdp->avail= 0;
            r= OK;
            break;
        case DEV_READ:
            if (kbdp->req_size)
            {
                /* We handle only request at a time */
                r= EIO;
                break;
            }
            if (kbdp->avail == 0)
            {
                /* Should record proc */
                kbdp->req_size= m->COUNT;
            }
        }
    }
}

```



```

        kbdp->req_proc= m->IO_ENDPT;
        kbdp->req_addr= (vir_bytes)m->ADDRESS;
        kbdp->incaller= m->m_source;
        r= SUSPEND;
        break;
    }

    /* Handle read request */
    n= kbdp->avail;
    if (n > m->COUNT)
        n= m->COUNT;
    if (kbdp->offset + n > KBD_BUFSZ)
        n= KBD_BUFSZ-kbdp->offset;
    if (n <= 0)
        panic("TTY", "do_kbd(READ): bad n", n);
    r= sys_vircopy(SEL, D, (vir_bytes)&kbdp->buf[kbdp->offset],
        m->IO_ENDPT, D, (vir_bytes) m->ADDRESS, n);
    if (r == OK)
    {
        kbdp->offset= (kbdp->offset+n) % KBD_BUFSZ;
        kbdp->avail -= n;
        r= n;
    }

    break;

case DEV_WRITE:
    if (kbdp != &kbdaux)
    {
        printf("write to keyboard not implemented\n");
        r= EINVAL;
        break;
    }

    /* Assume that output to AUX only happens during
     * initialization and we can afford to lose input. This should
     * be fixed at a later time.
     */
    for (i= 0; i<m->COUNT; i++)
    {
        r= sys_vircopy(m->IO_ENDPT, D, (vir_bytes) m->ADDRESS+i,
            SEL, D, (vir_bytes)&c, 1);
        if (r != OK)
            break;
        kbc_cmd1(KBC_WRITE_AUX, c);
    }
    r= i;
    break;

case CANCEL:
    kbdp->req_size= 0;
    r= OK;
    break;

case DEV_SELECT:
    ops = m->IO_ENDPT & (SEL_RD|SEL_WR|SEL_ERR);
    watch = (m->IO_ENDPT & SEL_NOTIFY) ? 1 : 0;

    r= 0;
    if (kbdp->avail && (ops & SEL_RD))
    {
        r |= SEL_RD;
        break;
    }

    if (ops && watch)
    {
        kbdp->select_ops |= ops;
        kbdp->select_proc= m->m_source;
    }
    break;

case DEV_IOCTL:
    if (kbdp == &kbd && m->TTY_REQUEST == KIOCSLEDS)
    {
        kio_leds_t leds;

```

```

        unsigned char b;

        r= sys_vircopy(m->IO_ENDPT, D, (vir_bytes) m->ADDRESS,
            SELF, D, (vir_bytes)&leds, sizeof(leds));
        if (r != OK)
            break;
        b= 0;
        if (leds.kl_bits & KBD_LEDS_NUM) b |= NUM_LOCK;
        if (leds.kl_bits & KBD_LEDS_CAPS) b |= CAPS_LOCK;
        if (leds.kl_bits & KBD_LEDS_SCROLL) b |= SCROLL_LOCK;
        if (kbdout.avail == 0)
            kbdout.offset= 0;
        if (kbdout.offset + kbdout.avail + 2 > KBD_OUT_BUFSZ)
        {
            /* Output buffer is full. Ignore this command.
             * Reset ACK flag.
             */
            kbdout.expect_ack= 0;
        }
        else
        {
            kbdout.buf[kbdout.offset+kbdout.avail]=
                LED_CODE;
            kbdout.buf[kbdout.offset+kbdout.avail+1]= b;
            kbdout.avail += 2;
        }
        if (!kbdout.expect_ack)
            kbd_send();
        r= OK;
        break;
    }
    if (kbdp == &kbd && m->TTY_REQUEST == KIOCBELL)
    {
        kio_bell_t bell;
        clock_t ticks;

        r= sys_vircopy(m->IO_ENDPT, D, (vir_bytes) m->ADDRESS,
            SELF, D, (vir_bytes)&bell, sizeof(bell));
        if (r != OK)
            break;

        ticks= bell.kb_duration.tv_usec * HZ / 1000000;
        ticks += bell.kb_duration.tv_sec * HZ;
        if (!ticks)
            ticks++;
        beep_x(bell.kb_pitch, ticks);

        r= OK;
        break;
    }
    r= ENOTTY;
    break;

default:
    printf("Warning, TTY(kbd) got unexpected request %d from %d\n",
        m->m_type, m->m_source);
    r= EINVAL;
}
tty_reply(TASK_REPLY, m->m_source, m->IO_ENDPT, r);
}

```

```

/*=====
 *                               handle_status                               *
 *=====*/
PRIVATE int handle_status(kbdp, m)
struct kbd *kbdp;
message *m;
{
    int n, r;

    if (kbdp->avail && kbdp->req_size && m->m_source == kbdp->incaller)
    {
        /* Handle read request */
    }
}

```

```

        n= kbdp->avail;
        if (n > kbdp->req_size)
            n= kbdp->req_size;
        if (kbdp->offset + n > KBD_BUFSZ)
            n= KBD_BUFSZ-kbdp->offset;
        if (n <= 0)
            panic("TTY", "kbd_status: bad n", n);
        kbdp->req_size= 0;
        r= sys_vircopy(SEL_F, D, (vir_bytes)&kbdp->buf[kbdp->offset],
            kbdp->req_proc, D, kbdp->req_addr, n);
        if (r == OK)
        {
            kbdp->offset= (kbdp->offset+n) % KBD_BUFSZ;
            kbdp->avail -= n;
            r= n;
        }

        m->m_type = DEV_REVIVE;
        m->REP_ENDPT= kbdp->req_proc;
        m->REP_STATUS= r;
        return 1;
    }
    if (kbdp->avail && (kbdp->select_ops & SEL_RD) &&
        m->m_source == kbdp->select_proc)
    {
        m->m_type = DEV_IO_READY;
        m->DEV_MINOR = kbdp->minor;
        m->DEV_SEL_OPS = SEL_RD;

        kbdp->select_ops &= ~SEL_RD;
        return 1;
    }

    return 0;
}

/*=====
 *                               map_key0                               *
 *=====*/
/* Map a scan code to an ASCII code ignoring modifiers. */
#define map_key0(scode) \
    ((unsigned) keymap[(scode) * MAP_COLS])

/*=====
 *                               map_key                               *
 *=====*/
PRIVATE unsigned map_key(scode)
int scode;
{
    /* Map a scan code to an ASCII code. */

    int caps, column, lk;
    ul6_t *keyrow;

    if (scode == SLASH_SCAN && esc) return '/'; /* don't map numeric slash */

    keyrow = &keymap[scode * MAP_COLS];

    caps = shift;
    lk = locks[ccurrent];
    if ((lk & NUM_LOCK) && HOME_SCAN <= scode && scode <= DEL_SCAN) caps = !caps;
    if ((lk & CAPS_LOCK) && (keyrow[0] & HASCAPS)) caps = !caps;

    if (alt) {
        column = 2;
        if (ctrl || alt_r) column = 3; /* Ctrl + Alt == AltGr */
        if (caps) column = 4;
    } else {
        column = 0;
        if (caps) column = 1;
        if (ctrl) column = 5;
    }

    return keyrow[column] & ~HASCAPS;
}

```

```

}

/*=====
 *                               kbd_interrupt                               *
 *=====*/
PUBLIC void kbd_interrupt(m_ptr)
message *m_ptr;
{
    /* A keyboard interrupt has occurred. Process it. */
    int o, isaux;
    unsigned char scode;
    struct kbd *kbp;
    static timer_t timer;          /* timer must be static! */

    /* Fetch the character from the keyboard hardware and acknowledge it. */
    if (!scan_keyboard(&scode, &isaux))
        return;

    if (isaux)
        kbp = &kbdaux;
    else if (kbd.nr_open && !panicing)
        kbp = &kbd;
    else
        kbp = NULL;

    if (kbp)
    {
        /* raw scan codes or aux data */
        if (kbp->avail >= KBD_BUFSZ)
        {
            printf("kbd_interrupt: %s buffer is full\n",
                   isaux ? "kbdaux" : "keyboard");
            return; /* Buffer is full */
        }
        o = (kbp->offset + kbp->avail) % KBD_BUFSZ;
        kbp->buf[o] = scode;
        kbp->avail++;
        if (kbp->req_size)
            notify(kbp->incaller);
        if (kbp->select_ops & SEL_RD)
            notify(kbp->select_proc);
        return;
    }

    /* Store the scancode in memory so the task can get at it later. */
    if (icount < KB_IN_BYTES) {
        *ihead++ = scode;
        if (ihead == ibuf + KB_IN_BYTES) ihead = ibuf;
        icount++;
        tty_table[ccurrent].tty_events = 1;
        if (tty_table[ccurrent].tty_select_ops & SEL_RD) {
            select_retry(&tty_table[ccurrent]);
        }
    }
}

/*=====
 *                               kb_read                               *
 *=====*/
PRIVATE int kb_read(tp, try)
tty_t *tp;
int try;
{
    /* Process characters from the circular keyboard buffer. */
    char buf[3];
    int scode;
    unsigned ch;

    tp = &tty_table[ccurrent];          /* always use the current console */

    if (try) {
        if (icount > 0) return 1;
        return 0;
    }
}

```

```

while (icount > 0) {
    scode = *itail++;
    if (itail == ibuf + KB_IN_BYTES) itail = ibuf;
    icount--;

    /* Function keys are being used for debug dumps. */
    if (func_key(scode)) continue;

    /* Perform make/break processing. */
    ch = make_break(scode);

    if (ch <= 0xFF) {
        /* A normal character. */
        buf[0] = ch;
        (void) in_process(tp, buf, 1);
    } else
    if (HOME <= ch && ch <= INSRT) {
        /* An ASCII escape sequence generated by the numeric pad. */
        buf[0] = ESC;
        buf[1] = '[';
        buf[2] = numpad_map[ch - HOME];
        (void) in_process(tp, buf, 3);
    } else
    if (ch == ALEFT) {
        /* Choose lower numbered console as current console. */
        select_console(ccurrent - 1);
        set_leds();
    } else
    if (ch == ARIGHT) {
        /* Choose higher numbered console as current console. */
        select_console(ccurrent + 1);
        set_leds();
    } else
    if (AF1 <= ch && ch <= AF12) {
        /* Alt-F1 is console, Alt-F2 is tty01, etc. */
        select_console(ch - AF1);
        set_leds();
    } else
    if (CF1 <= ch && ch <= CF12) {
        switch(ch) {
            case CF1: show_key_mappings(); break;
            case CF3: toggle_scroll(); break; /* hardware <-> software */
            case CF7: sigchar(&tty_table[CONSOLE], SIGQUIT); break;
            case CF8: sigchar(&tty_table[CONSOLE], SIGINT); break;
            case CF9: sigchar(&tty_table[CONSOLE], SIGKILL); break;
        }
    }
}

return 1;
}

/*=====
 *                               kbd_send                               *
 *=====*/
PRIVATE void kbd_send()
{
    unsigned long sb;
    int r;
    clock_t now;

    if (!kbdout.avail)
        return;
    if (kbdout.expect_ack)
        return;

    sys_inb(KB_STATUS, &sb);
    if (sb & (KB_OUT_FULL|KB_IN_FULL))
    {
        printf("not sending 1: sb = 0x%x\n", sb);
        return;
    }
    micro_delay(KBC_IN_DELAY);
}

```

```

    sys_inb(KB_STATUS, &sb);
    if (sb & (KB_OUT_FULL|KB_IN_FULL))
    {
        printf("not sending 2: sb = 0x%x\n", sb);
        return;
    }

    /* Okay, buffer is really empty */
#if 0
    printf("sending byte 0x%x to keyboard\n", kbdout.buf[kbdout.offset]);
#endif
    sys_outb(KEYBD, kbdout.buf[kbdout.offset]);
    kbdout.offset++;
    kbdout.avail--;
    kbdout.expect_ack = 1;

    kbd_alive = 1;
    if (kbd_watchdog_set)
    {
        /* Add a timer to the timers list. Possibly reschedule the
         * alarm.
         */
        if ((r = getuptime(&now)) != OK)
            panic("TTY", "Keyboard couldn't get clock's uptime.", r);
        tmrs_settimer(&tty_timers, &tmr_kbd_wd, now+HZ, kbd_watchdog,
                     NULL);
        if (tty_timers->tmr_exp_time != tty_next_timeout) {
            tty_next_timeout = tty_timers->tmr_exp_time;
            if ((r = sys_setalarm(tty_next_timeout, 1)) != OK)
                panic("TTY", "Keyboard couldn't set alarm.", r);
        }
        kbd_watchdog_set = 1;
    }
}

/*=====
 *
 *                               make_break
 *=====*/
PRIVATE unsigned make_break(scode)
int scode;                /* scan code of key just struck or released */
{
    /* This routine can handle keyboards that interrupt only on key depression,
     * as well as keyboards that interrupt on key depression and key release.
     * For efficiency, the interrupt routine filters out most key releases.
     */
    int ch, make, escape;
    static int CAD_count = 0;

    /* Check for CTRL-ALT-DEL, and if found, halt the computer. This would
     * be better done in keyboard() in case TTY is hung, except control and
     * alt are set in the high level code.
     */
    if (ctrl && alt && (scode == DEL_SCAN || scode == INS_SCAN))
    {
        if (++CAD_count == 3) {
            cons_stop();
            sys_abort(RBT_HALT);
        }
        sys_kill(INIT_PROC_NR, SIGABRT);
        return -1;
    }

    /* High-order bit set on key release. */
    make = (scode & KEY_RELEASE) == 0;                /* true if pressed */

    ch = map_key(scode &= ASCII_MASK);                /* map to ASCII */

    escape = esc;                /* Key is escaped? (true if added since the XT) */
    esc = 0;

    switch (ch) {
        case CTRL:                /* Left or right control key */
            *(escape ? &ctrl_r : &ctrl_l) = make;
            ctrl = ctrl_l | ctrl_r;

```

```

        break;
    case SHIFT:          /* Left or right shift key */
        *(scode == RSHIFT_SCAN ? &shift_r : &shift_l) = make;
        shift = shift_l | shift_r;
        break;
    case ALT:            /* Left or right alt key */
        *(escape ? &alt_r : &alt_l) = make;
        alt = alt_l | alt_r;
        break;
    case CALOCK:         /* Caps lock - toggle on 0 -> 1 transition */
        if (caps_down < make) {
            locks[current] ^= CAPS_LOCK;
            set_leds();
        }
        caps_down = make;
        break;
    case NLOCK:          /* Num lock */
        if (num_down < make) {
            locks[current] ^= NUM_LOCK;
            set_leds();
        }
        num_down = make;
        break;
    case SLOCK:          /* Scroll lock */
        if (scroll_down < make) {
            locks[current] ^= SCROLL_LOCK;
            set_leds();
        }
        scroll_down = make;
        break;
    case EXTKEY:         /* Escape keycode */
        esc = 1;          /* Next key is escaped */
        return(-1);
    default:             /* A normal key */
        if (make) return(ch);
}

/* Key release, or a shift type key. */
return(-1);
}

/*=====
 *                               set_leds                               *
 *=====*/
PRIVATE void set_leds()
{
    /* Set the LEDs on the caps, num, and scroll lock keys */
    int s;
    if (!machine.pc_at) return; /* PC/XT doesn't have LEDs */

    kb_wait();          /* wait for buffer empty */
    if ((s=sys_outb(KEYBD, LED_CODE)) != OK)
        printf("Warning, sys_outb couldn't prepare for LED values: %d\n", s);
    /* prepare keyboard to accept LED values */
    kb_ack();           /* wait for ack response */

    kb_wait();          /* wait for buffer empty */
    if ((s=sys_outb(KEYBD, locks[current])) != OK)
        printf("Warning, sys_outb couldn't give LED values: %d\n", s);
    /* give keyboard LED values */
    kb_ack();           /* wait for ack response */
}

/*=====
 *                               kbc_cmd0                               *
 *=====*/
PRIVATE void kbc_cmd0(cmd)
int cmd;
{
    kb_wait();
    sys_outb(KB_COMMAND, cmd);
}

/*=====

```

```

*                                     kbc_cmd1                                     *
*=====*/
PRIVATE void kbc_cmd1(cmd, data)
int cmd;
int data;
{
    kb_wait();
    sys_outb(KB_COMMAND, cmd);
    kb_wait();
    sys_outb(KEYBD, data);
}

/*=====*
*                                     kbc_read                                     *
*=====*/
PRIVATE int kbc_read()
{
    int i;
    unsigned long byte, st;
#if 0
    struct micro_state ms;
#endif
#if DEBUG
    printf("in kbc_read\n");
#endif

    /* Wait at most 1 second for a byte from the keyboard or
     * the kbd controller, return -1 on a timeout.
     */
    for (i= 0; i<1000; i++)
    #if 0
        micro_start(&ms);
    do
    #endif
    {
        sys_inb(KB_STATUS, &st);
        if (st & KB_OUT_FULL)
        {
            micro_delay(KBC_IN_DELAY);
            sys_inb(KEYBD, &byte);
            if (st & KB_AUX_BYTE)
            {
#if DEBUG
                printf(
                    "keyboard'kbc_read: ignoring byte (0x%x) from aux device.\n",
                    byte);
#endif
                continue;
            }
        }
        #if 0
            while (micro_elapsed(&ms) < 1000000);
        #endif
        panic("TTY", "kbc_read failed to complete", NO_NUM);
    }

    #if 0
        while (micro_elapsed(&ms) < 1000000);
    #endif
    panic("TTY", "kbc_read failed to complete", NO_NUM);
}

/*=====*
*                                     kb_wait                                     *
*=====*/
PRIVATE int kb_wait()
{
    /* Wait until the controller is ready; return zero if this times out. */

    int retries;

```



```

unsigned long status, temp;
int s, isaux;
unsigned char byte;

retries = MAX_KB_BUSY_RETRIES + 1;    /* wait until not busy */
do {
    s = sys_inb(KB_STATUS, &status);
    if (status & KB_OUT_FULL) {
        if (scan_keyboard(&byte, &isaux))
            {
                printf("ignoring %sbyte in kb_wait\n", isaux ? "AUX " : "");
            }
        if (! (status & (KB_IN_FULL|KB_OUT_FULL)) )
            break;
    } while (--retries != 0);
    return(retries);
}

/*=====
*                               kb_ack                               *
*=====*/
PRIVATE int kb_ack()
{
    /* Wait until kbd acknowledges last command; return zero if this times out. */

    int retries, s;
    unsigned long u8val;

    retries = MAX_KB_ACK_RETRIES + 1;
    do {
        s = sys_inb(KEYBD, &u8val);
        if (u8val == KB_ACK)
            break;
    } while(--retries != 0);
    return(retries);
}

/*=====
*                               kb_init                               *
*=====*/
PUBLIC void kb_init(tp)
tty_t *tp;
{
    /* Initialize the keyboard driver. */

    tp->tty_devread = kb_read;    /* input function */
}

/*=====
*                               kb_init_once                          *
*=====*/
PUBLIC void kb_init_once(void)
{
    int i;
    u8_t ccb;

    set_leds();    /* turn off numlock led */
    scan_keyboard(NULL, NULL);    /* discard leftover keystroke */

    /* Clear the function key observers array. Also see func_key(). */
    for (i=0; i<12; i++) {
        fkey_obs[i].proc_nr = NONE;    /* F1-F12 observers */
        fkey_obs[i].events = 0;    /* F1-F12 observers */
        sfkey_obs[i].proc_nr = NONE;    /* Shift F1-F12 observers */
        sfkey_obs[i].events = 0;    /* Shift F1-F12 observers */
    }

    kbd.minor= KBD_MINOR;
    kbdaux.minor= KBDAUX_MINOR;

```

```

/* Set interrupt handler and enable keyboard IRQ. */
irq_hook_id = KEYBOARD_IRQ; /* id to be returned on interrupt */
if ((i=sys_irqsetpolicy(KEYBOARD_IRQ, IRQ_REENABLE, &irq_hook_id)) != OK)
    panic("TTY", "Couldn't set keyboard IRQ policy", i);
if ((i=sys_irgenable(&irq_hook_id)) != OK)
    panic("TTY", "Couldn't enable keyboard IRQs", i);
kbd_irq_set |= (1 << KEYBOARD_IRQ);

/* Set AUX interrupt handler and enable AUX IRQ. */
aux_irq_hook_id = KBD_AUX_IRQ; /* id to be returned on interrupt */
if ((i=sys_irqsetpolicy(KBD_AUX_IRQ, IRQ_REENABLE,
    &aux_irq_hook_id)) != OK)
    panic("TTY", "Couldn't set AUX IRQ policy", i);
if ((i=sys_irgenable(&aux_irq_hook_id)) != OK)
    panic("TTY", "Couldn't enable AUX IRQs", i);
kbd_irq_set |= (1 << KBD_AUX_IRQ);

/* Disable the keyboard and aux */
kbc_cmd0(KBC_DI_KBD);
kbc_cmd0(KBC_DI_AUX);

/* Get the current configuration byte */
kbc_cmd0(KBC_RD_RAM_CCB);
ccb= kbc_read();

/* Enable both interrupts. */
kbc_cmd1(KBC_WR_RAM_CCB, ccb | 3);

/* Re-enable the keyboard device. */
kbc_cmd0(KBC_EN_KBD);

/* Enable the aux device. */
kbc_cmd0(KBC_EN_AUX);
}

/*=====
 *                                kbd_loadmap                                *
 *=====*/
PUBLIC int kbd_loadmap(m)
message *m;
{
/* Load a new keymap. */
int result;
result = sys_vircopy(m->IO_ENDPT, D, (vir_bytes) m->ADDRESS,
    SELF, D, (vir_bytes) keymap,
    (vir_bytes) sizeof(keymap));
return(result);
}

/*=====
 *                                do_fkey_ctl                                *
 *=====*/
PUBLIC void do_fkey_ctl(m_ptr)
message *m_ptr; /* pointer to the request message */
{
/* This procedure allows processes to register a function key to receive
 * notifications if it is pressed. At most one binding per key can exist.
 */
int i;
int result;

switch (m_ptr->FKEY_REQUEST) { /* see what we must do */
case FKEY_MAP: /* request for new mapping */
    result = OK; /* assume everything will be ok*/
    for (i=0; i < 12; i++) { /* check F1-F12 keys */
        if (bit_isset(m_ptr->FKEY_FKEYS, i+1) ) {
# if DEAD_CODE
/* Currently, we don't check if the slot is in use, so that IS
 * can recover after a crash by overtaking its existing mappings.
 * In future, a better solution will be implemented.
 */
            if (fkey_obs[i].proc_nr == NONE) {
# endif

```

```
        fkey_obs[i].proc_nr = m_ptr->m_source;
        fkey_obs[i].events = 0;
        bit_unset(m_ptr->FKEY_FKEYS, i+1);
#if DEAD_CODE
    } else {
        printf("WARNING, fkey_map failed F%d\n", i+1);
        result = EBUSY;        /* report failure, but try rest */
    }
#endif
    }
    for (i=0; i < 12; i++) {        /* check Shift+F1-F12 keys */
        if (bit_isset(m_ptr->FKEY_SFKEYS, i+1) ) {
#if DEAD_CODE
            if (sfkey_obs[i].proc_nr == NONE) {
#endif
                sfkey_obs[i].proc_nr = m_ptr->m_source;
                sfkey_obs[i].events = 0;
                bit_unset(m_ptr->FKEY_SFKEYS, i+1);
#if DEAD_CODE
            } else {
                printf("WARNING, fkey_map failed Shift F%d\n", i+1);
                result = EBUSY;        /* report failure but try rest */
            }
#endif
        }
    }
    break;
case FKEY_UNMAP:
    result = OK;
    for (i=0; i < 12; i++) {        /* check F1-F12 keys */
        if (bit_isset(m_ptr->FKEY_FKEYS, i+1) ) {
            if (fkey_obs[i].proc_nr == m_ptr->m_source) {
                fkey_obs[i].proc_nr = NONE;
                fkey_obs[i].events = 0;
                bit_unset(m_ptr->FKEY_FKEYS, i+1);
            } else {
                result = EPERM;        /* report failure, but try rest */
            }
        }
    }
    for (i=0; i < 12; i++) {        /* check Shift+F1-F12 keys */
        if (bit_isset(m_ptr->FKEY_SFKEYS, i+1) ) {
            if (sfkey_obs[i].proc_nr == m_ptr->m_source) {
                sfkey_obs[i].proc_nr = NONE;
                sfkey_obs[i].events = 0;
                bit_unset(m_ptr->FKEY_SFKEYS, i+1);
            } else {
                result = EPERM;        /* report failure, but try rest */
            }
        }
    }
    break;
case FKEY_EVENTS:
    m_ptr->FKEY_FKEYS = m_ptr->FKEY_SFKEYS = 0;
    for (i=0; i < 12; i++) {        /* check (Shift+) F1-F12 keys */
        if (fkey_obs[i].proc_nr == m_ptr->m_source) {
            if (fkey_obs[i].events) {
                bit_set(m_ptr->FKEY_FKEYS, i+1);
                fkey_obs[i].events = 0;
            }
        }
        if (sfkey_obs[i].proc_nr == m_ptr->m_source) {
            if (sfkey_obs[i].events) {
                bit_set(m_ptr->FKEY_SFKEYS, i+1);
                sfkey_obs[i].events = 0;
            }
        }
    }
    break;
default:
    result = EINVAL;        /* key cannot be observed */
}
```

```

/* Almost done, return result to caller. */
m_ptr->m_type = result;
send(m_ptr->m_source, m_ptr);
}

/*=====
 *                               func_key                               *
 *=====*/
PRIVATE int func_key(scode)
int scode;                               /* scan code for a function key */
{
/* This procedure traps function keys for debugging purposes. Observers of
 * function keys are kept in a global array. If a subject (a key) is pressed
 * the observer is notified of the event. Initialization of the arrays is done
 * in kb_init, where NONE is set to indicate there is no interest in the key.
 * Returns FALSE on a key release or if the key is not observable.
 */
message m;
int key;
int proc_nr;
int i,s;

/* Ignore key releases. If this is a key press, get full key code. */
if (scode & KEY_RELEASE) return(FALSE); /* key release */
key = map_key(scode);                  /* include modifiers */

/* Key pressed, now see if there is an observer for the pressed key.
 *      F1-F12   observers are in fkey_obs array.
 *      SHIFT   F1-F12   observers are in sfkey_req array.
 *      CTRL    F1-F12   reserved (see kb_read)
 *      ALT     F1-F12   reserved (see kb_read)
 * Other combinations are not in use. Note that Alt+Shift+F1-F12 is yet
 * defined in <minix/keymap.h>, and thus is easy for future extensions.
 */
if (F1 <= key && key <= F12) {          /* F1-F12 */
    proc_nr = fkey_obs[key - F1].proc_nr;
    fkey_obs[key - F1].events ++ ;
} else if (SF1 <= key && key <= SF12) { /* Shift F2-F12 */
    proc_nr = sfkey_obs[key - SF1].proc_nr;
    sfkey_obs[key - SF1].events ++;
}
else {
    return(FALSE);                      /* not observable */
}

/* See if an observer is registered and send it a message. */
if (proc_nr != NONE) {
    m.NOTIFY_TYPE = FKEY_PRESSED;
    notify(proc_nr);
}
return(TRUE);
}

/*=====
 *                               show_key_mappings                       *
 *=====*/
PRIVATE void show_key_mappings()
{
    int i,s;
    struct proc proc;

    printf("\n");
    printf("System information.  Known function key mappings to request debug dumps:\n" );
    printf("-----\n");
    for (i=0; i<12; i++) {
        printf(" %sF%d: ", i+1<10? " ":"", i+1);
        if (fkey_obs[i].proc_nr != NONE) {
            if ((s=sys_getproc(&proc, fkey_obs[i].proc_nr))!=OK)
                printf("sys_getproc: %d\n", s);
            printf("%-14.14s", proc.p_name);
        } else {
            printf("%-14.14s", "<none>");
        }
    }
}

```

```

    printf(" %sShift-F%d: ", i+1<10? " ":"", i+1);
    if (sfkey_obs[i].proc_nr != NONE) {
        if ((s=sys_getproc(&proc, sfkey_obs[i].proc_nr))!=OK)
            printf("sys_getproc: %d\n", s);
        printf("%-14.14s", proc.p_name);
    } else {
        printf("%-14.14s", "<none>");
    }
    printf("\n");
}
printf("\n");
printf("Press one of the registered function keys to trigger a debug dump.\n");
printf("\n");
}

/*=====
*
* scan_keyboard
*=====*/
PRIVATE int scan_keyboard(bp, isauxp)
unsigned char *bp;
int *isauxp;
{
    #if 0 /* Is this old XT code? It doesn't match the PS/2 hardware */
    /* Fetch the character from the keyboard hardware and acknowledge it. */
    pvb_pair_t byte_in[2], byte_out[2];

    byte_in[0].port = KEYBD; /* get the scan code for the key struck */
    byte_in[1].port = PORT_B; /* strobe the keyboard to ack the char */
    sys_vinb(byte_in, 2); /* request actual input */

    pv_set(byte_out[0], PORT_B, byte_in[1].value | KBIT); /* strobe bit high */
    pv_set(byte_out[1], PORT_B, byte_in[1].value); /* then strobe low */
    sys_voutb(byte_out, 2); /* request actual output */

    return(byte_in[0].value); /* return scan code */
    #else
    unsigned long b, sb;

    sys_inb(KB_STATUS, &sb);
    if (!(sb & KB_OUT_FULL))
    {
        if (kbdout.avail && !kbdout.expect_ack)
            kbd_send();
        return 0;
    }
    sys_inb(KEYBD, &b);
    #if 0
    printf("got byte 0x%x from %s\n", b, (sb & KB_AUX_BYTE) ? "AUX" : "keyboard");
    #endif
    if (!(sb & KB_AUX_BYTE) && b == KB_ACK && kbdout.expect_ack)
    {
    #if 1
        printf("got ACK from keyboard\n");
    #endif
        kbdout.expect_ack= 0;
        micro_delay(KBC_IN_DELAY);
        kbd_send();
        return 0;
    }
    if (bp)
        *bp= b;
    if (isauxp)
        *isauxp= !(sb & KB_AUX_BYTE);
    if (kbdout.avail && !kbdout.expect_ack)
    {
        micro_delay(KBC_IN_DELAY);
        kbd_send();
    }
    return 1;
    #endif
}

static void micro_delay(unsigned long usecs)

```

```
{
    tickdelay(MICROS_TO_TICKS(usecs));
}

/*=====
 *                               kbd_watchdog                               *
 *=====*/
PRIVATE void kbd_watchdog(tmrp)
timer_t *tmrp;
{
    int r;
    clock_t now;

    kbd_watchdog_set= 0;
    if (!kbdout.avail)
        return; /* Watchdog is no longer needed */
    if (!kbd_alive)
    {
        printf("kbd_watchdog: should reset keyboard\n");
    }
    kbd_alive= 0;

    if ((r= getuptime(&now)) != OK)
        panic("TTY", "Keyboard couldn't get clock's uptime.", r);
    tmrs_settimer(&tty_timers, &tmr_kbd_wd, now+HZ, kbd_watchdog,
        NULL);
    if (tty_timers->tmr_exp_time != tty_next_timeout) {
        tty_next_timeout = tty_timers->tmr_exp_time;
        if ((r= sys_setalarm(tty_next_timeout, 1)) != OK)
            panic("TTY", "Keyboard couldn't set alarm.", r);
    }
    kbd_watchdog_set= 1;
}
```

```

/*      pty.c - pseudo terminal driver                                Author: Kees J. Bot
*                                                              30 Dec 1995
* PTYS can be seen as a bidirectional pipe with TTY
* input and output processing.  For example a simple rlogin session:
*
*      keyboard -> rlogin -> in.rld -> /dev/ptypX -> /dev/ttypX -> shell
*      shell -> /dev/ttypX -> /dev/ptypX -> in.rld -> rlogin -> screen
*
* This file takes care of copying data between the tty/pty device pairs and
* the open/read/write/close calls on the pty devices.  The TTY task takes
* care of the input and output processing (interrupt, backspace, raw I/O,
* etc.) using the pty_read() and pty_write() functions as the "keyboard" and
* "screen" functions of the ttypX devices.
* Be careful when reading this code, the terms "reading" and "writing" are
* used both for the tty and the pty end of the pseudo tty.  Writes to one
* end are to be read at the other end and vice-versa.
*/

#include "../drivers.h"
#include <assert.h>
#include <termios.h>
#include <signal.h>
#include <minix/com.h>
#include <minix/callnr.h>
#include <sys/select.h>
#include "tty.h"

#if NR_PTYS > 0

/* PTY bookkeeping structure, one per pty/tty pair. */
typedef struct pty {
    tty_t      *tty;          /* associated TTY structure */
    char       state;         /* flags: busy, closed, ... */

    /* Read call on /dev/ptypX. */
    char       rdsendreply;   /* send a reply (instead of notify) */
    int        rdcaller;      /* process making the call (usually FS) */
    int        rdproc;        /* process that wants to read from the pty */
    vir_bytes  rdvir;         /* virtual address in readers address space */
    int        rdleft;        /* # bytes yet to be read */
    int        rdcum;         /* # bytes written so far */

    /* Write call to /dev/ptypX. */
    char       wrsendreply;   /* send a reply (instead of notify) */
    int        wrcaller;      /* process making the call (usually FS) */
    int        wrproc;        /* process that wants to write to the pty */
    vir_bytes  wrvir;         /* virtual address in writers address space */
    int        wrleft;        /* # bytes yet to be written */
    int        wrcum;         /* # bytes written so far */

    /* Output buffer. */
    int        ocount;        /* # characters in the buffer */
    char       *ohead, *otail; /* head and tail of the circular buffer */
    char       obuf[128];     /* buffer for bytes going to the pty reader */

    /* select() data. */
    int        select_ops,    /* Which operations do we want to know about? */
    select_proc, /* Who wants to know about it? */
    select_ready_ops;         /* For callback. */
} pty_t;

#define PTY_ACTIVE      0x01    /* pty is open/active */
#define TTY_CLOSED      0x02    /* tty side has closed down */
#define PTY_CLOSED      0x04    /* pty side has closed down */

PRIVATE pty_t pty_table[NR_PTYS];          /* PTY bookkeeping */

FORWARD _PROTOTYPE( int pty_write, (tty_t *tp, int try) )
FORWARD _PROTOTYPE( void pty_echo, (tty_t *tp, int c) )
FORWARD _PROTOTYPE( void pty_start, (pty_t *pp) )
FORWARD _PROTOTYPE( void pty_finish, (pty_t *pp) )
FORWARD _PROTOTYPE( int pty_read, (tty_t *tp, int try) )
FORWARD _PROTOTYPE( int pty_close, (tty_t *tp, int try) )
FORWARD _PROTOTYPE( int pty_icancel, (tty_t *tp, int try) )

```

```

FORWARD _PROTOTYPE( int pty_ocancel, (tty_t *tp, int try) ) ;
FORWARD _PROTOTYPE( int pty_select, (tty_t *tp, message *m) ) ;

/*=====
 *                               do_pty                               *
 *=====*/
PUBLIC void do_pty(tp, m_ptr)
tty_t *tp;
message *m_ptr;
{
/* Perform an open/close/read/write call on a /dev/ptypX device. */
pty_t *pp = tp->tty_priv;
int r;
phys_bytes p;

switch (m_ptr->m_type) {
case DEV_READ:
/* Check, store information on the reader, do I/O. */
if (pp->state & TTY_CLOSED) {
r = 0;
break;
}
if (pp->rdleft != 0 || pp->rdcum != 0) {
r = EIO;
break;
}
if (m_ptr->COUNT <= 0) {
r = EINVAL;
break;
}
#if DEAD_CODE
if (numap_local(m_ptr->IO_ENDPT, (vir_bytes) m_ptr->ADDRESS,
m_ptr->COUNT) == 0) {
#else
if ((r = sys_umap(m_ptr->IO_ENDPT, D, (vir_bytes) m_ptr->ADDRESS,
m_ptr->COUNT, &p)) != OK) {
#endif
break;
}
pp->rdsendreply = TRUE;
pp->rdcaller = m_ptr->m_source;
pp->rdproc = m_ptr->IO_ENDPT;
pp->rdvir = (vir_bytes) m_ptr->ADDRESS;
pp->rdleft = m_ptr->COUNT;
pty_start(pp);
handle_events(tp);
if (pp->rdleft == 0) return; /* already done */

if (m_ptr->TTY_FLAGS & O_NONBLOCK) {
r = EAGAIN; /* don't suspend */
pp->rdleft = pp->rdcum = 0;
} else {
r = SUSPEND; /* do suspend */
pp->rdsendreply = FALSE;
}
break;

case DEV_WRITE:
/* Check, store information on the writer, do I/O. */
if (pp->state & TTY_CLOSED) {
r = EIO;
break;
}
if (pp->wrleft != 0 || pp->wrcum != 0) {
r = EIO;
break;
}
if (m_ptr->COUNT <= 0) {
r = EINVAL;
break;
}
#if DEAD_CODE
if (numap_local(m_ptr->IO_ENDPT, (vir_bytes) m_ptr->ADDRESS,
m_ptr->COUNT) == 0) {

```



```

        r = EFAULT;
#else
    if ((r = sys_umap(m_ptr->IO_ENDPT, D, (vir_bytes) m_ptr->ADDRESS,
        m_ptr->COUNT, &p)) != OK) {
#endif
        break;
    }
    pp->wrsendreply = TRUE;
    pp->wrcaller = m_ptr->m_source;
    pp->wrproc = m_ptr->IO_ENDPT;
    pp->wrvir = (vir_bytes) m_ptr->ADDRESS;
    pp->wrleft = m_ptr->COUNT;
    handle_events(tp);
    if (pp->wrleft == 0) return;          /* already done */

    if (m_ptr->TTY_FLAGS & O_NONBLOCK) { /* don't suspend */
        r = pp->wrcum > 0 ? pp->wrcum : EAGAIN;
        pp->wrleft = pp->wrcum = 0;
    } else {
        pp->wrsendreply = FALSE;        /* do suspend */
        r = SUSPEND;
    }
    break;

case DEV_OPEN:
    r = pp->state != 0 ? EIO : OK;
    pp->state |= PTY_ACTIVE;
    pp->rdcum = 0;
    pp->wrcum = 0;
    break;

case DEV_CLOSE:
    r = OK;
    if (pp->state & TTY_CLOSED) {
        pp->state = 0;
    } else {
        pp->state |= PTY_CLOSED;
        sigchar(tp, SIGHUP);
    }
    break;

case DEV_SELECT:
    r = pty_select(tp, m_ptr);
    break;

case CANCEL:
    if (m_ptr->IO_ENDPT == pp->rdproc) {
        /* Cancel a read from a PTY. */
        pp->rdleft = pp->rdcum = 0;
    }
    if (m_ptr->IO_ENDPT == pp->wrproc) {
        /* Cancel a write to a PTY. */
        pp->wrleft = pp->wrcum = 0;
    }
    r = EINTR;
    break;

default:
    r = EINVAL;
}
tty_reply(TASK_REPLY, m_ptr->m_source, m_ptr->IO_ENDPT, r);
}

/*=====
 *                               pty_write                               *
 *=====*/
PRIVATE int pty_write(tp, try)
tty_t *tp;
int try;
{
    /* (*dev_write)() routine for PTYs. Transfer bytes from the writer on
    * /dev/ttypX to the output buffer.
    */
    pty_t *pp = tp->tty_priv;

```

```

int count, ocount, s;
phys_bytes user_phys;

/* PTY closed down? */
if (pp->state & PTY_CLOSED) {
    if (try) return 1;
    if (tp->tty_outleft > 0) {
        tty_reply(tp->tty_outrepcode, tp->tty_outcaller,
                  tp->tty_outproc, EIO);
        tp->tty_outleft = tp->tty_outcum = 0;
    }
    return;
}

/* While there is something to do. */
for (;;) {
    ocount = buflen(pp->obuf) - pp->ocount;
    if (try) return (ocount > 0);
    count = bufend(pp->obuf) - pp->ohhead;
    if (count > ocount) count = ocount;
    if (count > tp->tty_outleft) count = tp->tty_outleft;
    if (count == 0 || tp->tty_inhibited)
        break;

    /* Copy from user space to the PTY output buffer. */
    if ((s = sys_vircopy(tp->tty_outproc, D, (vir_bytes) tp->tty_out_vir,
                        SELF, D, (vir_bytes) pp->ohhead, (phys_bytes) count)) != OK) {
        printf("pty tty%d: copy failed (error %d)\n", s);
        break;
    }

    /* Perform output processing on the output buffer. */
    out_process(tp, pp->obuf, pp->ohhead, bufend(pp->obuf), &count, &ocount);
    if (count == 0) break;

    /* Assume echoing messed up by output. */
    tp->tty_reprint = TRUE;

    /* Bookkeeping. */
    pp->ocount += ocount;
    if ((pp->ohhead += ocount) >= bufend(pp->obuf))
        pp->ohhead -= buflen(pp->obuf);
    pty_start(pp);
    tp->tty_out_vir += count;
    tp->tty_outcum += count;
    if ((tp->tty_outleft -= count) == 0) {
        /* Output is finished, reply to the writer. */
        tty_reply(tp->tty_outrepcode, tp->tty_outcaller,
                  tp->tty_outproc, tp->tty_outcum);
        tp->tty_outcum = 0;
    }
}
pty_finish(pp);
return 1;
}

/*=====
*                               pty_echo                               *
*=====*/
PRIVATE void pty_echo(tp, c)
tty_t *tp;
int c;
{
    /* Echo one character. (Like pty_write, but only one character, optionally.) */

    pty_t *pp = tp->tty_priv;
    int count, ocount;

    ocount = buflen(pp->obuf) - pp->ocount;
    if (ocount == 0) return; /* output buffer full */
    count = 1;
    *pp->ohhead = c; /* add one character */

    out_process(tp, pp->obuf, pp->ohhead, bufend(pp->obuf), &count, &ocount);

```

```

    if (count == 0) return;

    pp->ocount += ocount;
    if ((pp->ohead += ocount) >= bufend(pp->obuf)) pp->ohead -= buflen(pp->obuf);
    pty_start(pp);
}

/*=====
 *                               pty_start                               *
 *=====*/
PRIVATE void pty_start(pp)
pty_t *pp;
{
    /* Transfer bytes written to the output buffer to the PTY reader. */
    int count;

    /* While there are things to do. */
    for (;;) {
        int s;
        count = bufend(pp->obuf) - pp->otail;
        if (count > pp->ocount) count = pp->ocount;
        if (count > pp->rdleft) count = pp->rdleft;
        if (count == 0) break;

        /* Copy from the output buffer to the readers address space. */
        if ((s = sys_vircopy(SELF, D, (vir_bytes)pp->otail,
            (vir_bytes) pp->rdproc, D, (vir_bytes) pp->rdvir, (phys_bytes) count)) !=
OK) {
            printf("pty tty%d: copy failed (error %d)\n", s);
            break;
        }

        /* Bookkeeping. */
        pp->ocount -= count;
        if ((pp->otail += count) == bufend(pp->obuf)) pp->otail = pp->obuf;
        pp->rdvir += count;
        pp->rdcum += count;
        pp->rdleft -= count;
    }
}

/*=====
 *                               pty_finish                               *
 *=====*/
PRIVATE void pty_finish(pp)
pty_t *pp;
{
    /* Finish the read request of a PTY reader if there is at least one byte
    * transferred.
    */
    if (pp->rdcum > 0) {
        if (pp->rdsendreply) {
            tty_reply(TASK_REPLY, pp->rdcaller, pp->rdproc, pp->rdcum);
            pp->rdleft = pp->rdcum = 0;
        }
        else
            notify(pp->rdcaller);
    }
}

/*=====
 *                               pty_read                               *
 *=====*/
PRIVATE int pty_read(tp, try)
tty_t *tp;
int try;
{
    /* Offer bytes from the PTY writer for input on the TTY. (Do it one byte at
    * a time, 99% of the writes will be for one byte, so no sense in being smart.)
    */
    pty_t *pp = tp->tty_priv;
    char c;

```

```

if (pp->state & PTY_CLOSED) {
    if (try) return 1;
    if (tp->tty_inleft > 0) {
        tty_reply(tp->tty_inrepcode, tp->tty_incaller, tp->tty_inproc,
                  tp->tty_incum);
        tp->tty_inleft = tp->tty_incum = 0;
    }
    return 1;
}

if (try) {
    if (pp->wrleft > 0)
        return 1;
    return 0;
}

while (pp->wrleft > 0) {
    int s;

    /* Transfer one character to 'c'. */
    if ((s = sys_vircopy(pp->wrproc, D, (vir_bytes) pp->wrvir,
                        SELF, D, (vir_bytes) &c, (phys_bytes) 1)) != OK) {
        printf("pty: copy failed (error %d)\n", s);
        break;
    }

    /* Input processing. */
    if (in_process(tp, &c, 1) == 0) break;

    /* PTY writer bookkeeping. */
    pp->wrvir++;
    pp->wrcum++;
    if (--pp->wrleft == 0) {
        if (pp->wrsendreply) {
            tty_reply(TASK_REPLY, pp->wrcaller, pp->wrproc,
                    pp->wrcum);
            pp->wrcum = 0;
        }
        else
            notify(pp->wrcaller);
    }
}

}

/*=====
*                               pty_close                               *
*=====*/
PRIVATE int pty_close(tp, try)
tty_t *tp;
int try;
{
    /* The tty side has closed, so shut down the pty side. */
    pty_t *pp = tp->tty_priv;

    if (!(pp->state & PTY_ACTIVE)) return;

    if (pp->rdleft > 0) {
        assert(!pp->rdsendreply);
        notify(pp->rdcaller);
    }

    if (pp->wrleft > 0) {
        assert(!pp->wrsendreply);
        notify(pp->wrcaller);
    }

    if (pp->state & PTY_CLOSED) pp->state = 0; else pp->state |= TTY_CLOSED;
}

/*=====
*                               pty_icancel                               *
*=====*/
PRIVATE int pty_icancel(tp, try)
tty_t *tp;

```

```

int try;
{
/* Discard waiting input. */
pty_t *pp = tp->tty_priv;

    if (pp->wrleft > 0) {
        assert(!pp->wrsendreply);
        pp->wrcum += pp->wrleft;
        pp->wrleft = 0;
        notify(pp->wrcaller);
    }
}

/*=====
 *                               pty_ocancel                               *
 *=====*/
PRIVATE int pty_ocancel(tp, try)
tty_t *tp;
int try;
{
/* Drain the output buffer. */
pty_t *pp = tp->tty_priv;

    pp->ocount = 0;
    pp->otail = pp->ohed;
}

/*=====
 *                               pty_init                               *
 *=====*/
PUBLIC void pty_init(tp)
tty_t *tp;
{
    pty_t *pp;
    int line;

/* Associate PTY and TTY structures. */
    line = tp - &tty_table[NR_CONS + NR_RS_LINES];
    pp = tp->tty_priv = &pty_table[line];
    pp->tty = tp;
    pp->select_ops = 0;

/* Set up output queue. */
    pp->ohed = pp->otail = pp->obuf;

/* Fill in TTY function hooks. */
    tp->tty_devread = pty_read;
    tp->tty_devwrite = pty_write;
    tp->tty_echo = pty_echo;
    tp->tty_icancel = pty_icancel;
    tp->tty_ocancel = pty_ocancel;
    tp->tty_close = pty_close;
    tp->tty_select_ops = 0;
}

/*=====
 *                               pty_status                               *
 *=====*/
PUBLIC int pty_status(message *m_ptr)
{
    int i, event_found;
    pty_t *pp;

    event_found = 0;
    for (i = 0, pp = pty_table; i < NR_PTYS; i++, pp++) {
        if (((pp->state & TTY_CLOSED) && pp->rdleft > 0) ||
            pp->rdcum > 0) &&
            pp->rdcaller == m_ptr->m_source)
        {
            m_ptr->m_type = DEV_REVIVE;
            m_ptr->REP_ENDPT = pp->rdproc;
            m_ptr->REP_STATUS = pp->rdcum;

            pp->rdleft = pp->rdcum = 0;

```

```

        event_found = 1;
        break;
    }

    if (((pp->state & TTY_CLOSED) && pp->wrleft > 0) ||
        pp->wrcum > 0) &&
        pp->wrcaller == m_ptr->m_source)
    {
        m_ptr->m_type = DEV_REVIVE;
        m_ptr->REP_ENDPT = pp->wrproc;
        if (pp->wrcum == 0)
            m_ptr->REP_STATUS = EIO;
        else
            m_ptr->REP_STATUS = pp->wrcum;

        pp->wrleft = pp->wrcum = 0;
        event_found = 1;
        break;
    }

    if (pp->select_ready_ops && pp->select_proc == m_ptr->m_source) {
        m_ptr->m_type = DEV_IO_READY;
        m_ptr->DEV_MINOR = PTYPX_MINOR + i;
        m_ptr->DEV_SEL_OPS = pp->select_ready_ops;
        pp->select_ready_ops = 0;
        event_found = 1;
        break;
    }
}
return event_found;
}

/*=====
 *                               select_try_pty                               *
 *=====*/
PRIVATE int select_try_pty(tty_t *tp, int ops)
{
    pty_t *pp = tp->tty_priv;
    int r = 0;

    if (ops & SEL_WR) {
        /* Write won't block on error. */
        if (pp->state & TTY_CLOSED) r |= SEL_WR;
        else if (pp->wrleft != 0 || pp->wrcum != 0) r |= SEL_WR;
        else r |= SEL_WR;
    }

    if (ops & SEL_RD) {
        /* Read won't block on error. */
        if (pp->state & TTY_CLOSED) r |= SEL_RD;
        else if (pp->rdleft != 0 || pp->rdcum != 0) r |= SEL_RD;
        else if (pp->ocount > 0) r |= SEL_RD; /* Actual data. */
    }

    return r;
}

/*=====
 *                               select_retry_pty                               *
 *=====*/
PUBLIC void select_retry_pty(tty_t *tp)
{
    pty_t *pp = tp->tty_priv;
    int r;

    /* See if the pty side of a pty is ready to return a select. */
    if (pp->select_ops && (r=select_try_pty(tp, pp->select_ops))) {
        pp->select_ops &= ~r;
        pp->select_ready_ops |= r;
        notify(pp->select_proc);
    }
}

/*=====

```

```

*                               pty_select                               *
*=====*/
PRIVATE int pty_select(tty_t *tp, message *m)
{
    pty_t *pp = tp->tty_priv;
    int ops, ready_ops = 0, watch;

    ops = m->IO_ENDPT & (SEL_RD|SEL_WR|SEL_ERR);
    watch = (m->IO_ENDPT & SEL_NOTIFY) ? 1 : 0;

    ready_ops = select_try_pty(tp, ops);

    if (!ready_ops && ops && watch) {
        pp->select_ops |= ops;
        pp->select_proc = m->m_source;
    }

    return ready_ops;
}

#endif /* NR_PTYS > 0 */
```

```
#include <minix/config.h>
/*-----*
 *          rs232.c - serial driver for 8250 and 16450 UARTs      *
 *          Added support for Atari ST M68901 and YM-2149    --kub  *
 *-----*/

#include "../drivers.h"
#include <termios.h>
#include <signal.h>
#include "tty.h"

#if NR_RS_LINES > 0

#if (MACHINE != IBM_PC) && (MACHINE != ATARI)
#error /* rs232.c only supports PC and Atari ST */
#endif

#if (MACHINE == ATARI)
#include "staddr.h"
#include "stsound.h"
#include "stmfp.h"
#if (NR_RS_LINES > 1)
#error /* Only one physical RS232 line available */
#endif
#endif

#if (MACHINE == IBM_PC) /* PC/AT 8250/16450 chip combination */

/* 8250 constants. */
#define UART_FREQ          115200L /* timer frequency */

/* Interrupt enable bits. */
#define IE_RECEIVER_READY  1
#define IE_TRANSMITTER_READY 2
#define IE_LINE_STATUS_CHANGE 4
#define IE_MODEM_STATUS_CHANGE 8

/* Interrupt status bits. */
#define IS_MODEM_STATUS_CHANGE 0
#define IS_TRANSMITTER_READY 2
#define IS_RECEIVER_READY 4
#define IS_LINE_STATUS_CHANGE 6

/* Line control bits. */
#define LC_2STOP_BITS      0x04
#define LC_PARITY          0x08
#define LC_PAREVEN        0x10
#define LC_BREAK          0x40
#define LC_ADDRESS_DIVISOR 0x80

/* Line status bits. */
#define LS_OVERRUN_ERR      2
#define LS_PARITY_ERR      4
#define LS_FRAMING_ERR     8
#define LS_BREAK_INTERRUPT 0x10
#define LS_TRANSMITTER_READY 0x20

/* Modem control bits. */
#define MC_DTR              1
#define MC_RTS              2
#define MC_OUT2             8 /* required for PC & AT interrupts */

/* Modem status bits. */
#define MS_CTS              0x10
#define MS_RLSD             0x80 /* Received Line Signal Detect */
#define MS_DRLSD            0x08 /* RLSD Delta */

#else /* MACHINE == ATARI */ /* Atari ST 68901 USART */

/* Most of the USART constants are already defined in stmfp.h . The local
 * definitions made here are for keeping C code changes smaller.  --kub
 */

#define UART_FREQ          19200L /* timer frequency */
```



```

/* Line status bits. */
#define LS_OVERRUN_ERR      R_OE
#define LS_PARITY_ERR       R_PE
#define LS_FRAMING_ERR      R_FE
#define LS_BREAK_INTERRUPT  R_BREAK

/* Modem status bits. */
#define MS_CTS              IO_SCTS      /* 0x04 */

#endif /* MACHINE == ATARI */

#define DATA_BITS_SHIFT    8           /* amount data bits shifted in mode */
#define DEF_BAUD             1200       /* default baud rate */

#define RS_IBUFSIZE          1024       /* RS232 input buffer size */
#define RS_OBUFSIZE          1024       /* RS232 output buffer size */

/* Input buffer watermarks.
 * The external device is asked to stop sending when the buffer
 * exactly reaches high water, or when TTY requests it. Sending restarts
 * when the input buffer empties below the low watermark.
 */
#define RS_ILOWWATER        (1 * RS_IBUFSIZE / 4)
#define RS_IHIGHWATER       (3 * RS_IBUFSIZE / 4)

/* Output buffer low watermark.
 * TTY is notified when the output buffer empties below the low watermark, so
 * it may continue filling the buffer if doing a large write.
 */
#define RS_OLOWWATER        (1 * RS_OBUFSIZE / 4)

#if (MACHINE == IBM_PC)

/* Macros to handle flow control.
 * Interrupts must be off when they are used.
 * Time is critical - already the function call for outb() is annoying.
 * If outb() can be done in-line, tests to avoid it can be dropped.
 * istart() tells external device we are ready by raising RTS.
 * istop() tells external device we are not ready by dropping RTS.
 * DTR is kept high all the time (it probably should be raised by open and
 * dropped by close of the device).
 * OUT2 is also kept high all the time.
 */
#define istart(rs) \
    (sys_outb((rs)->modem_ctl_port, MC_OUT2 | MC_RTS | MC_DTR), \
     (rs)->idevready = TRUE)
#define istop(rs) \
    (sys_outb((rs)->modem_ctl_port, MC_OUT2 | MC_DTR), \
     (rs)->idevready = FALSE)

/* Macro to tell if device is ready. The rs->cts field is set to MS_CTS if
 * CLOCAL is in effect for a line without a CTS wire.
 */
#define devready(rs) ((my_inb(rs->modem_status_port) | rs->cts) & MS_CTS)

/* Macro to tell if transmitter is ready. */
#define txready(rs) (my_inb(rs->line_status_port) & LS_TRANSMITTER_READY)

/* Macro to tell if carrier has dropped.
 * The RS232 Carrier Detect (CD) line is usually connected to the 8250
 * Received Line Signal Detect pin, reflected by bit MS_RLSD in the Modem
 * Status Register. The MS_DRLSD bit tells if MS_RLSD has just changed state.
 * So if MS_DRLSD is set and MS_RLSD cleared, we know that carrier has just
 * dropped.
 */
#define devhup(rs) \
    ((my_inb(rs->modem_status_port) & (MS_RLSD|MS_DRLSD)) == MS_DRLSD)

#else /* MACHINE == ATARI */

/* Macros to handle flow control.
 * Time is critical - already the function call for lock()/restore() is
 * annoying.

```

```

* istart() tells external device we are ready by raising RTS.
* istop() tells external device we are not ready by dropping RTS.
* DTR is kept high all the time (it probably should be raised by open and
* dropped by close of the device). NOTE: The modem lines are active low.
*/
#define set_porta(msk,val) { register int s = lock();          \
                           SOUND->sd_selr = YM_IOA;           \
                           SOUND->sd_wdat =                    \
                               SOUND->sd_rdat & (msk) | (val); \
                           restore(s);                        \
#define istart(rs)        { set_porta( ~(PA_SRTS|PA_SDTR),0 ); \
                           (rs)->idevready = TRUE;           \
#define istop(rs)         { set_porta( ~PA_SDTR, PA_SRTS );   \
                           (rs)->idevready = FALSE;           \

/* Macro to tell if device is ready. The rs->cts field is set to MS_CTS if
* CLOCAL is in effect for a line without a CTS wire.
*/
#define devready(rs)      ((~MFP->mf_gpip | rs->cts) & MS_CTS)

/* Transmitter ready test */
#define txready(rs)       (MFP->mf_tsr & (T_EMPTY | T_UE))

#endif /* MACHINE == ATARI */

/* Types. */
typedef unsigned char bool_t; /* boolean */

/* RS232 device structure, one per device. */
typedef struct rs232 {
    tty_t *tty; /* associated TTY structure */

    int icount; /* number of bytes in the input buffer */
    char *ihead; /* next free spot in input buffer */
    char *itail; /* first byte to give to TTY */
    bool_t idevready; /* nonzero if we are ready to receive (RTS) */
    char cts; /* normally 0, but MS_CTS if CLOCAL is set */

    unsigned char ostate; /* combination of flags: */
#define ODONE 1 /* output completed (< output enable bits) */
#define ORAW 2 /* raw mode for xoff disable (< enab. bits) */
#define OWAKEUP 4 /* tty_wakeup() pending (asm code only) */
#define ODEVREADY MS_CTS /* external device hardware ready (CTS) */
#define OQUEUED 0x20 /* output buffer not empty */
#define OSWREADY 0x40 /* external device software ready (no xoff) */
#define ODEVHUP MS_RLSD /* external device has dropped carrier */
#define OSOFTBITS (ODONE | ORAW | OWAKEUP | OQUEUED | OSWREADY) /* user-defined bits */
#if (OSOFTBITS | ODEVREADY | ODEVHUP) == OSOFTBITS /* a weak sanity check */
#error /* bits are not unique */
#endif
    unsigned char oxoff; /* char to stop output */
    bool_t inhibited; /* output inhibited? (follows tty_inhibited) */
    bool_t drain; /* if set drain output and reconfigure line */
    int ocount; /* number of bytes in the output buffer */
    char *ohead; /* next free spot in output buffer */
    char *otail; /* next char to output */

#if (MACHINE == IBM_PC)
    port_t xmit_port; /* i/o ports */
    port_t recv_port;
    port_t div_low_port;
    port_t div_hi_port;
    port_t int_enab_port;
    port_t int_id_port;
    port_t line_ctl_port;
    port_t modem_ctl_port;
    port_t line_status_port;
    port_t modem_status_port;
#endif
#define
    unsigned char lstatus; /* last line status */
    unsigned char pad; /* ensure alignment for 16-bit ints */

```

```

unsigned framing_errors;      /* error counts (no reporting yet) */
unsigned overrun_errors;
unsigned parity_errors;
unsigned break_interrupts;

int irq;                      /* irq for this line */
int irq_hook_id;              /* interrupt hook */

char ibuf[RS_IBUFSIZE];       /* input buffer */
char obuf[RS_OBUFSIZE];       /* output buffer */
} rs232_t;

PUBLIC rs232_t rs_lines[NR_RS_LINES];

/* Table and macro to translate an RS232 line number to its rs_lines entry. */
PRIVATE rs232_t *p_rs_addr[NR_RS_LINES];

#define rs_addr(line) (p_rs_addr[line])

#if (MACHINE == IBM_PC)
/* 8250 base addresses. */
PRIVATE port_t addr_8250[] = {
    0x3F8,      /* COM1 */
    0x2F8,      /* COM2 */
    0x3E8,      /* COM3 */
    0x2E8,      /* COM4 */
};
#endif

FORWARD _PROTOTYPE( void in_int, (rs232_t *rs) ) ;
FORWARD _PROTOTYPE( void line_int, (rs232_t *rs) ) ;
FORWARD _PROTOTYPE( void modem_int, (rs232_t *rs) ) ;
FORWARD _PROTOTYPE( int rs_write, (tty_t *tp, int try) ) ;
FORWARD _PROTOTYPE( void rs_echo, (tty_t *tp, int c) ) ;
FORWARD _PROTOTYPE( int rs_ioctl, (tty_t *tp, int try) ) ;
FORWARD _PROTOTYPE( void rs_config, (rs232_t *rs) ) ;
FORWARD _PROTOTYPE( int rs_read, (tty_t *tp, int try) ) ;
FORWARD _PROTOTYPE( int rs_icancel, (tty_t *tp, int try) ) ;
FORWARD _PROTOTYPE( int rs_ocancel, (tty_t *tp, int try) ) ;
FORWARD _PROTOTYPE( void rs_ostart, (rs232_t *rs) ) ;
FORWARD _PROTOTYPE( int rs_break, (tty_t *tp, int try) ) ;
FORWARD _PROTOTYPE( int rs_close, (tty_t *tp, int try) ) ;
FORWARD _PROTOTYPE( void out_int, (rs232_t *rs) ) ;
FORWARD _PROTOTYPE( void rs232_handler, (rs232_t *rs) ) ;

/* XXX */
PRIVATE void lock(void) {}
PRIVATE void unlock(void) {}

PRIVATE int my_inb(port_t port)
{
    int r;
    unsigned long v = 0;
    r = sys_inb(port, &v);
    if (r != OK)
        printf("RS232 warning: failed inb 0x%x\n", port);

    return (int) v;
}

/*=====
 *                               rs_write                               *
 *=====*/
PRIVATE int rs_write(tp, try)
register tty_t *tp;
int try;
{
    /* (*devwrite)() routine for RS232. */

    rs232_t *rs = tp->tty_priv;
    int count, ocount;

    if (rs->inhibited != tp->tty_inhibited) {
        /* Inhibition state has changed. */
    }

```

```

        lock();
        rs->ostate |= OSWREADY;
        if (tp->tty_inhibited) rs->ostate &= ~OSWREADY;
        unlock();
        rs->inhibited = tp->tty_inhibited;
    }

    if (rs->drain) {
        /* Wait for the line to drain then reconfigure and continue output. */
        if (rs->ocount > 0) return 0;
        rs->drain = FALSE;
        rs_config(rs);
    }

    /* While there is something to do. */
    for (;;) {
        ocount = buflen(rs->obuf) - rs->ocount;
        count = bufend(rs->obuf) - rs->ohhead;
        if (count > ocount) count = ocount;
        if (count > tp->tty_outleft) count = tp->tty_outleft;
        if (count == 0 || tp->tty_inhibited) {
            if (try) return 0;
            break;
        }
        if (try) return 1;

        /* Copy from user space to the RS232 output buffer. */
        sys_vircopy(tp->tty_outproc, D, (vir_bytes) tp->tty_out_vir,
                    SELF, D, (vir_bytes) rs->ohhead, (phys_bytes) count);

        /* Perform output processing on the output buffer. */
        out_process(tp, rs->obuf, rs->ohhead, bufend(rs->obuf), &count, &ocount);
        if (count == 0) break;

        /* Assume echoing messed up by output. */
        tp->tty_reprint = TRUE;

        /* Bookkeeping. */
        lock(); /* protect interrupt sensitive rs->ocount */
        rs->ocount += ocount;
        rs_ostart(rs);
        unlock();
        if ((rs->ohhead += ocount) >= bufend(rs->obuf))
            rs->ohhead -= buflen(rs->obuf);
        tp->tty_out_vir += count;
        tp->tty_outcum += count;
        if ((tp->tty_outleft -= count) == 0) {
            /* Output is finished, reply to the writer. */
            tty_reply(tp->tty_outrepcode, tp->tty_outcaller,
                     tp->tty_outproc, tp->tty_outcum);
            tp->tty_outcum = 0;
        }
    }
    if (tp->tty_outleft > 0 && tp->tty_termios.c_ospeed == B0) {
        /* Oops, the line has hung up. */
        tty_reply(tp->tty_outrepcode, tp->tty_outcaller, tp->tty_outproc, EIO);
        tp->tty_outleft = tp->tty_outcum = 0;
    }

    return 1;
}

/*=====
 *                               rs_echo                               *
 *=====*/
PRIVATE void rs_echo(tp, c)
tty_t *tp; /* which TTY */
int c; /* character to echo */
{
    /* Echo one character. (Like rs_write, but only one character, optionally.) */

    rs232_t *rs = tp->tty_priv;
    int count, ocount;

```

```

ocount = buflen(rs->obuf) - rs->ocount;
if (ocount == 0) return;          /* output buffer full */
count = 1;
*rs->ohead = c;                  /* add one character */

out_process(tp, rs->obuf, rs->ohead, bufend(rs->obuf), &count, &ocount);
if (count == 0) return;

lock();
rs->ocount += ocount;
rs_ostart(rs);
unlock();
if ((rs->ohead += ocount) >= bufend(rs->obuf)) rs->ohead -= buflen(rs->obuf);
}

/*=====
*                               rs_ioctl                               *
*=====*/
PRIVATE int rs_ioctl(tp, dummy)
tty_t *tp;                        /* which TTY */
int dummy;
{
/* Reconfigure the line as soon as the output has drained. */
rs232_t *rs = tp->tty_priv;

rs->drain = TRUE;
return 0;      /* dummy */
}

/*=====
*                               rs_config                               *
*=====*/
PRIVATE void rs_config(rs)
rs232_t *rs;                      /* which line */
{
/* Set various line control parameters for RS232 I/O.
 * If DataBits == 5 and StopBits == 2, 8250 will generate 1.5 stop bits.
 * The 8250 can't handle split speed, so we use the input speed.
 */

tty_t *tp = rs->tty;
int divisor;
int line_controls;
static struct speed2divisor {
    speed_t speed;
    int divisor;
} s2d[] = {
#if (MACHINE == IBM_PC)
    { B50,          UART_FREQ / 50          },
#endif
    { B75,          UART_FREQ / 75          },
    { B110,         UART_FREQ / 110         },
    { B134,         UART_FREQ * 10 / 1345    },
    { B150,         UART_FREQ / 150         },
    { B200,         UART_FREQ / 200         },
    { B300,         UART_FREQ / 300         },
    { B600,         UART_FREQ / 600         },
    { B1200,        UART_FREQ / 1200        },
#if (MACHINE == IBM_PC)
    { B1800,        UART_FREQ / 1800        },
#endif
    { B2400,        UART_FREQ / 2400        },
    { B4800,        UART_FREQ / 4800        },
    { B9600,        UART_FREQ / 9600        },
    { B19200,       UART_FREQ / 19200       },
#if (MACHINE == IBM_PC)
    { B38400,       UART_FREQ / 38400       },
    { B57600,       UART_FREQ / 57600       },
    { B115200,      UART_FREQ / 115200L     },
#endif
    };
struct speed2divisor *s2dp;

/* RS232 needs to know the xoff character, and if CTS works. */

```

```

rs->oxoff = tp->tty_termios.c_cc[VSTOP];
rs->cts = (tp->tty_termios.c_cflag & CLOCAL) ? MS_CTS : 0;

/* Look up the 8250 rate divisor from the output speed. */
divisor = 0;
for (s2dp = s2d; s2dp < s2d + sizeof(s2d)/sizeof(s2d[0]); s2dp++) {
    if (s2dp->speed == tp->tty_termios.c_ospeed) divisor = s2dp->divisor;
}
if (divisor == 0) return; /* B0? */

#if (MACHINE == IBM_PC)
/* Compute line control flag bits. */
line_controls = 0;
if (tp->tty_termios.c_cflag & PARENB) {
    line_controls |= LC_PARITY;
    if (!(tp->tty_termios.c_cflag & PARODD)) line_controls |= LC_PAREVEN;
}
if (divisor >= (UART_FREQ / 110)) line_controls |= LC_2STOP_BITS;
line_controls |= (tp->tty_termios.c_cflag & CSIZE) >> 2;

/* Lock out interrupts while setting the speed. The receiver register is
 * going to be hidden by the div_low register, but the input interrupt
 * handler relies on reading it to clear the interrupt and avoid looping
 * forever.
 */
lock();

/* Select the baud rate divisor registers and change the rate. */
sys_outb(rs->line_ctl_port, LC_ADDRESS_DIVISOR);
sys_outb(rs->div_low_port, divisor);
sys_outb(rs->div_hi_port, divisor >> 8);

/* Change the line controls and reselect the usual registers. */
sys_outb(rs->line_ctl_port, line_controls);

rs->ostate = devready(rs) | ORAW | OSWREADY; /* reads modem_ctl_port */
if ((tp->tty_termios.c_lflag & IXON) && rs->oxoff != _POSIX_VDISABLE)
    rs->ostate &= ~ORAW;

unlock();

#else /* MACHINE == ATARI */

line_controls = U_Q16;
if (tp->tty_termios.c_cflag & PARENB) {
    line_controls |= U_PAR;
    if (!(tp->tty_termios.c_cflag & PARODD)) line_controls |= U_EVEN;
}
line_controls |= (divisor >= (UART_FREQ / 110)) ? U_ST2 : U_ST1;

switch (tp->tty_termios.c_cflag & CSIZE) { /* XXX - are U_Dn like CSn? */
    case CS5: line_controls |= U_D5; break;
    case CS6: line_controls |= U_D6; break;
    case CS7: line_controls |= U_D7; break;
    case CS8: line_controls |= U_D8; break;
}
lock();
MFP->mf_ucr = line_controls;
MFP->mf_tddr = divisor;
unlock();
#endif /* MACHINE == ATARI */
}

/*=====
 *                               rs_init                               *
 *=====*/
PUBLIC void rs_init(tp)
tty_t *tp; /* which TTY */
{
    unsigned long dummy;
/* Initialize RS232 for one line. */

    register rs232_t *rs;
    int line;

```

```

#if (MACHINE == IBM_PC)
    port_t this_8250;
    int irq;
    long v;
#endif

    /* Associate RS232 and TTY structures. */
    line = tp - &tty_table[NR_CONS];
    rs = tp->tty_priv = &rs_lines[line];
    rs->tty = tp;

    /* Set up input queue. */
    rs->ihead = rs->itail = rs->ibuf;

#if (MACHINE == IBM_PC)
    /* Precalculate port numbers for speed. Magic numbers in the code (once). */
    this_8250 = addr_8250[line];
    rs->xmit_port = this_8250 + 0;
    rs->recv_port = this_8250 + 0;
    rs->div_low_port = this_8250 + 0;
    rs->div_hi_port = this_8250 + 1;
    rs->int_enab_port = this_8250 + 1;
    rs->int_id_port = this_8250 + 2;
    rs->line_ctl_port = this_8250 + 3;
    rs->modem_ctl_port = this_8250 + 4;
    rs->line_status_port = this_8250 + 5;
    rs->modem_status_port = this_8250 + 6;
#endif

    /* Set up the hardware to a base state, in particular
     *   o turn off DTR (MC_DTR) to try to stop the external device.
     *   o be careful about the divisor latch. Some BIOS's leave it enabled
     *     here and that caused trouble (no interrupts) in version 1.5 by
     *     hiding the interrupt enable port in the next step, and worse trouble
     *     (continual interrupts) in an old version by hiding the receiver
     *     port in the first interrupt. Call rs_ioctl() early to avoid this.
     *   o disable interrupts at the chip level, to force an edge transition
     *     on the 8259 line when interrupts are next enabled and active.
     *     RS232 interrupts are guaranteed to be disabled now by the 8259
     *     mask, but there used to be trouble if the mask was set without
     *     handling a previous interrupt.
     */
    istop(rs); /* sets modem_ctl_port */
    rs_config(rs);
#if (MACHINE == IBM_PC)
    sys_outb(rs->int_enab_port, 0);
#endif

    /* Clear any harmful leftover interrupts. An output interrupt is harmless
     * and will occur when interrupts are enabled anyway. Set up the output
     * queue using the status from clearing the modem status interrupt.
     */
#if (MACHINE == IBM_PC)
    sys_inb(rs->line_status_port, &dummy);
    sys_inb(rs->recv_port, &dummy);
#endif
    rs->ostate = devready(rs) | ORAW | OSWREADY; /* reads modem_ctl_port */
    rs->ohd = rs->otail = rs->obuf;

#if (MACHINE == IBM_PC)
    /* Enable interrupts for both interrupt controller and device. */
    irq = (line & 1) == 0 ? RS232_IRQ : SECONDARY_IRQ;

    rs->irq = irq;
    rs->irq_hook_id = rs->irq; /* call back with irq line number */
    if (sys_irqsetpolicy(irq, IRQ_REENABLE, &rs->irq_hook_id) != OK) {
        printf("RS232: Couldn't obtain hook for irq %d\n", irq);
    } else {
        if (sys_irgenable(&rs->irq_hook_id) != OK) {
            printf("RS232: Couldn't enable irq %d (hooked)\n", irq);
        }
    }

    rs_irq_set |= (1 << irq);

```

```

    sys_outb(rs->int_enab_port, IE_LINE_STATUS_CHANGE | IE_MODEM_STATUS_CHANGE
                                     | IE_RECEIVER_READY | IE_TRANSMITTER_READY);
#else /* MACHINE == ATARI */
    /* Initialize the 68901 chip, then enable interrupts. */
    MFP->mf_scr = 0x00;
    MFP->mf_tcdcr |= T_Q004;
    MFP->mf_rsr = R_ENA;
    MFP->mf_tsr = T_ENA;
    MFP->mf_aer = (MFP->mf_aer | (IO_SCTS|IO_SDCD)) ^
                  (MFP->mf_gpip & (IO_SCTS|IO_SDCD));
    MFP->mf_ddr = (MFP->mf_ddr & ~ (IO_SCTS|IO_SDCD));
    MFP->mf_iera = (IA_RRDY|IA_RERR|IA_TRDY|IA_TERR);
    MFP->mf_imra = (IA_RRDY|IA_RERR|IA_TRDY|IA_TERR);
    MFP->mf_ierb = (IB_SCTS|IB_SDCD);
    MFP->mf_imrb = (IB_SCTS|IB_SDCD);
#endif /* MACHINE == ATARI */

    /* Fill in TTY function hooks. */
    tp->tty_devread = rs_read;
    tp->tty_devwrite = rs_write;
    tp->tty_echo = rs_echo;
    tp->tty_icancel = rs_icancel;
    tp->tty_ocancel = rs_ocancel;
    tp->tty_ioctl = rs_ioctl;
    tp->tty_break = rs_break;
    tp->tty_close = rs_close;

    /* Tell external device we are ready. */
    istart(rs);
}

/*=====
 *                               rs_interrupt                               *
 *=====*/
PUBLIC void rs_interrupt(m)
message *m; /* which TTY */
{
    unsigned long irq_set;
    int i;
    rs232_t *rs;

    irq_set= m->NOTIFY_ARG;
    for (i= 0, rs = rs_lines; i<NR_RS_LINES; i++, rs++)
    {
        if (irq_set & (1 << rs->irq))
            rs232_handler(rs);
    }
}

/*=====
 *                               rs_icancel                               *
 *=====*/
PRIVATE int rs_icancel(tp, dummy)
tty_t *tp; /* which TTY */
int dummy;
{
    /* Cancel waiting input. */
    rs232_t *rs = tp->tty_priv;

    lock();
    rs->icount = 0;
    rs->itail = rs->ihead;
    istart(rs);
    unlock();

    return 0; /* dummy */
}

/*=====
 *                               rs_ocancel                               *
 *=====*/
PRIVATE int rs_ocancel(tp, dummy)

```



```

tty_t *tp;                                /* which TTY */
int dummy;
{
    /* Cancel pending output. */
    rs232_t *rs = tp->tty_priv;

    lock();
    rs->ostate &= ~(ODONE | OQUEUED);
    rs->ocount = 0;
    rs->otail = rs->ohd;
    unlock();

    return 0;    /* dummy */
}

/*=====
*                               rs_read                               *
*=====*/
PRIVATE int rs_read(tp, try)
tty_t *tp;                                /* which tty */
int try;
{
    /* Process characters from the circular input buffer. */

    rs232_t *rs = tp->tty_priv;
    int icount, count, ostate;

    if (!(tp->tty_termios.c_cflag & CLOCAL)) {
        if (try) return 1;
        /* Send a SIGHUP if hangup detected. */
        lock();
        ostate = rs->ostate;
        rs->ostate &= ~ODEVHUP;    /* save ostate, clear DEVHUP */
        unlock();
        if (ostate & ODEVHUP) {
            sigchar(tp, SIGHUP);
            tp->tty_termios.c_ospeed = B0;    /* Disable further I/O. */
            return;
        }
    }

    if (try) {
        if (rs->icount > 0)
            return 1;
        return 0;
    }

    while ((count = rs->icount) > 0) {
        icount = bufend(rs->ibuf) - rs->itail;
        if (count > icount) count = icount;

        /* Perform input processing on (part of) the input buffer. */
        if ((count = in_process(tp, rs->itail, count)) == 0) break;

        lock();    /* protect interrupt sensitive variables */
        rs->icount -= count;
        if (!rs->idevready && rs->icount < RS_ILOWWATER) istart(rs);
        unlock();
        if ((rs->itail += count) == bufend(rs->ibuf)) rs->itail = rs->ibuf;
    }
}

/*=====
*                               rs_ostart                             *
*=====*/
PRIVATE void rs_ostart(rs)
rs232_t *rs;                                /* which rs line */
{
    /* Tell RS232 there is something waiting in the output buffer. */

    rs->ostate |= OQUEUED;
    if (txready(rs)) out_int(rs);
}

```

```

/*=====
 *                               rs_break                               *
 *=====*/
PRIVATE int rs_break(tp, dummy)
tty_t *tp;                               /* which tty */
int dummy;
{
/* Generate a break condition by setting the BREAK bit for 0.4 sec. */
rs232_t *rs = tp->tty_priv;
unsigned long line_controls;

sys_inb(rs->line_ctl_port, &line_controls);
sys_outb(rs->line_ctl_port, line_controls | LC_BREAK);
/* XXX */
/* milli_delay(400); */                               /* ouch */
printf("RS232 break\n");
sys_outb(rs->line_ctl_port, line_controls);
return 0;      /* dummy */
}

/*=====
 *                               rs_close                               *
 *=====*/
PRIVATE int rs_close(tp, dummy)
tty_t *tp;                               /* which tty */
int dummy;
{
/* The line is closed; optionally hang up. */
rs232_t *rs = tp->tty_priv;
int r;

if (tp->tty_termios.c_cflag & HUPCL) {
    sys_outb(rs->modem_ctl_port, MC_OUT2 | MC_RTS);
}
return 0;      /* dummy */
}

/* Low level (interrupt) routines. */

#if (MACHINE == IBM_PC)
/*=====
 *                               rs232_handler                          *
 *=====*/
PRIVATE void rs232_handler(rs)
struct rs232 *rs;
{
/* Interrupt handler for RS232. */

while (TRUE) {
    unsigned long v;
    /* Loop to pick up ALL pending interrupts for device.
     * This usually just wastes time unless the hardware has a buffer
     * (and then we have to worry about being stuck in the loop too long).
     * Unfortunately, some serial cards lock up without this.
     */
    sys_inb(rs->int_id_port, &v);
    switch (v) {
    case IS_RECEIVER_READY:
        in_int(rs);
        continue;
    case IS_TRANSMITTER_READY:
        out_int(rs);
        continue;
    case IS_MODEM_STATUS_CHANGE:
        modem_int(rs);
        continue;
    case IS_LINE_STATUS_CHANGE:
        line_int(rs);
        continue;
    }
    return;
}
}
#endif /* MACHINE == IBM_PC */

```

```

#if (MACHINE == ATARI)
/*=====
*
*                               siaint
*=====*/
PRIVATE void siaint(type)
int    type;          /* interrupt type */
{
/* siaint is the rs232 interrupt procedure for Atari ST's. For ST there are
* as much as 5 interrupt lines used for rs232. The trap type byte left on the
* stack by the assembler interrupt handler identifies the interrupt cause.
*/

    register unsigned char  code;
    register rs232_t *rs = &rs_lines[0];
    int s = lock();

    switch (type & 0x00FF)
    {
        case 0x00:          /* receive buffer full */
            in_int(rs);
            break;
        case 0x01:          /* receive error */
            line_int(rs);
            break;
        case 0x02:          /* transmit buffer empty */
            out_int(rs);
            break;
        case 0x03:          /* transmit error */
            code = MFP->mf_tsr;
            if (code & ~(T_ENA | T_UE | T_EMPTY))
            {
                printf("sia: transmit error: status=%x\r\n", code);
                /* MFP->mf_udr = lastchar; */ /* retry */
            }
            break;
        case 0x04:          /* modem lines change */
            modem_int(rs);
            break;
    }
    restore(s);
}
#endif /* MACHINE == ATARI */

/*=====
*
*                               in_int
*=====*/
PRIVATE void in_int(rs)
register rs232_t *rs;          /* line with input interrupt */
{
/* Read the data which just arrived.
* If it is the oxoff char, clear OSWREADY, else if OSWREADY was clear, set
* it and restart output (any char does this, not just xon).
* Put data in the buffer if room, otherwise discard it.
* Set a flag for the clock interrupt handler to eventually notify TTY.
*/

    unsigned long c;

#if (MACHINE == IBM_PC)
    sys_inb(rs->recv_port, &c);
#else /* MACHINE == ATARI */
    c = MFP->mf_udr;
#endif

    if (!(rs->ostate & ORAW)) {
        if (c == rs->oxoff) {
            rs->ostate &= ~OSWREADY;
        } else
        if (!(rs->ostate & OSWREADY)) {
            rs->ostate |= OSWREADY;
            if (txready(rs)) out_int(rs);
        }
    }
}

```

```

if (rs->icount == buflen(rs->ibuf))
{
    printf("in_int: discarding byte\n");
    return; /* input buffer full, discard */
}

if (++rs->icount == RS_IHIGHWATER && rs->idevready) istop(rs);
*rs->ihead = c;
if (++rs->ihead == bufend(rs->ibuf)) rs->ihead = rs->ibuf;
if (rs->icount == 1) {
    rs->tty->tty_events = 1;
    force_timeout();
}
else
    printf("in_int: icount = %d\n", rs->icount);
}

/*=====
*
*                               line_int
*=====*/
PRIVATE void line_int(rs)
register rs232_t *rs;          /* line with line status interrupt */
{
    /* Check for and record errors. */

    unsigned long s;
    #if (MACHINE == IBM_PC)
        sys_inb(rs->line_status_port, &s);
        rs->lstatus = s;
    #else /* MACHINE == ATARI */
        rs->lstatus = MFP->mf_rsr;
        MFP->mf_rsr &= R_ENA;
        rs->pad = MFP->mf_udr;          /* discard char in case of LS_OVERRUN_ERR */
    #endif /* MACHINE == ATARI */
    if (rs->lstatus & LS_FRAMING_ERR) ++rs->framing_errors;
    if (rs->lstatus & LS_OVERRUN_ERR) ++rs->overrun_errors;
    if (rs->lstatus & LS_PARITY_ERR) ++rs->parity_errors;
    if (rs->lstatus & LS_BREAK_INTERRUPT) ++rs->break_interrupts;
}

/*=====
*
*                               modem_int
*=====*/
PRIVATE void modem_int(rs)
register rs232_t *rs;          /* line with modem interrupt */
{
    /* Get possibly new device-ready status, and clear ODEVREADY if necessary.
    * If the device just became ready, restart output.
    */

    #if (MACHINE == ATARI)
        /* Set active edge interrupt so that next change causes a new interrupt */
        MFP->mf_aer = (MFP->mf_aer | (IO_SCTS|IO_SD CD)) ^
            (MFP->mf_gpip & (IO_SCTS|IO_SD CD));
    #endif

    if (devhup(rs)) {
        rs->ostate |= ODEVHUP;
        rs->tty->tty_events = 1;
        force_timeout();
    }

    if (!devready(rs))
        rs->ostate &= ~ODEVREADY;
    else if (!(rs->ostate & ODEVREADY)) {
        rs->ostate |= ODEVREADY;
        if (txready(rs)) out_int(rs);
    }
}

/*=====
*
*                               out_int
*=====*/

```

```
PRIVATE void out_int(rs)
register rs232_t *rs;          /* line with output interrupt */
{
/* If there is output to do and everything is ready, do it (local device is
 * known ready).
 * Notify TTY when the buffer goes empty.
 */

    if (rs->ostate >= (ODEVREADY | OQUEUED | OSWREADY)) {
        /* Bit test allows ORAW and requires the others. */
    #if (MACHINE == IBM_PC)
        sys_outb(rs->xmit_port, *rs->otail);
    #else /* MACHINE == ATARI */
        MFP->mf_udr = *rs->otail;
    #endif
        if (++rs->otail == bufend(rs->obuf)) rs->otail = rs->obuf;
        if (--rs->ocount == 0) {
            rs->ostate ^= (ODONE | OQUEUED); /* ODONE on, OQUEUED off */
            rs->tty->tty_events = 1;
            force_timeout();
        } else
        if (rs->ocount == RS_OLOWWATER) { /* running low? */
            rs->tty->tty_events = 1;
            force_timeout();
        }
    }
}
#endif /* NR_RS_LINES > 0 */
```

```

/* This file contains the terminal driver, both for the IBM console and regular
 * ASCII terminals. It handles only the device-independent part of a TTY, the
 * device dependent parts are in console.c, rs232.c, etc. This file contains
 * two main entry points, tty_task() and tty_wakeup(), and several minor entry
 * points for use by the device-dependent code.
 *
 * The device-independent part accepts "keyboard" input from the device-
 * dependent part, performs input processing (special key interpretation),
 * and sends the input to a process reading from the TTY. Output to a TTY
 * is sent to the device-dependent code for output processing and "screen"
 * display. Input processing is done by the device by calling 'in_process'
 * on the input characters, output processing may be done by the device itself
 * or by calling 'out_process'. The TTY takes care of input queuing, the
 * device does the output queuing. If a device receives an external signal,
 * like an interrupt, then it causes tty_wakeup() to be run by the CLOCK task
 * to, you guessed it, wake up the TTY to check if input or output can
 * continue.
 *
 * The valid messages and their parameters are:
 *
 * HARD_INT:      output has been completed or input has arrived
 * SYS_SIG:       e.g., MINIX wants to shutdown; run code to cleanly stop
 * DEV_READ:      a process wants to read from a terminal
 * DEV_WRITE:     a process wants to write on a terminal
 * DEV_IOCTL:     a process wants to change a terminal's parameters
 * DEV_OPEN:      a tty line has been opened
 * DEV_CLOSE:     a tty line has been closed
 * DEV_SELECT:    start select notification request
 * DEV_STATUS:    FS wants to know status for SELECT or REVIVE
 * CANCEL:        terminate a previous incomplete system call immediately
 *
 * m_type      TTY_LINE  IO_ENDPT  COUNT  TTY_SPEK  TTY_FLAGS  ADDRESS
 * -----
 * | HARD_INT   |          |          |        |          |          |
 * |-----+-----+-----+-----+-----+-----+
 * | SYS_SIG    | sig set |          |        |          |          |
 * |-----+-----+-----+-----+-----+-----+
 * | DEV_READ   | minor dev| proc nr | count |          | O_NONBLOCK| buf ptr |
 * |-----+-----+-----+-----+-----+-----+
 * | DEV_WRITE  | minor dev| proc nr | count |          |          | buf ptr |
 * |-----+-----+-----+-----+-----+-----+
 * | DEV_IOCTL  | minor dev| proc nr | func code|erase etc| flags |
 * |-----+-----+-----+-----+-----+-----+
 * | DEV_OPEN   | minor dev| proc nr | O_NOCTTY|          |          |
 * |-----+-----+-----+-----+-----+-----+
 * | DEV_CLOSE  | minor dev| proc nr |          |          |          |
 * |-----+-----+-----+-----+-----+-----+
 * | DEV_STATUS |          |          |          |          |          |
 * |-----+-----+-----+-----+-----+-----+
 * | CANCEL     | minor dev| proc nr |          |          |          |
 * |-----+-----+-----+-----+-----+-----+
 *
 * Changes:
 * Jan 20, 2004  moved TTY driver to user-space (Jorrit N. Herder)
 * Sep 20, 2004  local timer management/ sync alarms (Jorrit N. Herder)
 * Jul 13, 2004  support for function key observers (Jorrit N. Herder)
 */

```

```

#include "../drivers.h"
#include <termios.h>
#if ENABLE_SRCCOMPAT || ENABLE_BINCOMPAT
#include <sgtty.h>
#endif
#include <sys/ioc_tty.h>
#include <signal.h>
#include <minix/callnr.h>
#if (CHIP == INTEL)
#include <minix/keymap.h>
#endif
#include "tty.h"

#include <sys/time.h>
#include <sys/select.h>

```

```

extern int irq_hook_id;

unsigned long kbd_irq_set = 0;
unsigned long rs_irq_set = 0;

/* Address of a tty structure. */
#define tty_addr(line) (&tty_table[line])

/* Macros for magic tty types. */
#define isconsole(tp) ((tp) < tty_addr(NR_CONS))
#define ispty(tp) ((tp) >= tty_addr(NR_CONS+NR_RS_LINES))

/* Macros for magic tty structure pointers. */
#define FIRST_TTY tty_addr(0)
#define END_TTY tty_addr(sizeof(tty_table) / sizeof(tty_table[0]))

/* A device exists if at least its 'devread' function is defined. */
#define tty_active(tp) ((tp)->tty_devread != NULL)

/* RS232 lines or pseudo terminals can be completely configured out. */
#if NR_RS_LINES == 0
#define rs_init(tp) ((void) 0)
#endif
#if NR_PTYS == 0
#define pty_init(tp) ((void) 0)
#define do_pty(tp, mp) ((void) 0)
#endif

struct kmessages kmess;

FORWARD _PROTOTYPE( void tty_timed_out, (timer_t *tp) );
FORWARD _PROTOTYPE( void expire_timers, (void) );
FORWARD _PROTOTYPE( void settimer, (tty_t *tty_ptr, int enable) );
FORWARD _PROTOTYPE( void do_cancel, (tty_t *tp, message *m_ptr) );
FORWARD _PROTOTYPE( void do_ioctl, (tty_t *tp, message *m_ptr) );
FORWARD _PROTOTYPE( void do_open, (tty_t *tp, message *m_ptr) );
FORWARD _PROTOTYPE( void do_close, (tty_t *tp, message *m_ptr) );
FORWARD _PROTOTYPE( void do_read, (tty_t *tp, message *m_ptr) );
FORWARD _PROTOTYPE( void do_write, (tty_t *tp, message *m_ptr) );
FORWARD _PROTOTYPE( void do_select, (tty_t *tp, message *m_ptr) );
FORWARD _PROTOTYPE( void do_status, (message *m_ptr) );
FORWARD _PROTOTYPE( void in_transfer, (tty_t *tp) );
FORWARD _PROTOTYPE( int tty_echo, (tty_t *tp, int ch) );
FORWARD _PROTOTYPE( void rawecho, (tty_t *tp, int ch) );
FORWARD _PROTOTYPE( int back_over, (tty_t *tp) );
FORWARD _PROTOTYPE( void reprint, (tty_t *tp) );
FORWARD _PROTOTYPE( void dev_ioctl, (tty_t *tp) );
FORWARD _PROTOTYPE( void setattr, (tty_t *tp) );
FORWARD _PROTOTYPE( void tty_icancel, (tty_t *tp) );
FORWARD _PROTOTYPE( void tty_init, (void) );
#if ENABLE_SRCCOMPAT || ENABLE_BINCOMPAT
FORWARD _PROTOTYPE( int compat_getp, (tty_t *tp, struct sgtyb *sg) );
FORWARD _PROTOTYPE( int compat_getc, (tty_t *tp, struct tchars *sg) );
FORWARD _PROTOTYPE( int compat_setp, (tty_t *tp, struct sgtyb *sg) );
FORWARD _PROTOTYPE( int compat_setc, (tty_t *tp, struct tchars *sg) );
FORWARD _PROTOTYPE( int tspd2sgspd, (speed_t tspd) );
FORWARD _PROTOTYPE( speed_t sgspd2tspd, (int sgspd) );
#endif
#if ENABLE_BINCOMPAT
FORWARD _PROTOTYPE( void do_ioctl_compat, (tty_t *tp, message *m_ptr) );
#endif
#endif

/* Default attributes. */
PRIVATE struct termios termios_defaults = {
    TINPUT_DEF, TOUTPUT_DEF, TCTRL_DEF, TLOCAL_DEF, TSPEED_DEF, TSPEED_DEF,
    {
        TEOF_DEF, TEOL_DEF, TERASE_DEF, TINTR_DEF, TKILL_DEF, TMIN_DEF,
        TQUIT_DEF, TTIME_DEF, TSUSP_DEF, TSTART_DEF, TSTOP_DEF,
        TREPRINT_DEF, TLNEXT_DEF, TDISCARD_DEF,
    },
};
PRIVATE struct winsize winsize_defaults; /* = all zeroes */

/* Global variables for the TTY task (declared extern in tty.h). */

```

```

PUBLIC tty_t tty_table[NR_CONS+NR_RS_LINES+NR_PTYS];
PUBLIC int ccurrent;           /* currently active console */
PUBLIC timer_t *tty_timers;    /* queue of TTY timers */
PUBLIC clock_t tty_next_timeout; /* time that the next alarm is due */
PUBLIC struct machine machine; /* kernel environment variables */

/*=====
 *                               tty_task                               *
 *=====*/
PUBLIC void main(void)
{
/* Main routine of the terminal task. */

    message tty_mess;          /* buffer for all incoming messages */
    unsigned line;
    int r, s;
    register struct proc *rp;
    register tty_t *tp;

#ifdef DEBUG
    kputc('H');
    kputc('e');
    kputc('l');
    kputc('l');
    kputc('o');
    kputc(',');
    kputc(' ');
    printf("TTY\n");
#endif

    /* Get kernel environment (protected_mode, pc_at and ega are needed). */
    if (OK != (s=sys_getmachine(&machine))) {
        panic("TTY", "Couldn't obtain kernel environment.", s);
    }

    /* Initialize the TTY driver. */
    tty_init();

    /* Final one-time keyboard initialization. */
    kb_init_once();

    printf("\n");

    while (TRUE) {

        /* Check for and handle any events on any of the ttys. */
        for (tp = FIRST_TTY; tp < END_TTY; tp++) {
            if (tp->tty_events) handle_events(tp);
        }

        /* Get a request message. */
        r = receive(ANY, &tty_mess);
        if (r != 0)
            panic("TTY", "receive failed with %d", r);

        /* First handle all kernel notification types that the TTY supports.
         * - An alarm went off, expire all timers and handle the events.
         * - A hardware interrupt also is an invitation to check for events.
         * - A new kernel message is available for printing.
         * - Reset the console on system shutdown.
         * Then see if this message is different from a normal device driver
         * request and should be handled separately. These extra functions
         * do not operate on a device, in constrast to the driver requests.
         */
        switch (tty_mess.m_type) {
        case SYN_ALARM:          /* fall through */
            expire_timers();     /* run watchdogs of expired timers */
            continue;           /* continue to check for events */
        case DEV_PING:
            notify(tty_mess.m_source);
            continue;
        case HARD_INT: {        /* hardware interrupt notification */
            if (tty_mess.NOTIFY_ARG & kbd_irq_set)
                kbd_interrupt(&tty_mess); /* fetch chars from keyboard */
        }
        }
    }
}

```



```

#if NR_RS_LINES > 0
    if (tty_mess.NOTIFY_ARG & rs_irq_set)
        rs_interrupt(&tty_mess); /* serial I/O */
#endif

    expire_timers(); /* run watchdogs of expired timers */
    continue; /* continue to check for events */
}
case PROC_EVENT: {
    cons_stop(); /* switch to primary console */
    printf("TTY got PROC_EVENT, assuming SIGTERM\n");
#if DEAD_CODE
    if (irq_hook_id != -1) {
        sys_irqdisable(&irq_hook_id);
        sys_irqrmpolicy(KEYBOARD_IRQ, &irq_hook_id);
    }
#endif

    continue;
}
case SYS_SIG: { /* system signal */
    sigset_t sigset = (sigset_t) tty_mess.NOTIFY_ARG;
    if (sigismember(&sigset, SIGKMESS)) do_new_kmess(&tty_mess);
    continue;
}
case DIAGNOSTICS: /* a server wants to print some */
    do_diagnostics(&tty_mess);
    continue;
case GET_KMESS:
    do_get_kmess(&tty_mess);
    continue;
case FKEY_CONTROL: /* (un)register a fkey observer */
    do_fkey_ctl(&tty_mess);
    continue;
default: /* should be a driver request */
    ; /* do nothing; end switch */
}

/* Only device requests should get to this point. All requests,
 * except DEV_STATUS, have a minor device number. Check this
 * exception and get the minor device number otherwise.
 */
if (tty_mess.m_type == DEV_STATUS) {
    do_status(&tty_mess);
    continue;
}
line = tty_mess.TTY_LINE;
if (line == KBD_MINOR) {
    do_kbd(&tty_mess);
    continue;
} else if (line == KBD_AUX_MINOR) {
    do_kbdaux(&tty_mess);
    continue;
} else if (line == VIDEO_MINOR) {
    do_video(&tty_mess);
    continue;
} else if ((line - CONS_MINOR) < NR_CONS) {
    tp = tty_addr(line - CONS_MINOR);
} else if (line == LOG_MINOR) {
    tp = tty_addr(0);
} else if ((line - RS232_MINOR) < NR_RS_LINES) {
    tp = tty_addr(line - RS232_MINOR + NR_CONS);
} else if ((line - TTYPX_MINOR) < NR_PTYS) {
    tp = tty_addr(line - TTYPX_MINOR + NR_CONS + NR_RS_LINES);
} else if ((line - PTYPX_MINOR) < NR_PTYS) {
    tp = tty_addr(line - PTYPX_MINOR + NR_CONS + NR_RS_LINES);
    if (tty_mess.m_type != DEV_IOCTL) {
        do_pty(tp, &tty_mess);
        continue;
    }
} else {
    tp = NULL;
}

/* If the device doesn't exist or is not configured return ENXIO. */
if (tp == NULL || ! tty_active(tp)) {

```

```

        printf("Warning, TTY got illegal request %d from %d\n",
               tty_mess.m_type, tty_mess.m_source);
        if (tty_mess.m_source != LOG_PROC_NR)
        {
            tty_reply(TASK_REPLY, tty_mess.m_source,
                     tty_mess.IO_ENDPT, ENXIO);
        }
        continue;
    }

    /* Execute the requested device driver function. */
    switch (tty_mess.m_type) {
        case DEV_READ:      do_read(tp, &tty_mess);          break;
        case DEV_WRITE:     do_write(tp, &tty_mess);         break;
        case DEV_IOCTL:     do_ioctl(tp, &tty_mess);         break;
        case DEV_OPEN:      do_open(tp, &tty_mess);          break;
        case DEV_CLOSE:     do_close(tp, &tty_mess);         break;
        case DEV_SELECT:    do_select(tp, &tty_mess);        break;
        case CANCEL:        do_cancel(tp, &tty_mess);        break;
        default:
            printf("Warning, TTY got unexpected request %d from %d\n",
                   tty_mess.m_type, tty_mess.m_source);
            tty_reply(TASK_REPLY, tty_mess.m_source,
                     tty_mess.IO_ENDPT, EINVAL);
    }
}

/*=====
 *                               do_status                               *
 *=====*/
PRIVATE void do_status(m_ptr)
message *m_ptr;
{
    register struct tty *tp;
    int event_found;
    int status;
    int ops;

    /* Check for select or revive events on any of the ttys. If we found an,
     * event return a single status message for it. The FS will make another
     * call to see if there is more.
     */
    event_found = 0;
    for (tp = FIRST_TTY; tp < END_TTY; tp++) {
        if ((ops = select_try(tp, tp->tty_select_ops)) &&
            tp->tty_select_proc == m_ptr->m_source) {

            /* I/O for a selected minor device is ready. */
            m_ptr->m_type = DEV_IO_READY;
            m_ptr->DEV_MINOR = tp->tty_minor;
            m_ptr->DEV_SEL_OPS = ops;

            tp->tty_select_ops &= ~ops;    /* unmark select event */
            event_found = 1;
            break;
        }
        else if (tp->tty_inrevived && tp->tty_incaller == m_ptr->m_source) {

            /* Suspended request finished. Send a REVIVE. */
            m_ptr->m_type = DEV_REVIVE;
            m_ptr->REP_ENDPT = tp->tty_inproc;
            m_ptr->REP_STATUS = tp->tty_incum;

            tp->tty_inleft = tp->tty_incum = 0;
            tp->tty_inrevived = 0;        /* unmark revive event */
            event_found = 1;
            break;
        }
        else if (tp->tty_outrevived && tp->tty_outcaller == m_ptr->m_source) {

            /* Suspended request finished. Send a REVIVE. */
            m_ptr->m_type = DEV_REVIVE;
            m_ptr->REP_ENDPT = tp->tty_outproc;

```

```

        m_ptr->REP_STATUS = tp->tty_outcum;

        tp->tty_outcum = 0;
        tp->tty_outrevived = 0;          /* unmark revive event */
        event_found = 1;
        break;
    }
}

#if NR_PTYS > 0
    if (!event_found)
        event_found = pty_status(m_ptr);
#endif
    if (!event_found)
        event_found = kbd_status(m_ptr);

    if (!event_found) {
        /* No events of interest were found. Return an empty message. */
        m_ptr->m_type = DEV_NO_STATUS;
    }

    /* Almost done. Send back the reply message to the caller. */
    if ((status = send(m_ptr->m_source, m_ptr)) != OK) {
        panic("TTY", "send in do_status failed, status\n", status);
    }
}

/*=====
*                               do_read                               *
*=====*/
PRIVATE void do_read(tp, m_ptr)
register tty_t *tp;          /* pointer to tty struct */
register message *m_ptr;    /* pointer to message sent to the task */
{
    /* A process wants to read from a terminal. */
    int r, status;
    phys_bytes phys_addr;
    int more_verbose = (tp == tty_addr(NR_CONS));

    /* Check if there is already a process hanging in a read, check if the
     * parameters are correct, do I/O.
     */
    if (tp->tty_inleft > 0) {
        if (more_verbose) printf("do_read: EIO\n");
        r = EIO;
    } else
    if (m_ptr->COUNT <= 0) {
        if (more_verbose) printf("do_read: EINVAL\n");
        r = EINVAL;
    } else
    if (sys_umap(m_ptr->IO_ENDPT, D, (vir_bytes) m_ptr->ADDRESS, m_ptr->COUNT,
        &phys_addr) != OK) {
        if (more_verbose) printf("do_read: EFAULT\n");
        r = EFAULT;
    } else {
        /* Copy information from the message to the tty struct. */
        tp->tty_inrepcode = TASK_REPLY;
        tp->tty_incaller = m_ptr->m_source;
        tp->tty_inproc = m_ptr->IO_ENDPT;
        tp->tty_invir = (vir_bytes) m_ptr->ADDRESS;
        tp->tty_inleft = m_ptr->COUNT;

        if (!(tp->tty_termios.c_lflag & ICANON)
            && tp->tty_termios.c_cc[VTIME] > 0) {
            if (tp->tty_termios.c_cc[VMIN] == 0) {
                /* MIN & TIME specify a read timer that finishes the
                 * read in TIME/10 seconds if no bytes are available.
                 */
                settimer(tp, TRUE);
                tp->tty_min = 1;
            } else {
                /* MIN & TIME specify an inter-byte timer that may
                 * have to be cancelled if there are no bytes yet.
                 */

```

```

        if (tp->tty_eotct == 0) {
            settimer(tp, FALSE);
            tp->tty_min = tp->tty_termios.c_cc[VMIN];
        }
    }

    /* Anything waiting in the input buffer? Clear it out... */
    in_transfer(tp);
    /* ...then go back for more. */
    handle_events(tp);
    if (tp->tty_inleft == 0) {
        if (tp->tty_select_ops)
            select_retry(tp);
        return; /* already done */
    }

    /* There were no bytes in the input queue available, so either suspend
     * the caller or break off the read if nonblocking.
     */
    if (m_ptr->TTY_FLAGS & O_NONBLOCK) {
        r = EAGAIN; /* cancel the read */
        tp->tty_inleft = tp->tty_incum = 0;
    } else {
        r = SUSPEND; /* suspend the caller */
        tp->tty_inrepcode = REVIVE;
    }
}

if (more_verbose) printf("do_read: replying %d\n", r);
tty_reply(TASK_REPLY, m_ptr->m_source, m_ptr->IO_ENDPT, r);
if (tp->tty_select_ops)
    select_retry(tp);
}

/*=====
 *                               do_write                               *
 *=====*/
PRIVATE void do_write(tp, m_ptr)
register tty_t *tp;
register message *m_ptr; /* pointer to message sent to the task */
{
    /* A process wants to write on a terminal. */
    int r;
    phys_bytes phys_addr;

    /* Check if there is already a process hanging in a write, check if the
     * parameters are correct, do I/O.
     */
    if (tp->tty_outleft > 0) {
        r = EIO;
    } else
    if (m_ptr->COUNT <= 0) {
        r = EINVAL;
    } else
    if (sys_umap(m_ptr->IO_ENDPT, D, (vir_bytes) m_ptr->ADDRESS, m_ptr->COUNT,
                &phys_addr) != OK) {
        r = EFAULT;
    } else {
        /* Copy message parameters to the tty structure. */
        tp->tty_outrepcode = TASK_REPLY;
        tp->tty_outcaller = m_ptr->m_source;
        tp->tty_outproc = m_ptr->IO_ENDPT;
        tp->tty_out_vir = (vir_bytes) m_ptr->ADDRESS;
        tp->tty_outleft = m_ptr->COUNT;

        /* Try to write. */
        handle_events(tp);
        if (tp->tty_outleft == 0)
            return; /* already done */

        /* None or not all the bytes could be written, so either suspend the
         * caller or break off the write if nonblocking.
         */
        if (m_ptr->TTY_FLAGS & O_NONBLOCK) {
            /* cancel the write */

```

```

        r = tp->tty_outcum > 0 ? tp->tty_outcum : EAGAIN;
        tp->tty_outleft = tp->tty_outcum = 0;
    } else {
        r = SUSPEND;
        tp->tty_outrepcode = REVIVE;
    }
}
tty_reply(TASK_REPLY, m_ptr->m_source, m_ptr->IO_ENDPT, r);
}

/*=====
 *                               do_ioctl                               *
 *=====*/
PRIVATE void do_ioctl(tp, m_ptr)
register tty_t *tp;
message *m_ptr;
/* pointer to message sent to task */
{
/* Perform an IOCTL on this terminal. Posix termios calls are handled
 * by the IOCTL system call
 */

    int r;
    union {
        int i;
#ifdef ENABLE_SRCCOMPAT
        struct sgtyb sg;
        struct tchars tc;
#endif
    } param;
    size_t size;

/* Size of the ioctl parameter. */
    switch (m_ptr->TTY_REQUEST) {
        case TCGETS: /* Posix tcgetattr function */
        case TCSETS: /* Posix tcsetattr function, TCSANOW option */
        case TCSETSW: /* Posix tcsetattr function, TCSADRAIN option */
        case TCSETSF: /* Posix tcsetattr function, TCSAFLUSH option */
            size = sizeof(struct termios);
            break;

        case TCSBRK: /* Posix tcsendbreak function */
        case TCFLW: /* Posix tcflow function */
        case TCFLSH: /* Posix tcflush function */
        case TIOCGPGRP: /* Posix tcgetpgrp function */
        case TIOCSGRP: /* Posix tcsetpgrp function */
            size = sizeof(int);
            break;

        case TIOCGWINSZ: /* get window size (not Posix) */
        case TIOCSWINSZ: /* set window size (not Posix) */
            size = sizeof(struct winsize);
            break;

#ifdef ENABLE_SRCCOMPAT
        case TIOCGETP: /* BSD-style get terminal properties */
        case TIOCSETP: /* BSD-style set terminal properties */
            size = sizeof(struct sgtyb);
            break;

        case TIOCGETC: /* BSD-style get terminal special characters */
        case TIOCSETC: /* BSD-style set terminal special characters */
            size = sizeof(struct tchars);
            break;
#endif
    }

#ifdef (MACHINE == IBM_PC)
    case KIOCSMAP: /* load keymap (Minix extension) */
        size = sizeof(keymap_t);
        break;

    case TIOCSFON: /* load font (Minix extension) */
        size = sizeof(u8_t [8192]);
        break;
#endif
}

```

```

    case TCDRAIN:          /* Posix tcdrain function -- no parameter */
    default:                size = 0;
}

r = OK;
switch (m_ptr->TTY_REQUEST) {
    case TCGETS:
        /* Get the termios attributes. */
        r = sys_vircopy(SELF, D, (vir_bytes) &tp->tty_termios,
                        m_ptr->IO_ENDPT, D, (vir_bytes) m_ptr->ADDRESS,
                        (vir_bytes) size);
        break;

    case TCSETSW:
    case TCSETSF:
    case TCDRAIN:
        if (tp->tty_outleft > 0) {
            /* Wait for all ongoing output processing to finish. */
            tp->tty_iocaller = m_ptr->m_source;
            tp->tty_ioproc = m_ptr->IO_ENDPT;
            tp->tty_ioreq = m_ptr->REQUEST;
            tp->tty_iovir = (vir_bytes) m_ptr->ADDRESS;
            r = SUSPEND;
            break;
        }
        if (m_ptr->TTY_REQUEST == TCDRAIN) break;
        if (m_ptr->TTY_REQUEST == TCSETSF) tty_icancel(tp);
        /*FALL THROUGH*/
    case TCSETS:
        /* Set the termios attributes. */
        r = sys_vircopy( m_ptr->IO_ENDPT, D, (vir_bytes) m_ptr->ADDRESS,
                        SELF, D, (vir_bytes) &tp->tty_termios, (vir_bytes) size);
        if (r != OK) break;
        setattr(tp);
        break;

    case TCFLSH:
        r = sys_vircopy( m_ptr->IO_ENDPT, D, (vir_bytes) m_ptr->ADDRESS,
                        SELF, D, (vir_bytes) &param.i, (vir_bytes) size);
        if (r != OK) break;
        switch (param.i) {
            case TCIFLUSH:          tty_icancel(tp);                      break;
            case TCOFLUSH:          (*tp->tty_ocancel)(tp, 0);            break;
            case TCIOFLUSH:        tty_icancel(tp); (*tp->tty_ocancel)(tp, 0); break;
            default:                r = EINVAL;
        }
        break;

    case TCFLOW:
        r = sys_vircopy( m_ptr->IO_ENDPT, D, (vir_bytes) m_ptr->ADDRESS,
                        SELF, D, (vir_bytes) &param.i, (vir_bytes) size);
        if (r != OK) break;
        switch (param.i) {
            case TCOOFF:
            case TCOON:
                tp->tty_inhibited = (param.i == TCOOFF);
                tp->tty_events = 1;
                break;
            case TCIOFF:
                (*tp->tty_echo)(tp, tp->tty_termios.c_cc[VSTOP]);
                break;
            case TCION:
                (*tp->tty_echo)(tp, tp->tty_termios.c_cc[VSTART]);
                break;
            default:
                r = EINVAL;
        }
        break;

    case TCSBRK:
        if (tp->tty_break != NULL) (*tp->tty_break)(tp, 0);
        break;

    case TIOCGWINSZ:

```

```

        r = sys_vircopy(SELF, D, (vir_bytes) &tp->tty_winsize,
                        m_ptr->IO_ENDPT, D, (vir_bytes) m_ptr->ADDRESS,
                        (vir_bytes) size);
        break;

    case TIOCSWINSZ:
        r = sys_vircopy( m_ptr->IO_ENDPT, D, (vir_bytes) m_ptr->ADDRESS,
                        SELF, D, (vir_bytes) &tp->tty_winsize, (vir_bytes) size);
        sigchar(tp, SIGWINCH);
        break;

#if ENABLE_SRCCOMPAT
    case TIOCGGETP:
        compat_getp(tp, &param.sg);
        r = sys_vircopy(SELF, D, (vir_bytes) &param.sg,
                        m_ptr->IO_ENDPT, D, (vir_bytes) m_ptr->ADDRESS,
                        (vir_bytes) size);
        break;

    case TIOCSETP:
        r = sys_vircopy( m_ptr->IO_ENDPT, D, (vir_bytes) m_ptr->ADDRESS,
                        SELF, D, (vir_bytes) &param.sg, (vir_bytes) size);
        if (r != OK) break;
        compat_setp(tp, &param.sg);
        break;

    case TIOCGETC:
        compat_getc(tp, &param.tc);
        r = sys_vircopy(SELF, D, (vir_bytes) &param.tc,
                        m_ptr->IO_ENDPT, D, (vir_bytes) m_ptr->ADDRESS,
                        (vir_bytes) size);
        break;

    case TIOCSETC:
        r = sys_vircopy( m_ptr->IO_ENDPT, D, (vir_bytes) m_ptr->ADDRESS,
                        SELF, D, (vir_bytes) &param.tc, (vir_bytes) size);
        if (r != OK) break;
        compat_setc(tp, &param.tc);
        break;
#endif

#if (MACHINE == IBM_PC)
    case KIOCSMAP:
        /* Load a new keymap (only /dev/console). */
        if (isconsole(tp)) r = kbd_loadmap(m_ptr);
        break;

    case TIOCSFON:
        /* Load a font into an EGA or VGA card (hs@hck.hr) */
        if (isconsole(tp)) r = con_loadfont(m_ptr);
        break;
#endif

#if (MACHINE == ATARI)
    case VDU_LOADFONT:
        r = vdu_loadfont(m_ptr);
        break;
#endif

/* These Posix functions are allowed to fail if _POSIX_JOB_CONTROL is
 * not defined.
 */
    case TIOCGPGRP:
    case TIOCSPGRP:
    default:
#if ENABLE_BINCOMPAT
        do_ioctl_compat(tp, m_ptr);
        return;
#else
        r = ENOTTY;
#endif
}

/* Send the reply. */

```

```

tty_reply(TASK_REPLY, m_ptr->m_source, m_ptr->IO_ENDPT, r);
}

/*=====
*                                     do_open                                     *
*=====*/
PRIVATE void do_open(tp, m_ptr)
register tty_t *tp;
message *m_ptr;          /* pointer to message sent to task */
{
/* A tty line has been opened. Make it the callers controlling tty if
* O_NOCTTY is *not* set and it is not the log device. 1 is returned if
* the tty is made the controlling tty, otherwise OK or an error code.
*/
int r = OK;

if (m_ptr->TTY_LINE == LOG_MINOR) {
/* The log device is a write-only diagnostics device. */
if (m_ptr->COUNT & R_BIT) r = EACCES;
} else {
if (!(m_ptr->COUNT & O_NOCTTY)) {
tp->tty_pgrp = m_ptr->IO_ENDPT;
r = 1;
}
tp->tty_openct++;
}
tty_reply(TASK_REPLY, m_ptr->m_source, m_ptr->IO_ENDPT, r);
}

/*=====
*                                     do_close                                    *
*=====*/
PRIVATE void do_close(tp, m_ptr)
register tty_t *tp;
message *m_ptr;          /* pointer to message sent to task */
{
/* A tty line has been closed. Clean up the line if it is the last close. */

if (m_ptr->TTY_LINE != LOG_MINOR && --tp->tty_openct == 0) {
tp->tty_pgrp = 0;
tty_icancel(tp);
(*tp->tty_ocancel)(tp, 0);
(*tp->tty_close)(tp, 0);
tp->tty_termios = termios_defaults;
tp->tty_winsize = winsize_defaults;
setattr(tp);
}
tty_reply(TASK_REPLY, m_ptr->m_source, m_ptr->IO_ENDPT, OK);
}

/*=====
*                                     do_cancel                                    *
*=====*/
PRIVATE void do_cancel(tp, m_ptr)
register tty_t *tp;
message *m_ptr;          /* pointer to message sent to task */
{
/* A signal has been sent to a process that is hanging trying to read or write.
* The pending read or write must be finished off immediately.
*/

int proc_nr;
int mode;

/* Check the parameters carefully, to avoid cancelling twice. */
proc_nr = m_ptr->IO_ENDPT;
mode = m_ptr->COUNT;
if ((mode & R_BIT) && tp->tty_inleft != 0 && proc_nr == tp->tty_inproc) {
/* Process was reading when killed. Clean up input. */
tty_icancel(tp);
tp->tty_inleft = tp->tty_incum = 0;
}
if ((mode & W_BIT) && tp->tty_outleft != 0 && proc_nr == tp->tty_outproc) {
/* Process was writing when killed. Clean up output. */

```



```

        (*tp->tty_ocancel)(tp, 0);
        tp->tty_outleft = tp->tty_outcum = 0;
    }
    if (tp->tty_ioreq != 0 && proc_nr == tp->tty_ioproc) {
        /* Process was waiting for output to drain. */
        tp->tty_ioreq = 0;
    }
    tp->tty_events = 1;
    tty_reply(TASK_REPLY, m_ptr->m_source, proc_nr, EINTR);
}

PUBLIC int select_try(struct tty *tp, int ops)
{
    int ready_ops = 0;

    /* Special case. If line is hung up, no operations will block.
     * (and it can be seen as an exceptional condition.)
     */
    if (tp->tty_termios.c_ospeed == B0) {
        ready_ops |= ops;
    }

    if (ops & SEL_RD) {
        /* will i/o not block on read? */
        if (tp->tty_inleft > 0) {
            ready_ops |= SEL_RD;    /* EIO - no blocking */
        } else if (tp->tty_incount > 0) {
            /* Is a regular read possible? tty_incount
             * says there is data. But a read will only succeed
             * in canonical mode if a newline has been seen.
             */
            if (!(tp->tty_termios.c_lflag & ICANON) ||
                tp->tty_eotct > 0) {
                ready_ops |= SEL_RD;
            }
        }
    }

    if (ops & SEL_WR) {
        if (tp->tty_outleft > 0) ready_ops |= SEL_WR;
        else if ((*tp->tty_devwrite)(tp, 1)) ready_ops |= SEL_WR;
    }
    return ready_ops;
}

PUBLIC int select_retry(struct tty *tp)
{
    if (tp->tty_select_ops && select_try(tp, tp->tty_select_ops))
        notify(tp->tty_select_proc);
    return OK;
}

/*=====
 *                               handle_events                               *
 *=====*/
PUBLIC void handle_events(tp)
tty_t *tp;                /* TTY to check for events. */
{
    /* Handle any events pending on a TTY. These events are usually device
     * interrupts.
     *
     * Two kinds of events are prominent:
     *   - a character has been received from the console or an RS232 line.
     *   - an RS232 line has completed a write request (on behalf of a user).
     * The interrupt handler may delay the interrupt message at its discretion
     * to avoid swamping the TTY task. Messages may be overwritten when the
     * lines are fast or when there are races between different lines, input
     * and output, because MINIX only provides single buffering for interrupt
     * messages (in proc.c). This is handled by explicitly checking each line
     * for fresh input and completed output on each interrupt.
     */
    char *buf;
    unsigned count;
    int status;

```

```

do {
    tp->tty_events = 0;

    /* Read input and perform input processing. */
    (*tp->tty_devread)(tp, 0);

    /* Perform output processing and write output. */
    (*tp->tty_devwrite)(tp, 0);

    /* Ioctl waiting for some event? */
    if (tp->tty_ioreq != 0) dev_ioctl(tp);
} while (tp->tty_events);

/* Transfer characters from the input queue to a waiting process. */
in_transfer(tp);

/* Reply if enough bytes are available. */
if (tp->tty_incum >= tp->tty_min && tp->tty_inleft > 0) {
    if (tp->tty_inrepcode == REVIVE) {
        notify(tp->tty_incaller);
        tp->tty_inrevived = 1;
    } else {
        tty_reply(tp->tty_inrepcode, tp->tty_incaller,
            tp->tty_inproc, tp->tty_incum);
        tp->tty_inleft = tp->tty_incum = 0;
    }
}
if (tp->tty_select_ops)
{
    select_retry(tp);
}
#if NR_PTYS > 0
    if (ispty(tp))
        select_retry_pty(tp);
#endif
}

/*=====
*                                     in_transfer
*=====*/
PRIVATE void in_transfer(tp)
register tty_t *tp;          /* pointer to terminal to read from */
{
    /* Transfer bytes from the input queue to a process reading from a terminal. */

    int ch;
    int count;
    char buf[64], *bp;

    /* Force read to succeed if the line is hung up, looks like EOF to reader. */
    if (tp->tty_termios.c_ospeed == B0) tp->tty_min = 0;

    /* Anything to do? */
    if (tp->tty_inleft == 0 || tp->tty_eotct < tp->tty_min) return;

    bp = buf;
    while (tp->tty_inleft > 0 && tp->tty_eotct > 0) {
        ch = *tp->tty_intail;

        if (!(ch & IN_EOF)) {
            /* One character to be delivered to the user. */
            *bp = ch & IN_CHAR;
            tp->tty_inleft--;
            if (++bp == bufend(buf)) {
                /* Temp buffer full, copy to user space. */
                sys_vircopy(SELFF, D, (vir_bytes) buf,
                    tp->tty_inproc, D, tp->tty_in_vir,
                    (vir_bytes) buflen(buf));
                tp->tty_in_vir += buflen(buf);
                tp->tty_incum += buflen(buf);
                bp = buf;
            }
        }
    }
}

```

```

    /* Remove the character from the input queue. */
    if (++tp->tty_intail == bufend(tp->tty_inbuf))
        tp->tty_intail = tp->tty_inbuf;
    tp->tty_incount--;
    if (ch & IN_EOT) {
        tp->tty_eotct--;
        /* Don't read past a line break in canonical mode. */
        if (tp->tty_termios.c_lflag & ICANON) tp->tty_inleft = 0;
    }
}

if (bp > buf) {
    /* Leftover characters in the buffer. */
    count = bp - buf;
    sys_vircopy(SELFS, D, (vir_bytes) buf,
        tp->tty_inproc, D, tp->tty_in_vir, (vir_bytes) count);
    tp->tty_in_vir += count;
    tp->tty_incum += count;
}

/* Usually reply to the reader, possibly even if incum == 0 (EOF). */
if (tp->tty_inleft == 0) {
    if (tp->tty_inrepcode == REVIVE) {
        notify(tp->tty_incaller);
        tp->tty_inrevived = 1;
    } else {
        tty_reply(tp->tty_inrepcode, tp->tty_incaller,
            tp->tty_inproc, tp->tty_incum);
        tp->tty_inleft = tp->tty_incum = 0;
    }
}
}

/*=====
 *                                     in_process                                     *
 *=====*/
PUBLIC int in_process(tp, buf, count)
register tty_t *tp;          /* terminal on which character has arrived */
char *buf;                  /* buffer with input characters */
int count;                  /* number of input characters */
{
    /* Characters have just been typed in. Process, save, and echo them. Return
     * the number of characters processed.
     */

    int ch, sig, ct;
    int timeset = FALSE;
    static unsigned char csize_mask[] = { 0x1F, 0x3F, 0x7F, 0xFF };

    for (ct = 0; ct < count; ct++) {
        /* Take one character. */
        ch = *buf++ & BYTE;

        /* Strip to seven bits? */
        if (tp->tty_termios.c_iflag & ISTRIP) ch &= 0x7F;

        /* Input extensions? */
        if (tp->tty_termios.c_lflag & IEXTEN) {

            /* Previous character was a character escape? */
            if (tp->tty_escaped) {
                tp->tty_escaped = NOT_ESCAPED;
                ch |= IN_ESC; /* protect character */
            }

            /* LNEXT (^V) to escape the next character? */
            if (ch == tp->tty_termios.c_cc[VLNEXT]) {
                tp->tty_escaped = ESCAPED;
                rawecho(tp, '^');
                rawecho(tp, '\b');
                continue; /* do not store the escape */
            }

```

```
/* REPRINT (^R) to reprint echoed characters? */
if (ch == tp->tty_termios.c_cc[VREPRINT]) {
    reprint(tp);
    continue;
}

/* _POSIX_VDISABLE is a normal character value, so better escape it. */
if (ch == _POSIX_VDISABLE) ch |= IN_ESC;

/* Map CR to LF, ignore CR, or map LF to CR. */
if (ch == '\r') {
    if (tp->tty_termios.c_iflag & IGNCR) continue;
    if (tp->tty_termios.c_iflag & ICRNL) ch = '\n';
} else
if (ch == '\n') {
    if (tp->tty_termios.c_iflag & INLCR) ch = '\r';
}

/* Canonical mode? */
if (tp->tty_termios.c_lflag & ICANON) {

    /* Erase processing (rub out of last character). */
    if (ch == tp->tty_termios.c_cc[VERASE]) {
        (void) back_over(tp);
        if (!(tp->tty_termios.c_lflag & ECHOE)) {
            (void) tty_echo(tp, ch);
        }
        continue;
    }

    /* Kill processing (remove current line). */
    if (ch == tp->tty_termios.c_cc[VKILL]) {
        while (back_over(tp)) {}
        if (!(tp->tty_termios.c_lflag & ECHOE)) {
            (void) tty_echo(tp, ch);
            if (tp->tty_termios.c_lflag & ECHOK)
                rawecho(tp, '\n');
        }
        continue;
    }

    /* EOF (^D) means end-of-file, an invisible "line break". */
    if (ch == tp->tty_termios.c_cc[VEOF]) ch |= IN_EOT | IN_EOF;

    /* The line may be returned to the user after an LF. */
    if (ch == '\n') ch |= IN_EOT;

    /* Same thing with EOL, whatever it may be. */
    if (ch == tp->tty_termios.c_cc[VEOL]) ch |= IN_EOT;
}

/* Start/stop input control? */
if (tp->tty_termios.c_iflag & IXON) {

    /* Output stops on STOP (^S). */
    if (ch == tp->tty_termios.c_cc[VSTOP]) {
        tp->tty_inhibited = STOPPED;
        tp->tty_events = 1;
        continue;
    }

    /* Output restarts on START (^Q) or any character if IXANY. */
    if (tp->tty_inhibited) {
        if (ch == tp->tty_termios.c_cc[VSTART]
            || (tp->tty_termios.c_iflag & IXANY)) {
            tp->tty_inhibited = RUNNING;
            tp->tty_events = 1;
            if (ch == tp->tty_termios.c_cc[VSTART])
                continue;
        }
    }
}
```

```

    if (tp->tty_termios.c_lflag & ISIG) {
        /* Check for INTR (^?) and QUIT (^\\) characters. */
        if (ch == tp->tty_termios.c_cc[VINTR]
            || ch == tp->tty_termios.c_cc[VQUIT]) {
            sig = SIGINT;
            if (ch == tp->tty_termios.c_cc[VQUIT]) sig = SIGQUIT;
            sigchar(tp, sig);
            (void) tty_echo(tp, ch);
            continue;
        }
    }

    /* Is there space in the input buffer? */
    if (tp->tty_incount == buflen(tp->tty_inbuf)) {
        /* No space; discard in canonical mode, keep in raw mode. */
        if (tp->tty_termios.c_lflag & ICANON) continue;
        break;
    }

    if (!(tp->tty_termios.c_lflag & ICANON)) {
        /* In raw mode all characters are "line breaks". */
        ch |= IN_EOT;

        /* Start an inter-byte timer? */
        if (!timeset && tp->tty_termios.c_cc[VMIN] > 0
            && tp->tty_termios.c_cc[VTIME] > 0) {
            settimer(tp, TRUE);
            timeset = TRUE;
        }
    }

    /* Perform the intricate function of echoing. */
    if (tp->tty_termios.c_lflag & (ECHO|ECHONL)) ch = tty_echo(tp, ch);

    /* Save the character in the input queue. */
    *tp->tty_inhead++ = ch;
    if (tp->tty_inhead == bufend(tp->tty_inbuf))
        tp->tty_inhead = tp->tty_inbuf;
    tp->tty_incount++;
    if (ch & IN_EOT) tp->tty_eotct++;

    /* Try to finish input if the queue threatens to overflow. */
    if (tp->tty_incount == buflen(tp->tty_inbuf)) in_transfer(tp);
}
return ct;
}

/*=====
 *                               echo                               *
 *=====*/
PRIVATE int tty_echo(tp, ch)
register tty_t *tp;          /* terminal on which to echo */
register int ch;             /* pointer to character to echo */
{
    /* Echo the character if echoing is on. Some control characters are echoed
     * with their normal effect, other control characters are echoed as "^X",
     * normal characters are echoed normally. EOF (^D) is echoed, but immediately
     * backspaced over. Return the character with the echoed length added to its
     * attributes.
     */
    int len, rp;

    ch &= ~IN_LEN;
    if (!(tp->tty_termios.c_lflag & ECHO)) {
        if (ch == ('\n' | IN_EOT) && (tp->tty_termios.c_lflag
            & (ICANON|ECHONL)) == (ICANON|ECHONL))
            (*tp->tty_echo)(tp, '\n');
        return(ch);
    }

    /* "Reprint" tells if the echo output has been messed up by other output. */
    rp = tp->tty_incount == 0 ? FALSE : tp->tty_reprint;

    if ((ch & IN_CHAR) < ' ') {

```

```

    switch (ch & (IN_ESC|IN_EOF|IN_EOT|IN_CHAR)) {
        case '\t':
            len = 0;
            do {
                (*tp->tty_echo)(tp, ' ');
                len++;
            } while (len < TAB_SIZE && (tp->tty_position & TAB_MASK) != 0);
            break;
        case '\r' | IN_EOT:
        case '\n' | IN_EOT:
            (*tp->tty_echo)(tp, ch & IN_CHAR);
            len = 0;
            break;
        default:
            (*tp->tty_echo)(tp, '^');
            (*tp->tty_echo)(tp, '@' + (ch & IN_CHAR));
            len = 2;
    }
} else
if ((ch & IN_CHAR) == '\177') {
    /* A DEL prints as "^?". */
    (*tp->tty_echo)(tp, '^');
    (*tp->tty_echo)(tp, '?');
    len = 2;
} else {
    (*tp->tty_echo)(tp, ch & IN_CHAR);
    len = 1;
}
if (ch & IN_EOF) while (len > 0) { (*tp->tty_echo)(tp, '\b'); len--; }

tp->tty_reprint = rp;
return(ch | (len << IN_LSHIFT));
}

/*=====
*                                     rawecho                                     *
*=====*/
PRIVATE void rawecho(tp, ch)
register tty_t *tp;
int ch;
{
    /* Echo without interpretation if ECHO is set. */
    int rp = tp->tty_reprint;
    if (tp->tty_termios.c_lflag & ECHO) (*tp->tty_echo)(tp, ch);
    tp->tty_reprint = rp;
}

/*=====
*                                     back_over                                     *
*=====*/
PRIVATE int back_over(tp)
register tty_t *tp;
{
    /* Backspace to previous character on screen and erase it. */
    ul6_t *head;
    int len;

    if (tp->tty_incount == 0) return(0); /* queue empty */
    head = tp->tty_inhead;
    if (head == tp->tty_inbuf) head = bufend(tp->tty_inbuf);
    if (*--head & IN_EOT) return(0); /* can't erase "line breaks" */
    if (tp->tty_reprint) reprint(tp); /* reprint if messed up */
    tp->tty_inhead = head;
    tp->tty_incount--;
    if (tp->tty_termios.c_lflag & ECHOE) {
        len = (*head & IN_LEN) >> IN_LSHIFT;
        while (len > 0) {
            rawecho(tp, '\b');
            rawecho(tp, ' ');
            rawecho(tp, '\b');
            len--;
        }
    }
    return(1); /* one character erased */
}

```

```

}

/*=====
 *
 *                      reprint
 *=====*/
PRIVATE void reprint(tp)
register tty_t *tp;          /* pointer to tty struct */
{
    /* Restore what has been echoed to screen before if the user input has been
     * messed up by output, or if REPRINT (^R) is typed.
     */
    int count;
    ul6_t *head;

    tp->tty_reprint = FALSE;

    /* Find the last line break in the input. */
    head = tp->tty_inhead;
    count = tp->tty_incount;
    while (count > 0) {
        if (head == tp->tty_inbuf) head = bufend(tp->tty_inbuf);
        if (head[-1] & IN_EOT) break;
        head--;
        count--;
    }
    if (count == tp->tty_incount) return;          /* no reason to reprint */

    /* Show REPRINT (^R) and move to a new line. */
    (void) tty_echo(tp, tp->tty_termios.c_cc[VREPRINT] | IN_ESC);
    rawecho(tp, '\r');
    rawecho(tp, '\n');

    /* Reprint from the last break onwards. */
    do {
        if (head == bufend(tp->tty_inbuf)) head = tp->tty_inbuf;
        *head = tty_echo(tp, *head);
        head++;
        count++;
    } while (count < tp->tty_incount);
}

/*=====
 *
 *                      out_process
 *=====*/
PUBLIC void out_process(tp, bstart, bpos, bend, icount, ocount)
tty_t *tp;
char *bstart, *bpos, *bend;          /* start/pos/end of circular buffer */
int *icount;                          /* # input chars / input chars used */
int *ocount;                          /* max output chars / output chars used */
{
    /* Perform output processing on a circular buffer. *icount is the number of
     * bytes to process, and the number of bytes actually processed on return.
     * *ocount is the space available on input and the space used on output.
     * (Naturally *icount < *ocount.) The column position is updated modulo
     * the TAB size, because we really only need it for tabs.
     */

    int tablen;
    int ict = *icount;
    int oct = *ocount;
    int pos = tp->tty_position;

    while (ict > 0) {
        switch (*bpos) {
            case '\7':
                break;
            case '\b':
                pos--;
                break;
            case '\r':
                pos = 0;
                break;
            case '\n':
                if ((tp->tty_termios.c_oflag & (OPOST|ONLCR))

```

```

                                == (OPOST|ONLCR)) {
/* Map LF to CR+LF if there is space. Note that the
 * next character in the buffer is overwritten, so
 * we stop at this point.
 */
if (oct >= 2) {
    *bpos = '\r';
    if (++bpos == bend) bpos = bstart;
    *bpos = '\n';
    pos = 0;
    ict--;
    oct -= 2;
}
goto out_done; /* no space or buffer got changed */
}
break;
case '\t':
/* Best guess for the tab length. */
tablen = TAB_SIZE - (pos & TAB_MASK);

if ((tp->tty_termios.c_oflag & (OPOST|XTABS))
    == (OPOST|XTABS)) {
/* Tabs must be expanded. */
if (oct >= tablen) {
    pos += tablen;
    ict--;
    oct -= tablen;
    do {
        *bpos = ' ';
        if (++bpos == bend) bpos = bstart;
    } while (--tablen != 0);
}
goto out_done;
}
/* Tabs are output directly. */
pos += tablen;
break;
default:
/* Assume any other character prints as one character. */
pos++;
}
if (++bpos == bend) bpos = bstart;
ict--;
oct--;
}
out_done:
tp->tty_position = pos & TAB_MASK;

*icount -= ict; /* [io]ct are the number of chars not used */
*ocount -= oct; /* *[io]count are the number of chars that are used */
}

/*=====
 *                               dev_ioctl                               *
 *=====*/
PRIVATE void dev_ioctl(tp)
tty_t *tp;
{
/* The ioctl's TCSETSW, TCSETSF and TCDRAIN wait for output to finish to make
 * sure that an attribute change doesn't affect the processing of current
 * output. Once output finishes the ioctl is executed as in do_ioctl().
 */
int result;

if (tp->tty_outleft > 0) return; /* output not finished */

if (tp->tty_ioreq != TCDRAIN) {
    if (tp->tty_ioreq == TCSETSF) tty_icancel(tp);
    result = sys_vircopy(tp->tty_ioproc, D, tp->tty_iovir,
        SELF, D, (vir_bytes) &tp->tty_termios,
        (vir_bytes) sizeof(tp->tty_termios));
    setattr(tp);
}
tp->tty_ioreq = 0;

```



```

    tty_reply(REVIVE, tp->tty_iocaller, tp->tty_ioproc, result);
}

/*=====
 *                               setattr                               *
 *=====*/
PRIVATE void setattr(tp)
tty_t *tp;
{
    /* Apply the new line attributes (raw/canonical, line speed, etc.) */
    ul6_t *inp;
    int count;

    if (!(tp->tty_termios.c_lflag & ICANON)) {
        /* Raw mode; put a "line break" on all characters in the input queue.
         * It is undefined what happens to the input queue when ICANON is
         * switched off, a process should use TCSAFLUSH to flush the queue.
         * Keeping the queue to preserve typeahead is the Right Thing, however
         * when a process does use TCSANOW to switch to raw mode.
         */
        count = tp->tty_eotct = tp->tty_incount;
        inp = tp->tty_intail;
        while (count > 0) {
            *inp |= IN_EOT;
            if (++inp == bufend(tp->tty_inbuf)) inp = tp->tty_inbuf;
            --count;
        }
    }

    /* Inspect MIN and TIME. */
    settimer(tp, FALSE);
    if (tp->tty_termios.c_lflag & ICANON) {
        /* No MIN & TIME in canonical mode. */
        tp->tty_min = 1;
    } else {
        /* In raw mode MIN is the number of chars wanted, and TIME how long
         * to wait for them. With interesting exceptions if either is zero.
         */
        tp->tty_min = tp->tty_termios.c_cc[VMIN];
        if (tp->tty_min == 0 && tp->tty_termios.c_cc[VTIME] > 0)
            tp->tty_min = 1;
    }

    if (!(tp->tty_termios.c_iflag & IXON)) {
        /* No start/stop output control, so don't leave output inhibited. */
        tp->tty_inhibited = RUNNING;
        tp->tty_events = 1;
    }

    /* Setting the output speed to zero hangs up the phone. */
    if (tp->tty_termios.c_ospeed == B0) sigchar(tp, SIGHUP);

    /* Set new line speed, character size, etc at the device level. */
    (*tp->tty_ioctl)(tp, 0);
}

/*=====
 *                               tty_reply                               *
 *=====*/
PUBLIC void tty_reply(code, replyee, proc_nr, status)
int code;                /* TASK_REPLY or REVIVE */
int replyee;             /* destination address for the reply */
int proc_nr;             /* to whom should the reply go? */
int status;              /* reply code */
{
    /* Send a reply to a process that wanted to read or write data. */
    message tty_mess;

    tty_mess.m_type = code;
    tty_mess.REP_ENDPT = proc_nr;
    tty_mess.REP_STATUS = status;

    if ((status = send(replyee, &tty_mess)) != OK) {
        printf("TTY: couldn't reply to %d\n", replyee);
    }
}

```

```

        panic("TTY", "tty_reply failed, status\n", status);
    }
}

/*=====
 *                               sigchar                               *
 *=====*/
PUBLIC void sigchar(tp, sig)
register tty_t *tp;
int sig;                                /* SIGINT, SIGQUIT, SIGKILL or SIGHUP */
{
    /* Process a SIGINT, SIGQUIT or SIGKILL char from the keyboard or SIGHUP from
     * a tty close, "stty 0", or a real RS-232 hangup. MM will send the signal to
     * the process group (INT, QUIT), all processes (KILL), or the session leader
     * (HUP).
     */
    int status;

    if (tp->tty_pgrp != 0)
        if (OK != (status = sys_kill(tp->tty_pgrp, sig)))
            panic("TTY", "Error, call to sys_kill failed", status);

    if (!(tp->tty_termios.c_lflag & NOFLSH)) {
        tp->tty_incount = tp->tty_eotct = 0;    /* kill earlier input */
        tp->tty_intail = tp->tty_inhead;
        (*tp->tty_ocancel)(tp, 0);              /* kill all output */
        tp->tty_inhibited = RUNNING;
        tp->tty_events = 1;
    }
}

/*=====
 *                               tty_icancel                               *
 *=====*/
PRIVATE void tty_icancel(tp)
register tty_t *tp;
{
    /* Discard all pending input, tty buffer or device. */

    tp->tty_incount = tp->tty_eotct = 0;
    tp->tty_intail = tp->tty_inhead;
    (*tp->tty_icancel)(tp, 0);
}

/*=====
 *                               tty_init                               *
 *=====*/
PRIVATE void tty_init()
{
    /* Initialize tty structure and call device initialization routines. */

    register tty_t *tp;
    int s;
    struct sigaction sa;

    /* Initialize the terminal lines. */
    for (tp = FIRST_TTY, s=0; tp < END_TTY; tp++, s++) {

        tp->tty_index = s;

        tmr_inittimer(&tp->tty_tmr);

        tp->tty_intail = tp->tty_inhead = tp->tty_inbuf;
        tp->tty_min = 1;
        tp->tty_termios = termios_defaults;
        tp->tty_icancel = tp->tty_ocancel = tp->tty_ioctl = tp->tty_close =
                                                    tty_devnop;

        if (tp < tty_addr(NR_CONS)) {
            scr_init(tp);

            /* Initialize the keyboard driver. */
            kb_init(tp);

            tp->tty_minor = CONS_MINOR + s;

```

```

    } else
    if (tp < tty_addr(NR_CONS+NR_RS_LINES)) {
        rs_init(tp);
        tp->tty_minor = RS232_MINOR + s-NR_CONS;
    } else {
        pty_init(tp);
        tp->tty_minor = s - (NR_CONS+NR_RS_LINES) + TTYPX_MINOR;
    }
}

#if DEAD_CODE
/* Install signal handlers. Ask PM to transform signal into message. */
sa.sa_handler = SIG_MESS;
sigemptyset(&sa.sa_mask);
sa.sa_flags = 0;
if (sigaction(SIGTERM,&sa,NULL)<0) panic("TTY","sigaction failed", errno);
if (sigaction(SIGKMESS,&sa,NULL)<0) panic("TTY","sigaction failed", errno);
if (sigaction(SIGKSTOP,&sa,NULL)<0) panic("TTY","sigaction failed", errno);
#endif
#if DEBUG
printf("end of tty_init\n");
#endif
}

/*=====
 *                               tty_timed_out                               *
 *=====*/
PRIVATE void tty_timed_out(timer_t *tp)
{
/* This timer has expired. Set the events flag, to force processing. */
tty_t *tty_ptr;
tty_ptr = &tty_table[tmr_arg(tp)->ta_int];
tty_ptr->tty_min = 0; /* force read to succeed */
tty_ptr->tty_events = 1;
}

/*=====
 *                               expire_timers                               *
 *=====*/
PRIVATE void expire_timers(void)
{
/* A synchronous alarm message was received. Check if there are any expired
 * timers. Possibly set the event flag and reschedule another alarm.
 */
clock_t now; /* current time */
int s;

/* Get the current time to compare the timers against. */
if ((s=getuptime(&now)) != OK)
    panic("TTY","Couldn't get uptime from clock.", s);

/* Scan the queue of timers for expired timers. This dispatch the watchdog
 * functions of expired timers. Possibly a new alarm call must be scheduled.
 */
tmrs_exptimers(&tty_timers, now, NULL);
if (tty_timers == NULL) tty_next_timeout = TMR_NEVER;
else { /* set new sync alarm */
    tty_next_timeout = tty_timers->tmr_exp_time;
    if ((s=sys_setalarm(tty_next_timeout, 1)) != OK)
        panic("TTY","Couldn't set synchronous alarm.", s);
}
}

/*=====
 *                               settimer                               *
 *=====*/
PRIVATE void settimer(tty_ptr, enable)
tty_t *tty_ptr; /* line to set or unset a timer on */
int enable; /* set timer if true, otherwise unset */
{
    clock_t now; /* current time */
    clock_t exp_time;
    int s;

```

```

/* Get the current time to calculate the timeout time. */
if ((s=getuptime(&now)) != OK)
    panic("TTY", "Couldn't get uptime from clock.", s);
if (enable) {
    exp_time = now + tty_ptr->tty_termios.c_cc[VTIME] * (HZ/10);
    /* Set a new timer for enabling the TTY events flags. */
    tmrs_settimer(&tty_timers, &tty_ptr->tty_tmr,
        exp_time, tty_timed_out, NULL);
} else {
    /* Remove the timer from the active and expired lists. */
    tmrs_clrtimer(&tty_timers, &tty_ptr->tty_tmr, NULL);
}

/* Now check if a new alarm must be scheduled. This happens when the front
 * of the timers queue was disabled or reinserted at another position, or
 * when a new timer was added to the front.
 */
if (tty_timers == NULL) tty_next_timeout = TMR_NEVER;
else if (tty_timers->tmr_exp_time != tty_next_timeout) {
    tty_next_timeout = tty_timers->tmr_exp_time;
    if ((s=sys_setalarm(tty_next_timeout, 1)) != OK)
        panic("TTY", "Couldn't set synchronous alarm.", s);
}
}

/*=====
 *
 *                      tty_devnop
 *=====*/
PUBLIC int tty_devnop(tp, try)
tty_t *tp;
int try;
{
    /* Some functions need not be implemented at the device level. */
}

/*=====
 *
 *                      do_select
 *=====*/
PRIVATE void do_select(tp, m_ptr)
register tty_t *tp;          /* pointer to tty struct */
register message *m_ptr;     /* pointer to message sent to the task */
{
    int ops, ready_ops = 0, watch;

    ops = m_ptr->IO_ENDPT & (SEL_RD|SEL_WR|SEL_ERR);
    watch = (m_ptr->IO_ENDPT & SEL_NOTIFY) ? 1 : 0;

    ready_ops = select_try(tp, ops);

    if (!ready_ops && ops && watch) {
        tp->tty_select_ops |= ops;
        tp->tty_select_proc = m_ptr->m_source;
    }

    tty_reply(TASK_REPLY, m_ptr->m_source, m_ptr->IO_ENDPT, ready_ops);

    return;
}

#if ENABLE_SRCCOMPAT || ENABLE_BINCOMPAT
/*=====
 *
 *                      compat_getp
 *=====*/
PRIVATE int compat_getp(tp, sg)
tty_t *tp;
struct sgttyb *sg;
{
    /* Translate an old TIOCGETP to the termios equivalent. */
    int flgs;

    sg->sg_erase = tp->tty_termios.c_cc[VERASE];
    sg->sg_kill = tp->tty_termios.c_cc[VKILL];
    sg->sg_ospeed = tspd2sgspd(cfgetospeed(&tp->tty_termios));
    sg->sg_ispeed = tspd2sgspd(cfgetispeed(&tp->tty_termios));
}

```

```

    flgs = 0;

    /* XTABS      - if OPOST and XTABS */
    if ((tp->tty_termios.c_oflag & (OPOST|XTABS)) == (OPOST|XTABS))
        flgs |= 0006000;

    /* BITS5..BITS8 - map directly to CS5..CS8 */
    flgs |= (tp->tty_termios.c_cflag & CSIZE) << (8-2);

    /* EVENP      - if PARENB and not PARODD */
    if ((tp->tty_termios.c_cflag & (PARENB|PARODD)) == PARENB)
        flgs |= 0000200;

    /* ODDP       - if PARENB and PARODD */
    if ((tp->tty_termios.c_cflag & (PARENB|PARODD)) == (PARENB|PARODD))
        flgs |= 0000100;

    /* RAW        - if not ICANON and not ISIG */
    if (!(tp->tty_termios.c_lflag & (ICANON|ISIG)))
        flgs |= 0000040;

    /* CRMOD      - if ICRNL */
    if (tp->tty_termios.c_iflag & ICRNL)
        flgs |= 0000020;

    /* ECHO       - if ECHO */
    if (tp->tty_termios.c_lflag & ECHO)
        flgs |= 0000010;

    /* CBREAK     - if not ICANON and ISIG */
    if ((tp->tty_termios.c_lflag & (ICANON|ISIG)) == ISIG)
        flgs |= 0000002;

    sg->sg_flags = flgs;
    return(OK);
}

/*=====
 *                               compat_getc                               *
 *=====*/
PRIVATE int compat_getc(tp, tc)
tty_t *tp;
struct tchars *tc;
{
    /* Translate an old TIOCGETC to the termios equivalent. */

    tc->t_intrc = tp->tty_termios.c_cc[VINTR];
    tc->t_quitc = tp->tty_termios.c_cc[VQUIT];
    tc->t_startc = tp->tty_termios.c_cc[VSTART];
    tc->t_stopc = tp->tty_termios.c_cc[VSTOP];
    tc->t_brkc = tp->tty_termios.c_cc[VEOL];
    tc->t_eofc = tp->tty_termios.c_cc[VEOF];
    return(OK);
}

/*=====
 *                               compat_setp                               *
 *=====*/
PRIVATE int compat_setp(tp, sg)
tty_t *tp;
struct sgttyb *sg;
{
    /* Translate an old TIOCSETP to the termios equivalent. */
    struct termios termios;
    int flags;

    termios = tp->tty_termios;

    termios.c_cc[VERASE] = sg->sg_erase;
    termios.c_cc[VKILL] = sg->sg_kill;
    cfsetispeed(&termios, sgspd2tspd(sg->sg_ispeed & BYTE));
    cfsetospeed(&termios, sgspd2tspd(sg->sg_ospeed & BYTE));
    flags = sg->sg_flags;

```

```
/* Input flags */

/* BRKINT      - not changed */
/* ICRNL       - set if CRMOD is set and not RAW */
/*              (CRMOD also controls output) */
termios.c_iflag &= ~ICRNL;
if ((flags & 0000020) && !(flags & 0000040))
    termios.c_iflag |= ICRNL;

/* IGNBRK      - not changed */
/* IGNCR       - forced off (ignoring cr's is not supported) */
termios.c_iflag &= ~IGNCR;

/* IGNPAR      - not changed */
/* INLCR       - forced off (mapping nl's to cr's is not supported) */
termios.c_iflag &= ~INLCR;

/* INPCK       - not changed */
/* ISTRIP      - not changed */
/* IXOFF       - not changed */
/* IXON        - forced on if not RAW */
termios.c_iflag &= ~IXON;
if (!(flags & 0000040))
    termios.c_iflag |= IXON;

/* PARMRK      - not changed */

/* Output flags */

/* OPOST       - forced on if not RAW */
termios.c_oflag &= ~OPOST;
if (!(flags & 0000040))
    termios.c_oflag |= OPOST;

/* ONLCR       - forced on if CRMOD */
termios.c_oflag &= ~ONLCR;
if (flags & 0000020)
    termios.c_oflag |= ONLCR;

/* XTABS       - forced on if XTABS */
termios.c_oflag &= ~XTABS;
if (flags & 0006000)
    termios.c_oflag |= XTABS;

/* CLOCAL      - not changed */
/* CREAD       - forced on (receiver is always enabled) */
termios.c_cflag |= CREAD;

/* CSIZE       - CS5-CS8 correspond directly to BITS5-BITS8 */
termios.c_cflag = (termios.c_cflag & ~CSIZE) | ((flags & 0001400) >> (8-2));

/* CSTOPB     - not changed */
/* HUPCL      - not changed */
/* PARENB     - set if EVENP or ODDP is set */
termios.c_cflag &= ~PARENB;
if (flags & (0000200|0000100))
    termios.c_cflag |= PARENB;

/* PARODD     - set if ODDP is set */
termios.c_cflag &= ~PARODD;
if (flags & 0000100)
    termios.c_cflag |= PARODD;

/* Local flags */

/* ECHO        - set if ECHO is set */
termios.c_lflag &= ~ECHO;
if (flags & 0000010)
    termios.c_lflag |= ECHO;

/* ECHOE      - not changed */
/* ECHOK      - not changed */
/* ECHONL     - not changed */
```

```

/* ICANON      - set if neither CBREAK nor RAW */
termios.c_lflag &= ~ICANON;
if (!(flags & (0000002|0000040)))
    termios.c_lflag |= ICANON;

/* IEXTEN      - set if not RAW */
/* ISIG        - set if not RAW */
termios.c_lflag &= ~(IEXTEN|ISIG);
if (!(flags & 0000040))
    termios.c_lflag |= (IEXTEN|ISIG);

/* NOFLSH      - not changed */
/* TOSTOP      - not changed */

tp->tty_termios = termios;
setattr(tp);
return(OK);
}

/*=====
*                                     compat_setc                                     *
*=====*/
PRIVATE int compat_setc(tp, tc)
tty_t *tp;
struct tchars *tc;
{
/* Translate an old TIOCSETC to the termios equivalent. */
    struct termios termios;

    termios = tp->tty_termios;

    termios.c_cc[VINTR] = tc->t_intrc;
    termios.c_cc[VQUIT] = tc->t_quitc;
    termios.c_cc[VSTART] = tc->t_startc;
    termios.c_cc[VSTOP] = tc->t_stopc;
    termios.c_cc[VEOL] = tc->t_brkc;
    termios.c_cc[VEOF] = tc->t_eofc;

    tp->tty_termios = termios;
    setattr(tp);
    return(OK);
}

/* Table of termios line speed to sgTTY line speed translations.  All termios
* speeds are present even if sgTTY didn't know about them.  (Now it does.)
*/
PRIVATE struct s2s {
    speed_t      tspd;
    u8_t         sgspd;
} ts2sgs[] = {
    { B0,          0 },
    { B50,         50 },
    { B75,         75 },
    { B110,         1 },
    { B134,        134 },
    { B200,         2 },
    { B300,         3 },
    { B600,         6 },
    { B1200,        12 },
    { B1800,        18 },
    { B2400,        24 },
    { B4800,        48 },
    { B9600,        96 },
    { B19200,       192 },
    { B38400,       195 },
    { B57600,       194 },
    { B115200,      193 },
};

/*=====
*                                     tspd2sgspd                                     *
*=====*/
PRIVATE int tspd2sgspd(tspd)
speed_t tspd;

```

```

{
/* Translate a termios speed to sgTTY speed. */
struct s2s *s;

for (s = ts2sgs; s < ts2sgs + sizeof(ts2sgs)/sizeof(ts2sgs[0]); s++) {
    if (s->tspd == tspd) return(s->sgspd);
}
return 96;
}

/*=====
*                                     sgspd2tspd
*=====*/
PRIVATE speed_t sgspd2tspd(sgspd)
int sgspd;
{
/* Translate a sgTTY speed to termios speed. */
struct s2s *s;

for (s = ts2sgs; s < ts2sgs + sizeof(ts2sgs)/sizeof(ts2sgs[0]); s++) {
    if (s->sgspd == sgspd) return(s->tspd);
}
return B9600;
}

#if ENABLE_BINCOMPAT
/*=====
*                                     do_ioctl_compat
*=====*/
PRIVATE void do_ioctl_compat(tp, m_ptr)
tty_t *tp;
message *m_ptr;
{
/* Handle the old sgTTY ioctl's that packed the sgTTY or tchars struct into
* the Minix message. Efficient then, troublesome now.
*/
int minor, proc, func, result, r;
long flags, erki, spek;
u8_t erase, kill, intr, quit, xon, xoff, brk, eof, ispeed, ospeed;
struct sgTTYb sg;
struct tchars tc;
message reply_mess;

minor = m_ptr->TTY_LINE;
proc = m_ptr->IO_ENDPT;
func = m_ptr->REQUEST;
spek = m_ptr->m2_l1;
flags = m_ptr->m2_l2;

switch(func)
{
case (('t'<<8) | 8): /* TIOCGETP */
    r = compat_getp(tp, &sg);
    erase = sg.sg_erase;
    kill = sg.sg_kill;
    ispeed = sg.sg_ispeed;
    ospeed = sg.sg_ospeed;
    flags = sg.sg_flags;
    erki = ((long)ospeed<<24) | ((long)ispeed<<16) | ((long)erase<<8) | kill;
    break;
case (('t'<<8) | 18): /* TIOCGETC */
    r = compat_getc(tp, &tc);
    intr = tc.t_intrc;
    quit = tc.t_quitc;
    xon = tc.t_startc;
    xoff = tc.t_stopc;
    brk = tc.t_brkc;
    eof = tc.t_eofc;
    erki = ((long)intr<<24) | ((long)quit<<16) | ((long)xon<<8) | xoff;
    flags = (eof << 8) | brk;
    break;
case (('t'<<8) | 17): /* TIOCSETC */
    tc.t_stopc = (spek >> 0) & 0xFF;
    tc.t_startc = (spek >> 8) & 0xFF;

```



```
    tc.t_quitc = (spek >> 16) & 0xFF;
    tc.t_intrc = (spek >> 24) & 0xFF;
    tc.t_brkc = (flags >> 0) & 0xFF;
    tc.t_eofc = (flags >> 8) & 0xFF;
    r = compat_setc(tp, &tc);
    break;
case (('t' << 8) | 9): /* TIOCSETP */
    sg.sg_erase = (spek >> 8) & 0xFF;
    sg.sg_kill = (spek >> 0) & 0xFF;
    sg.sg_ispeed = (spek >> 16) & 0xFF;
    sg.sg_ospeed = (spek >> 24) & 0xFF;
    sg.sg_flags = flags;
    r = compat_setp(tp, &sg);
    break;
default:
    r = ENOTTY;
}
reply_mess.m_type = TASK_REPLY;
reply_mess.REP_ENDPT = m_ptr->IO_ENDPT;
reply_mess.REP_STATUS = r;
reply_mess.m2_l1 = erki;
reply_mess.m2_l2 = flags;
send(m_ptr->m_source, &reply_mess);
}
#endif /* ENABLE_BINCOMPAT */
#endif /* ENABLE_SRCCOMPAT || ENABLE_BINCOMPAT */
```

```

/*      tty.h - Terminals      */

#include <timers.h>
#include "../kernel/const.h"
#include "../kernel/type.h"

#undef lock
#undef unlock

/* First minor numbers for the various classes of TTY devices. */
#define CONS_MINOR      0
#define LOG_MINOR      15
#define RS232_MINOR     16
#define KBD_MINOR      127
#define KBD_AUX_MINOR   126
#define VIDEO_MINOR     125
#define TTYPX_MINOR     128
#define PTYPX_MINOR     192

#define LINEWRAP        1      /* console.c - wrap lines at column 80 */

#define TTY_IN_BYTES     256    /* tty input queue size */
#define TAB_SIZE         8      /* distance between tab stops */
#define TAB_MASK         7      /* mask to compute a tab stop position */

#define ESC              '\33'  /* escape */

#define O_NOCTTY         00400   /* from <fcntl.h>, or cc will choke */
#define O_NONBLOCK       04000

struct tty;
typedef _PROTOTYPE( int (*devfun_t), (struct tty *tp, int try_only) );
typedef _PROTOTYPE( void (*devfunarg_t), (struct tty *tp, int c) );

typedef struct tty {
    int tty_events;          /* set when TTY should inspect this line */
    int tty_index;          /* index into TTY table */
    int tty_minor;          /* device minor number */

    /* Input queue. Typed characters are stored here until read by a program. */
    ul6_t *tty_inhead;      /* pointer to place where next char goes */
    ul6_t *tty_intail;      /* pointer to next char to be given to prog */
    int tty_incount;        /* # chars in the input queue */
    int tty_eotct;          /* number of "line breaks" in input queue */
    devfun_t tty_devread;    /* routine to read from low level buffers */
    devfun_t tty_icancel;    /* cancel any device input */
    int tty_min;            /* minimum requested #chars in input queue */
    timer_t tty_tmr;        /* the timer for this tty */

    /* Output section. */
    devfun_t tty_devwrite;   /* routine to start actual device output */
    devfunarg_t tty_echo;    /* routine to echo characters input */
    devfun_t tty_ocancel;    /* cancel any ongoing device output */
    devfun_t tty_break;      /* let the device send a break */

    /* Terminal parameters and status. */
    int tty_position;        /* current position on the screen for echoing */
    char tty_reprint;        /* 1 when echoed input messed up, else 0 */
    char tty_escaped;        /* 1 when LNEXT (^V) just seen, else 0 */
    char tty_inhibited;      /* 1 when STOP (^S) just seen (stops output) */
    int tty_pgrp;            /* slot number of controlling process */
    char tty_opent;          /* count of number of opens of this tty */

    /* Information about incomplete I/O requests is stored here. */
    int tty_inrepcode;       /* reply code, TASK_REPLY or REVIVE */
    char tty_inrevived;      /* set to 1 if revive callback is pending */
    int tty_incaller;        /* process that made the call (usually FS) */
    int tty_inproc;          /* process that wants to read from tty */
    vir_bytes tty_in_vir;    /* virtual address where data is to go */
    int tty_inleft;          /* how many chars are still needed */
    int tty_incum;           /* # chars input so far */
    int tty_outrepcode;       /* reply code, TASK_REPLY or REVIVE */
    int tty_outrevived;      /* set to 1 if revive callback is pending */
    int tty_outcaller;       /* process that made the call (usually FS) */

```

```

int tty_outproc;           /* process that wants to write to tty */
vir_bytes tty_out_vir;    /* virtual address where data comes from */
int tty_outleft;          /* # chars yet to be output */
int tty_outcum;           /* # chars output so far */
int tty_iocaller;         /* process that made the call (usually FS) */
int tty_ioproc;           /* process that wants to do an ioctl */
int tty_ioreq;            /* ioctl request code */
vir_bytes tty_iovir;      /* virtual address of ioctl buffer */

/* select() data */
int tty_select_ops;       /* which operations are interesting */
int tty_select_proc;      /* which process wants notification */

/* Miscellaneous. */
devfun_t tty_ioctl;       /* set line speed, etc. at the device level */
devfun_t tty_close;       /* tell the device that the tty is closed */
void *tty_priv;           /* pointer to per device private data */
struct termios tty_termios; /* terminal attributes */
struct winsize tty_winsize; /* window size (#lines and #columns) */

ul6_t tty_inbuf[TTY_IN_BYTES]; /* tty input buffer */
} tty_t;

/* Memory allocated in tty.c, so extern here. */
extern tty_t tty_table[NR_CONS+NR_RS_LINES+NR_PTYS];
extern int ccurrent;      /* currently visible console */
extern int irq_hook_id;   /* hook id for keyboard irq */

extern unsigned long kbd_irq_set;
extern unsigned long rs_irq_set;

extern int panicing;      /* From panic.c in sysutil */

/* Values for the fields. */
#define NOT_ESCAPED      0    /* previous character is not LNEXT (^V) */
#define ESCAPED          1    /* previous character was LNEXT (^V) */
#define RUNNING          0    /* no STOP (^S) has been typed to stop output */
#define STOPPED          1    /* STOP (^S) has been typed to stop output */

/* Fields and flags on characters in the input queue. */
#define IN_CHAR          0x00FF /* low 8 bits are the character itself */
#define IN_LEN           0x0F00 /* length of char if it has been echoed */
#define IN_LSHIFT        8     /* length = (c & IN_LEN) >> IN_LSHIFT */
#define IN_EOT           0x1000 /* char is a line break (^D, LF) */
#define IN_EOF           0x2000 /* char is EOF (^D), do not return to user */
#define IN_ESC           0x4000 /* escaped by LNEXT (^V), no interpretation */

/* Times and timeouts. */
#define force_timeout() ((void) (0))

/* Memory allocated in tty.c, so extern here. */
extern timer_t *tty_timers; /* queue of TTY timers */
extern clock_t tty_next_timeout; /* next TTY timeout */

/* Number of elements and limit of a buffer. */
#define buflen(buf)      (sizeof(buf) / sizeof((buf)[0]))
#define bufend(buf)      ((buf) + buflen(buf))

/* Memory allocated in tty.c, so extern here. */
extern struct machine machine; /* machine information (a.o.: pc_at, ega) */

/* The tty outputs diagnostic messages in a circular buffer. */
extern struct kmessages kmess;

/* Function prototypes for TTY driver. */
/* tty.c */
_PROTOTYPE( void handle_events, (struct tty *tp) );
_PROTOTYPE( void sigchar, (struct tty *tp, int sig) );
_PROTOTYPE( void tty_task, (void) );
_PROTOTYPE( int in_process, (struct tty *tp, char *buf, int count) );
_PROTOTYPE( void out_process, (struct tty *tp, char *bstart, char *bpos,
                             char *bend, int *icount, int *ocount) );
_PROTOTYPE( void tty_wakeup, (clock_t now) );

```

```
_PROTOTYPE( void tty_reply, (int code, int replyee, int proc_nr,
                             int status) ) ;
_PROTOTYPE( int tty_devnop, (struct tty *tp, int try) ) ;
_PROTOTYPE( int select_try, (struct tty *tp, int ops) ) ;
_PROTOTYPE( int select_retry, (struct tty *tp) ) ;

/* rs232.c */
_PROTOTYPE( void rs_init, (struct tty *tp) ) ;
_PROTOTYPE( void rs_interrupt, (message *m) ) ;

#if (CHIP == INTEL)
/* console.c */
_PROTOTYPE( void kputc, (int c) ) ;
_PROTOTYPE( void cons_stop, (void) ) ;
_PROTOTYPE( void do_new_kmess, (message *m) ) ;
_PROTOTYPE( void do_diagnostics, (message *m) ) ;
_PROTOTYPE( void do_get_kmess, (message *m) ) ;
_PROTOTYPE( void scr_init, (struct tty *tp) ) ;
_PROTOTYPE( void toggle_scroll, (void) ) ;
_PROTOTYPE( int con_loadfont, (message *m) ) ;
_PROTOTYPE( void select_console, (int cons_line) ) ;
_PROTOTYPE( void beep_x, ( unsigned freq, clock_t dur) ) ;
_PROTOTYPE( void do_video, (message *m) ) ;

/* keyboard.c */
_PROTOTYPE( void kb_init, (struct tty *tp) ) ;
_PROTOTYPE( void kb_init_once, (void) ) ;
_PROTOTYPE( int kbd_loadmap, (message *m) ) ;
_PROTOTYPE( void do_panic_dumps, (message *m) ) ;
_PROTOTYPE( void do_fkey_ctl, (message *m) ) ;
_PROTOTYPE( void kbd_interrupt, (message *m) ) ;
_PROTOTYPE( void do_kbd, (message *m) ) ;
_PROTOTYPE( void do_kbdaux, (message *m) ) ;
_PROTOTYPE( int kbd_status, (message *m_ptr) ) ;

/* pty.c */
_PROTOTYPE( void do_pty, (struct tty *tp, message *m_ptr) ) ;
_PROTOTYPE( void pty_init, (struct tty *tp) ) ;
_PROTOTYPE( void select_retry_pty, (struct tty *tp) ) ;
_PROTOTYPE( int pty_status, (message *m_ptr) ) ;

/* vidcopy.s */
_PROTOTYPE( void vid_vid_copy, (unsigned src, unsigned dst, unsigned count));
_PROTOTYPE( void mem_vid_copy, (ul6_t *src, unsigned dst, unsigned count));

#endif /* (CHIP == INTEL) */
```

```

#
! This file contains two specialized assembly code routines to update the
! video memory. The routines can copy from user to video memory, or from
! video to video memory.

! sections

.sect .text; .sect .rom; .sect .data; .sect .bss

! exported functions

.define _mem_vid_copy    ! copy data to video ram
.define _vid_vid_copy    ! move data in video ram

! The routines only guarantee to preserve the registers the C compiler
! expects to be preserved (ebx, esi, edi, ebp, esp, segment registers, and
! direction bit in the flags).

.sect .text
!*=====*
!*                               mem_vid_copy                               *
!*=====*
! PUBLIC void mem_vid_copy(u16 *src, unsigned dst, unsigned count);
!
! Copy count characters from kernel memory to video memory. Src is an ordinary
! pointer to a word, but dst and count are character (word) based video offset
! and count. If src is null then screen memory is blanked by filling it with
! blank_color.

_mem_vid_copy:
    push    ebp
    mov     ebp, esp
    push    esi
    push    edi
    push    es
    mov     esi, 8(ebp)          ! source
    mov     edi, 12(ebp)         ! destination
    mov     edx, 16(ebp)         ! count
    mov     es, (_vid_seg)       ! segment containing video memory
    cld                          ! make sure direction is up
mvc_loop:
    and     edi, (_vid_mask)     ! wrap address
    mov     ecx, edx             ! one chunk to copy
    mov     eax, (_vid_size)
    sub     eax, edi
    cmp     ecx, eax
    jbe     0f
    mov     ecx, eax             ! ecx = min(ecx, vid_size - edi)
0:    sub     edx, ecx            ! count -= ecx
    shl     edi, 1               ! byte address
    add     edi, (_vid_off)      ! in video memory
    test    esi, esi             ! source == 0 means blank the screen
    jz      mvc_blank
mvc_copy:
    rep     rep                  ! copy words to video memory
    016 movs
    jmp     mvc_test
mvc_blank:
    mov     eax, (_blank_color)  ! ax = blanking character
    rep     rep
    016 stos                     ! copy blanks to video memory
    ! jmp     mvc_test
mvc_test:
    sub     edi, (_vid_off)
    shr     edi, 1               ! back to a word address
    test    edx, edx
    jnz     mvc_loop
mvc_done:
    pop     es
    pop     edi
    pop     esi
    pop     ebp
    ret

```

```

!*=====*
!*                                     vid_vid_copy                                     *
!*=====*
! PUBLIC void vid_vid_copy(unsigned src, unsigned dst, unsigned count);
!
! Copy count characters from video memory to video memory.  Handle overlap.
! Used for scrolling, line or character insertion and deletion.  Src, dst
! and count are character (word) based video offsets and count.

```

```

_vid_vid_copy:
    push    ebp
    mov     ebp, esp
    push    esi
    push    edi
    push    es
    mov     esi, 8(ebp)          ! source
    mov     edi, 12(ebp)         ! destination
    mov     edx, 16(ebp)         ! count
    mov     es, (_vid_seg)       ! segment containing video memory
    cmp     esi, edi             ! copy up or down?
    jb      vvc_down

vvc_up:
    cld                          ! direction is up
vvc_uploop:
    and     esi, (_vid_mask)     ! wrap addresses
    and     edi, (_vid_mask)
    mov     ecx, edx             ! one chunk to copy
    mov     eax, (_vid_size)
    sub     eax, esi
    cmp     ecx, eax
    jbe     0f
    mov     ecx, eax             ! ecx = min(ecx, vid_size - esi)
0:    mov     eax, (_vid_size)
    sub     eax, edi
    cmp     ecx, eax
    jbe     0f
    mov     ecx, eax             ! ecx = min(ecx, vid_size - edi)
0:    sub     edx, ecx            ! count -= ecx
    call    vvc_copy             ! copy video words
    test    edx, edx
    jnz     vvc_uploop           ! again?
    jmp     vvc_done

vvc_down:
    std                          ! direction is down
    lea     esi, -1(esi)(edx*1)   ! start copying at the top
    lea     edi, -1(edi)(edx*1)
vvc_downloop:
    and     esi, (_vid_mask)     ! wrap addresses
    and     edi, (_vid_mask)
    mov     ecx, edx             ! one chunk to copy
    lea     eax, 1(esi)
    cmp     ecx, eax
    jbe     0f
    mov     ecx, eax             ! ecx = min(ecx, esi + 1)
0:    lea     eax, 1(edi)
    cmp     ecx, eax
    jbe     0f
    mov     ecx, eax             ! ecx = min(ecx, edi + 1)
0:    sub     edx, ecx            ! count -= ecx
    call    vvc_copy
    test    edx, edx
    jnz     vvc_downloop         ! again?
    cld
    ! jmp     vvc_done           ! C compiler expects up

vvc_done:
    pop     es
    pop     edi
    pop     esi
    pop     ebp
    ret

```

```

! Copy video words.  (Inner code of both the up and downcopying loop.)
vvc_copy:

```

```
    shl     esi, 1
    shl     edi, 1                ! byte addresses
    add     esi, (_vid_off)
    add     edi, (_vid_off)      ! in video memory
    rep
eseg o16 movs                    ! copy video words
    sub     esi, (_vid_off)
    sub     edi, (_vid_off)
    shr     esi, 1
    shr     edi, 1                ! back to word addresses
    ret
```

```
# Generate binary keymaps.

LK =      /usr/lib/keymaps

.SUFFIXES:      .src .map

.src.map:
$(CC) -DKEYSRC=\"${<}\" genmap.c
./a.out > $@
@rm -f a.out

all:      \
dvorak.map \
french.map \
german.map \
italian.map \
japanese.map \
latin-america.map \
olivetti.map \
polish.map \
scandinavian.map \
spanish.map \
uk.map \
us-std.map \
us-swap.map \

install:      \
$(LK) \
$(LK)/dvorak.map \
$(LK)/french.map \
$(LK)/german.map \
$(LK)/italian.map \
$(LK)/japanese.map \
$(LK)/latin-america.map \
$(LK)/olivetti.map \
$(LK)/polish.map \
$(LK)/scandinavian.map \
$(LK)/spanish.map \
$(LK)/uk.map \
$(LK)/us-std.map \
$(LK)/us-swap.map \

clean:
rm -f a.out *.map

$(LK):
install -d $@

$(LK)/dvorak.map:      dvorak.map
install -c $? $@

$(LK)/french.map:      french.map
install -c $? $@

$(LK)/german.map:      german.map
install -c $? $@

$(LK)/italian.map:      italian.map
install -c $? $@

$(LK)/japanese.map:      japanese.map
install -c $? $@

$(LK)/latin-america.map:      latin-america.map
install -c $? $@

$(LK)/olivetti.map:      olivetti.map
install -c $? $@

$(LK)/polish.map:      polish.map
install -c $? $@

$(LK)/scandinavian.map:      scandinavian.map
install -c $? $@
```



```
$(LK)/spanish.map:    spanish.map
    install -c $? $@

$(LK)/uk.map:    uk.map
    install -c $? $@

$(LK)/us-std.map:    us-std.map
    install -c $? $@

$(LK)/us-swap.map:    us-swap.map
    install -c $? $@
```

```

/* Keymap for Dvorak keyboard.
 * Contributed by: Ulrich Hobelmann <u.hobelmann@web.de>
 */

u16_t keymap[NR_SCAN_CODES * MAP_COLS] = {

/* scan-code      !Shift  Shift    Alt1     Alt2     Alt+Sh  Ctrl     */
/* ===== */
/* 00 - none      */      0,        0,        0,        0,        0,        0,
/* 01 - ESC       */      C('['), C('['), CA('['), CA('['), CA('['), C('['),
/* 02 - '1'       */      '1',      '!',      A('1'), A('1'), A('!'), C('A'),
/* 03 - '2'       */      '2',      '@',      A('2'), A('2'), A('@'), C('@'),
/* 04 - '3'       */      '3',      '#',      A('3'), A('3'), A('#'), C('C'),
/* 05 - '4'       */      '4',      '$',      A('4'), A('4'), A('$'), C('D'),
/* 06 - '5'       */      '5',      '%',      A('5'), A('5'), A('%'), C('E'),
/* 07 - '6'       */      '6',      '^',      A('6'), A('6'), A('^'), C('^'),
/* 08 - '7'       */      '7',      '&',      A('7'), A('7'), A('&'), C('G'),
/* 09 - '8'       */      '8',      '*',      A('8'), A('8'), A('*'), C('H'),
/* 10 - '9'       */      '9',      '(',      A('9'), A('9'), A('('), C('I'),
/* 11 - '0'       */      '0',      ')',      A('0'), A('0'), A(')'), C('@'),
/* 12 - '-'       */      '[',      '{',      A('['), A('['), A('}'), C(']'),
/* 13 - '='       */      ']',      '}',      A(']'), A(']'), A('}') C('['),
/* 14 - BS        */      C('H'), C('H'), CA('H'), CA('H'), CA('H'), 0177,
/* 15 - TAB       */      C('I'), C('I'), CA('I'), CA('I'), CA('I'), C('I'),
/* 16 - 'q'       */      '\',      '"',      A('\'), A('\'), A('"'), C('@'),
/* 17 - 'w'       */      ',',      '<',      A(','), A(','), A('<'), C('@'),
/* 18 - 'e'       */      '.',      '>',      A('.'), A('.'), A('>'), C('@'),
/* 19 - 'r'       */      L('p'), 'P',      A('p'), A('p'), A('P'), C('P'),
/* 20 - 't'       */      L('y'), 'Y',      A('y'), A('y'), A('Y'), C('Y'),
/* 21 - 'y'       */      L('f'), 'F',      A('f'), A('f'), A('F'), C('F'),
/* 22 - 'u'       */      L('g'), 'G',      A('g'), A('g'), A('G'), C('G'),
/* 23 - 'i'       */      L('c'), 'C',      A('c'), A('c'), A('C'), C('C'),
/* 24 - 'o'       */      L('r'), 'R',      A('r'), A('r'), A('R'), C('R'),
/* 25 - 'p'       */      L('l'), 'L',      A('l'), A('l'), A('L'), C('L'),
/* 26 - '['       */      '/',      '?',      A('/'), A('/'), A('?'), C('@'),
/* 27 - ']'       */      '=',      '+',      A('='), A('='), A('+'), C('@'),
/* 28 - CR/LF     */      C('M'), C('M'), CA('M'), CA('M'), CA('M'), C('J'),
/* 29 - Ctrl      */      CTRL,  CTRL,  CTRL,  CTRL,  CTRL,  CTRL,
/* 30 - 'a'       */      L('a'), 'A',      A('a'), A('a'), A('A'), C('A'),
/* 31 - 's'       */      L('o'), 'O',      A('o'), A('o'), A('O'), C('O'),
/* 32 - 'd'       */      L('e'), 'E',      A('e'), A('e'), A('E'), C('E'),
/* 33 - 'f'       */      L('u'), 'U',      A('u'), A('u'), A('U'), C('U'),
/* 34 - 'g'       */      L('i'), 'I',      A('i'), A('i'), A('I'), C('I'),
/* 35 - 'h'       */      L('d'), 'D',      A('d'), A('d'), A('D'), C('D'),
/* 36 - 'j'       */      L('h'), 'H',      A('h'), A('h'), A('H'), C('H'),
/* 37 - 'k'       */      L('t'), 'T',      A('t'), A('t'), A('T'), C('T'),
/* 38 - 'l'       */      L('n'), 'N',      A('n'), A('n'), A('N'), C('N'),
/* 39 - ';'       */      L('s'), 'S',      A('s'), A('s'), A('S'), C('S'),
/* 40 - '\'       */      '-',      '_',      A('-'), A('-'), A('_'), C('_'),
/* 41 - '~'       */      '~',      '~',      A('~'), A('~'), A('~'), C('@'),
/* 42 - l. SHIFT  */      SHIFT, SHIFT, SHIFT, SHIFT, SHIFT, SHIFT,
/* 43 - '\\'      */      '\\',  '|',      A('\\'), A('\\'), A('|'), C('\\'),
/* 44 - 'z'       */      ';',      ':',      A(';'), A(';'), A(':'), C('@'),
/* 45 - 'x'       */      L('q'), 'Q',      A('q'), A('q'), A('Q'), C('Q'),
/* 46 - 'c'       */      L('j'), 'J',      A('j'), A('j'), A('J'), C('J'),
/* 47 - 'v'       */      L('k'), 'K',      A('k'), A('k'), A('K'), C('K'),
/* 48 - 'b'       */      L('x'), 'X',      A('x'), A('x'), A('X'), C('X'),
/* 49 - 'n'       */      L('b'), 'B',      A('b'), A('b'), A('B'), C('B'),
/* 50 - 'm'       */      L('m'), 'M',      A('m'), A('m'), A('M'), C('M'),
/* 51 - ','       */      L('w'), 'W',      A('w'), A('w'), A('W'), C('W'),
/* 52 - '.'       */      L('v'), 'V',      A('v'), A('v'), A('V'), C('V'),
/* 53 - '/'       */      L('z'), 'Z',      A('z'), A('z'), A('Z'), C('Z'),
/* 54 - r. SHIFT  */      SHIFT, SHIFT, SHIFT, SHIFT, SHIFT, SHIFT,
/* 55 - '*'       */      '*',      '*',      A('*'), A('*'), A('*'), C('@'),
/* 56 - ALT       */      ALT,      ALT,      ALT,      ALT,      ALT,      ALT,
/* 57 - ' '       */      ' ',      ' ',      A(' '), A(' '), A(' '), C('@'),
/* 58 - CapsLck   */      CALOCK, CALOCK, CALOCK, CALOCK, CALOCK, CALOCK,
/* 59 - F1        */      F1,      SF1,      AF1,      AF1,      ASF1,      CF1,
/* 60 - F2        */      F2,      SF2,      AF2,      AF2,      ASF2,      CF2,
/* 61 - F3        */      F3,      SF3,      AF3,      AF3,      ASF3,      CF3,
/* 62 - F4        */      F4,      SF4,      AF4,      AF4,      ASF4,      CF4,
/* 63 - F5        */      F5,      SF5,      AF5,      AF5,      ASF5,      CF5,
/* 64 - F6        */      F6,      SF6,      AF6,      AF6,      ASF6,      CF6,

```

```

/* 65 - F7      */      F7,      SF7,      AF7,      AF7,      ASF7,      CF7,
/* 66 - F8      */      F8,      SF8,      AF8,      AF8,      ASF8,      CF8,
/* 67 - F9      */      F9,      SF9,      AF9,      AF9,      ASF9,      CF9,
/* 68 - F10     */      F10,     SF10,     AF10,     AF10,     ASF10,     CF10,
/* 69 - NumLock */      NLOCK,   NLOCK,   NLOCK,   NLOCK,   NLOCK,   NLOCK,
/* 70 - ScrLock */      SLOCK,   SLOCK,   SLOCK,   SLOCK,   SLOCK,   SLOCK,
/* 71 - Home    */      HOME,    '7',    AHOME,    AHOME,    A('7'),  CHOME,
/* 72 - CurUp   */      UP,      '8',    AUP,      AUP,      A('8'),  CUP,
/* 73 - PgUp    */      PGUP,    '9',    APGUP,    APGUP,    A('9'),  CPGUP,
/* 74 - '-'     */      NMIN,    '-',    ANMIN,    ANMIN,    A('-'),  CNMIN,
/* 75 - Left    */      LEFT,    '4',    ALEFT,    ALEFT,    A('4'),  CLEFT,
/* 76 - MID     */      MID,     '5',    AMID,     AMID,     A('5'),  CMID,
/* 77 - Right   */      RIGHT,   '6',    ARIGHT,   ARIGHT,   A('6'),  CRIGHT,
/* 78 - '+'     */      PLUS,    '+',    APLUS,    APLUS,    A('+'),  CPLUS,
/* 79 - End     */      END,     '1',    AEND,     AEND,     A('1'),  CEND,
/* 80 - Down    */      DOWN,    '2',    ADOWN,    ADOWN,    A('2'),  CDOWN,
/* 81 - PgDown  */      PGDN,    '3',    APGDN,    APGDN,    A('3'),  CPGDN,
/* 82 - Insert  */      INSRT,   '0',    AINSRT,   AINSRT,   A('0'),  CINSRT,
/* 83 - Delete  */      0177,   '.',    A(0177),  A(0177),  A('.'),  0177,
/* 84 - Enter   */      C('M'),  C('M'),  CA('M'),  CA('M'),  CA('M'),  C('J'),
/* 85 - ???     */      0,      0,      0,      0,      0,      0,
/* 86 - ???     */      '<',    '>',    A('<'),  A('|'),  A('>'),  C('@'),
/* 87 - F11     */      F11,     SF11,    AF11,     AF11,     ASF11,    CF11,
/* 88 - F12     */      F12,     SF12,    AF12,     AF12,     ASF12,    CF12,
/* 89 - ???     */      0,      0,      0,      0,      0,      0,
/* 90 - ???     */      0,      0,      0,      0,      0,      0,
/* 91 - ???     */      0,      0,      0,      0,      0,      0,
/* 92 - ???     */      0,      0,      0,      0,      0,      0,
/* 93 - ???     */      0,      0,      0,      0,      0,      0,
/* 94 - ???     */      0,      0,      0,      0,      0,      0,
/* 95 - ???     */      0,      0,      0,      0,      0,      0,
/* 96 - EXT_KEY */      EXTKEY,  EXTKEY,  EXTKEY,  EXTKEY,  EXTKEY,  EXTKEY,
/* 97 - ???     */      0,      0,      0,      0,      0,      0,
/* 98 - ???     */      0,      0,      0,      0,      0,      0,
/* 99 - ???     */      0,      0,      0,      0,      0,      0,
/*100 - ???     */      0,      0,      0,      0,      0,      0,
/*101 - ???     */      0,      0,      0,      0,      0,      0,
/*102 - ???     */      0,      0,      0,      0,      0,      0,
/*103 - ???     */      0,      0,      0,      0,      0,      0,
/*104 - ???     */      0,      0,      0,      0,      0,      0,
/*105 - ???     */      0,      0,      0,      0,      0,      0,
/*106 - ???     */      0,      0,      0,      0,      0,      0,
/*107 - ???     */      0,      0,      0,      0,      0,      0,
/*108 - ???     */      0,      0,      0,      0,      0,      0,
/*109 - ???     */      0,      0,      0,      0,      0,      0,
/*110 - ???     */      0,      0,      0,      0,      0,      0,
/*111 - ???     */      0,      0,      0,      0,      0,      0,
/*112 - ???     */      0,      0,      0,      0,      0,      0,
/*113 - ???     */      0,      0,      0,      0,      0,      0,
/*114 - ???     */      0,      0,      0,      0,      0,      0,
/*115 - ???     */      0,      0,      0,      0,      0,      0,
/*116 - ???     */      0,      0,      0,      0,      0,      0,
/*117 - ???     */      0,      0,      0,      0,      0,      0,
/*118 - ???     */      0,      0,      0,      0,      0,      0,
/*119 - ???     */      0,      0,      0,      0,      0,      0,
/*120 - ???     */      0,      0,      0,      0,      0,      0,
/*121 - ???     */      0,      0,      0,      0,      0,      0,
/*122 - ???     */      0,      0,      0,      0,      0,      0,
/*123 - ???     */      0,      0,      0,      0,      0,      0,
/*124 - ???     */      0,      0,      0,      0,      0,      0,
/*125 - ???     */      0,      0,      0,      0,      0,      0,
/*126 - ???     */      0,      0,      0,      0,      0,      0,
/*127 - ???     */      0,      0,      0,      0,      0,      0
};

```

```
/* Keymap for the French keyboard. */
```

```
u16_t keymap[NR_SCAN_CODES * MAP_COLS] = {
```

```
/* scan-code      !Shift  Shift  Alt    AltGr  Alt+Sh  Ctrl    */
/* ===== */
/* 00 - none      */      0,      0,      0,      0,      0,      0,
/* 01 - ESC       */      C('['), C('['), CA('['), C('['), C('['), C('['),
/* 02 - '1'       */      '&',    '1',    A('1'), '&',    '1',    C('A'),
/* 03 - '2'       */      0202,   '2',    A('2'), '~',    '2',    C('B'),
/* 04 - '3'       */      '"',    '3',    A('3'), '#',    '3',    C('C'),
/* 05 - '4'       */      '\',    '4',    A('4'), '{',    '4',    C('D'),
/* 06 - '5'       */      '(',    '5',    A('5'), '[',    '5',    C('E'),
/* 07 - '6'       */      '-',    '6',    A('6'), '|',    '6',    C('F'),
/* 08 - '7'       */      0212,   '7',    A('7'), '\',    '7',    C('G'),
/* 09 - '8'       */      '_',    '8',    A('8'), '\\',   '8',    C('H'),
/* 10 - '9'       */      0207,   '9',    A('9'), '^',    '9',    C('I'),
/* 11 - '0'       */      0205,   '0',    A('0'), '@',    '0',    C('J'),
/* 12 - '-'       */      ')',    0370,  A(')'), ']',    '-',    C('K'),
/* 13 - '='       */      '=',    '+',    A('='), '}',    '=',    C('L'),
/* 14 - BS        */      C('H'), C('H'), CA('H'), C('H'), C('H'), 0177,
/* 15 - TAB       */      C('I'), C('I'), CA('I'), C('I'), C('I'), C('I'),
/* 16 - 'q'       */      L('a'), 'A',    A('a'), 'a',    'Q',    C('A'),
/* 17 - 'w'       */      L('z'), 'Z',    A('z'), 'z',    'W',    C('Z'),
/* 18 - 'e'       */      L('e'), 'E',    A('e'), 'e',    'E',    C('E'),
/* 19 - 'r'       */      L('r'), 'R',    A('r'), 'r',    'R',    C('R'),
/* 20 - 't'       */      L('t'), 'T',    A('t'), 't',    'T',    C('T'),
/* 21 - 'y'       */      L('y'), 'Y',    A('y'), 'y',    'Y',    C('Y'),
/* 22 - 'u'       */      L('u'), 'U',    A('u'), 'u',    'U',    C('U'),
/* 23 - 'i'       */      L('i'), 'I',    A('i'), 'i',    'I',    C('I'),
/* 24 - 'o'       */      L('o'), 'O',    A('o'), 'o',    'O',    C('O'),
/* 25 - 'p'       */      L('p'), 'P',    A('p'), 'p',    'P',    C('P'),
/* 26 - '['       */      '^',    '"',    A('^'), '^',    '[',    C('^'),
/* 27 - ']'       */      '$',    0234,  A('$'), '$',    ']',    C('$'),
/* 28 - CR/LF     */      C('M'), C('M'), CA('M'), C('M'), C('M'), C('J'),
/* 29 - Ctrl      */      CTRL,   CTRL,   CTRL,   CTRL,   CTRL,   CTRL,
/* 30 - 'a'       */      L('q'), 'Q',    A('q'), 'q',    'A',    C('Q'),
/* 31 - 's'       */      L('s'), 'S',    A('s'), 's',    'S',    C('S'),
/* 32 - 'd'       */      L('d'), 'D',    A('d'), 'd',    'D',    C('D'),
/* 33 - 'f'       */      L('f'), 'F',    A('f'), 'f',    'F',    C('F'),
/* 34 - 'g'       */      L('g'), 'G',    A('g'), 'g',    'G',    C('G'),
/* 35 - 'h'       */      L('h'), 'H',    A('h'), 'h',    'H',    C('H'),
/* 36 - 'j'       */      L('j'), 'J',    A('j'), 'j',    'J',    C('J'),
/* 37 - 'k'       */      L('k'), 'K',    A('k'), 'k',    'K',    C('K'),
/* 38 - 'l'       */      L('l'), 'L',    A('l'), 'l',    'L',    C('L'),
/* 39 - ';'       */      L('m'), 'M',    A('m'), 'm',    'M',    C('M'),
/* 40 - '\'       */      0227,   '%',    A('%'), 0227,   '\\',   C('G'),
/* 41 - '`'       */      0375,   0375,  0375,   0375,   '',     C('['),
/* 42 - l. SHIFT */      SHIFT,  SHIFT,  SHIFT,  SHIFT,  SHIFT,  SHIFT,
/* 43 - '`'       */      '*',    0346,  A('*'), '*',    '',     C('*'),
/* 44 - 'z'       */      L('w'), 'W',    A('w'), 'w',    'Z',    C('W'),
/* 45 - 'x'       */      L('x'), 'X',    A('x'), 'x',    'X',    C('X'),
/* 46 - 'c'       */      L('c'), 'C',    A('c'), 'c',    'C',    C('C'),
/* 47 - 'v'       */      L('v'), 'V',    A('v'), 'v',    'V',    C('V'),
/* 48 - 'b'       */      L('b'), 'B',    A('b'), 'b',    'B',    C('B'),
/* 49 - 'n'       */      L('n'), 'N',    A('n'), 'n',    'N',    C('N'),
/* 50 - 'm'       */      ',',    '?',    A(','), ',',    'm',    C('@'),
/* 51 - ':'       */      ';',    ',',    A(';'), ':',    '',     C('@'),
/* 52 - '.'       */      ':',    '/',    A(':'), ':',    '',     C('@'),
/* 53 - '/'       */      '!',   '$'/*025*/, A('!'), '!',   '/',    C('@'),
/* 54 - r. SHIFT */      SHIFT,  SHIFT,  SHIFT,  SHIFT,  SHIFT,  SHIFT,
/* 55 - '*'       */      '*',    '*',    A('*'), '*',    '',     C('@'),
/* 56 - ALT       */      ALT,    ALT,    ALT,    ALT,    ALT,    ALT,
/* 57 - ' '       */      ' ',    ' ',    A(' '), ' ',    ' ',     C('@'),
/* 58 - CapsLck   */      CALOCK,  CALOCK,  CALOCK,  CALOCK,  CALOCK,  CALOCK,
/* 59 - F1        */      F1,     SF1,    AF1,    AF1,    ASF1,    CF1,
/* 60 - F2        */      F2,     SF2,    AF2,    AF2,    ASF2,    CF2,
/* 61 - F3        */      F3,     SF3,    AF3,    AF3,    ASF3,    CF3,
/* 62 - F4        */      F4,     SF4,    AF4,    AF4,    ASF4,    CF4,
/* 63 - F5        */      F5,     SF5,    AF5,    AF5,    ASF5,    CF5,
/* 64 - F6        */      F6,     SF6,    AF6,    AF6,    ASF6,    CF6,
/* 65 - F7        */      F7,     SF7,    AF7,    AF7,    ASF7,    CF7,
/* 66 - F8        */      F8,     SF8,    AF8,    AF8,    ASF8,    CF8,
/* 67 - F9        */      F9,     SF9,    AF9,    AF9,    ASF9,    CF9,
```

```

/* 68 - F10 */ F10, SF10, AF10, AF10, ASF10, CF10,
/* 69 - NumLock */ NLOCK, NLOCK, NLOCK, NLOCK, NLOCK, NLOCK,
/* 70 - ScrLock */ SLOCK, SLOCK, SLOCK, SLOCK, SLOCK, SLOCK,
/* 71 - Home */ HOME, '7', AHOME, AHOME, '7', CHOME,
/* 72 - CurUp */ UP, '8', AUP, AUP, '8', CUP,
/* 73 - PgUp */ PGUP, '9', APGUP, APGUP, '9', CPGUP,
/* 74 - '-' */ NMIN, '-', ANMIN, ANMIN, '-', CNMIN,
/* 75 - Left */ LEFT, '4', ALEFT, ALEFT, '4', CLEFT,
/* 76 - MID */ MID, '5', AMID, AMID, '5', CMID,
/* 77 - Right */ RIGHT, '6', ARIGHT, ARIGHT, '6', CRIGHT,
/* 78 - '+' */ PLUS, '+', APLUS, APLUS, '+', CPLUS,
/* 79 - End */ END, '1', AEND, AEND, '1', CEND,
/* 80 - Down */ DOWN, '2', ADOWN, ADOWN, '2', CDOWN,
/* 81 - PgDown */ PGDN, '3', APGDN, APGDN, '3', CPGDN,
/* 82 - Insert */ INSRT, '0', AINSRT, AINSRT, '0', CINSRT,
/* 83 - Delete */ 0177, '.', A(0177), 0177, '.', 0177,
/* 84 - Enter */ C('M'), C('M'), CA('M'), C('M'), C('M'), C('J'),
/* 85 - ??? */ 0, 0, 0, 0, 0, 0,
/* 86 - ??? */ '<', '>', A('<'), '<', '>', C('@'),
/* 87 - F11 */ F11, SF11, AF11, AF11, ASF11, CF11,
/* 88 - F12 */ F12, SF12, AF12, AF12, ASF12, CF12,
/* 89 - ??? */ 0, 0, 0, 0, 0, 0,
/* 90 - ??? */ 0, 0, 0, 0, 0, 0,
/* 91 - ??? */ 0, 0, 0, 0, 0, 0,
/* 92 - ??? */ 0, 0, 0, 0, 0, 0,
/* 93 - ??? */ 0, 0, 0, 0, 0, 0,
/* 94 - ??? */ 0, 0, 0, 0, 0, 0,
/* 95 - ??? */ 0, 0, 0, 0, 0, 0,
/* 96 - EXT_KEY */ EXTKEY, EXTKEY, EXTKEY, EXTKEY, EXTKEY, EXTKEY,
/* 97 - ??? */ 0, 0, 0, 0, 0, 0,
/* 98 - ??? */ 0, 0, 0, 0, 0, 0,
/* 99 - ??? */ 0, 0, 0, 0, 0, 0,
/*100 - ??? */ 0, 0, 0, 0, 0, 0,
/*101 - ??? */ 0, 0, 0, 0, 0, 0,
/*102 - ??? */ 0, 0, 0, 0, 0, 0,
/*103 - ??? */ 0, 0, 0, 0, 0, 0,
/*104 - ??? */ 0, 0, 0, 0, 0, 0,
/*105 - ??? */ 0, 0, 0, 0, 0, 0,
/*106 - ??? */ 0, 0, 0, 0, 0, 0,
/*107 - ??? */ 0, 0, 0, 0, 0, 0,
/*108 - ??? */ 0, 0, 0, 0, 0, 0,
/*109 - ??? */ 0, 0, 0, 0, 0, 0,
/*110 - ??? */ 0, 0, 0, 0, 0, 0,
/*111 - ??? */ 0, 0, 0, 0, 0, 0,
/*112 - ??? */ 0, 0, 0, 0, 0, 0,
/*113 - ??? */ 0, 0, 0, 0, 0, 0,
/*114 - ??? */ 0, 0, 0, 0, 0, 0,
/*115 - ??? */ 0, 0, 0, 0, 0, 0,
/*116 - ??? */ 0, 0, 0, 0, 0, 0,
/*117 - ??? */ 0, 0, 0, 0, 0, 0,
/*118 - ??? */ 0, 0, 0, 0, 0, 0,
/*119 - ??? */ 0, 0, 0, 0, 0, 0,
/*120 - ??? */ 0, 0, 0, 0, 0, 0,
/*121 - ??? */ 0, 0, 0, 0, 0, 0,
/*122 - ??? */ 0, 0, 0, 0, 0, 0,
/*123 - ??? */ 0, 0, 0, 0, 0, 0,
/*124 - ??? */ 0, 0, 0, 0, 0, 0,
/*125 - ??? */ 0, 0, 0, 0, 0, 0,
/*126 - ??? */ 0, 0, 0, 0, 0, 0,
/*127 - ??? */ 0, 0, 0, 0, 0, 0
};

```

```
/*      genmap - output binary keymap                        Author: Marcus Hampel
 */
#include <sys/types.h>
#include <minix/keymap.h>
#include <fcntl.h>
#include <unistd.h>
#include <stdlib.h>
#include <string.h>
#include <errno.h>

#include KEYSRC

u8_t comprmap[4 + NR_SCAN_CODES * MAP_COLS * 9/8 * 2 + 1];

void tell(const char *s)
{
    write(2, s, strlen(s));
}

int main(void)
{
    u8_t *cm, *fb;
    ul6_t *km;
    int n;

    /* Compress the keymap. */
    memcpy(comprmap, KEY_MAGIC, 4);
    cm = comprmap + 4;
    n = 8;
    for (km = keymap; km < keymap + NR_SCAN_CODES * MAP_COLS; km++) {
        if (n == 8) {
            /* Allocate a new flag byte. */
            fb = cm;
            *cm++ = 0;
            n = 0;
        }
        *cm++ = (*km & 0x00FF);          /* Low byte. */
        if (*km & 0xFF00) {
            *cm++ = (*km >> 8);        /* High byte only when set. */
            *fb |= (1 << n);          /* Set a flag if so. */
        }
        n++;
    }

    /* Don't store trailing zeros. */
    while (cm > comprmap && cm[-1] == 0) cm--;

    /* Emit the compressed keymap. */
    if (write(1, comprmap, cm - comprmap) < 0) {
        int err = errno;

        tell("genmap: ");
        tell(strerror(err));
        tell("\n");
        exit(1);
    }
    exit(0);
}
```

```
/* Keymap for German MF-2 keyboard. */
```

```
u16_t keymap[NR_SCAN_CODES * MAP_COLS] = {
```

```
/* scan-code      unsh      Shift      Alt      AltGr      Alt+Sh      Strg      */
/* ===== */
/* 00 - none      */      0,          0,          0,          0,          0,          0,
/* 01 - ESC       */      C('['), C('['), CA('['), C('['), C('['), C('['),
/* 02 - '1'       */      '1',      '!',      A('1'), '1',      '!',      C('A'),
/* 03 - '2'       */      '2',      '"',      A('2'), 0375, '@',      C('@'),
/* 04 - '3'       */      '3',          025,      A('3'), 0374, '#',      C('#'),
/* 05 - '4'       */      '4',      '$',      A('4'), '4',      '$',      C('D'),
/* 06 - '5'       */      '5',      '%',      A('5'), '5',      '%',      C('E'),
/* 07 - '6'       */      '6',      '&',      A('6'), '6',      '^',      C('^'),
/* 08 - '7'       */      '7',      '/',      A('7'), '{',      '&',      C('G'),
/* 09 - '8'       */      '8',      '(',      A('8'), '[',      '*',      C('H'),
/* 10 - '9'       */      '9',      ')',      A('9'), ']',      '(',      C('I'),
/* 11 - '0'       */      '0',      '=',      A('0'), '}',      ')',      C('@'),
/* 12 - '-'       */      0341,      '?',      0341,      '\\',      '_',      C('_'),
/* 13 - '='       */      '\\',      ',',      A('\\'), '=',      '+',      C('@'),
/* 14 - BS        */      C('H'), C('H'), CA('H'), C('H'), C('H'), 0177,
/* 15 - TAB       */      C('I'), C('I'), CA('I'), C('I'), C('I'), C('I'),
/* 16 - 'q'       */      L('q'), 'Q',      A('q'), '@',      'Q',      C('Q'),
/* 17 - 'w'       */      L('w'), 'W',      A('w'), 'w',      'W',      C('W'),
/* 18 - 'e'       */      L('e'), 'E',      A('e'), 'e',      'E',      C('E'),
/* 19 - 'r'       */      L('r'), 'R',      A('r'), 'r',      'R',      C('R'),
/* 20 - 't'       */      L('t'), 'T',      A('t'), 't',      'T',      C('T'),
/* 21 - 'y'       */      L('z'), 'Z',      A('z'), 'z',      'Z',      C('Z'),
/* 22 - 'u'       */      L('u'), 'U',      A('u'), 'u',      'U',      C('U'),
/* 23 - 'i'       */      L('i'), 'I',      A('i'), 'i',      'I',      C('I'),
/* 24 - 'o'       */      L('o'), 'O',      A('o'), 'o',      'O',      C('O'),
/* 25 - 'p'       */      L('p'), 'P',      A('p'), 'p',      'P',      C('P'),
/* 26 - '['       */      L(0201), 0232,      0201,      '[',      '{',      C('['),
/* 27 - ']'       */      '+',      '*',      A('+'), '~',      ']',      C(']'),
/* 28 - CR/LF     */      C('M'), C('M'), CA('M'), C('M'), C('M'), C('J'),
/* 29 - Strg;-)   */      CTRL,  CTRL,      CTRL,      CTRL,      CTRL,      CTRL,
/* 30 - 'a'       */      L('a'), 'A',      A('a'), 'a',      'A',      C('A'),
/* 31 - 's'       */      L('s'), 'S',      A('s'), 's',      'S',      C('S'),
/* 32 - 'd'       */      L('d'), 'D',      A('d'), 'd',      'D',      C('D'),
/* 33 - 'f'       */      L('f'), 'F',      A('f'), 'f',      'F',      C('F'),
/* 34 - 'g'       */      L('g'), 'G',      A('g'), 'g',      'G',      C('G'),
/* 35 - 'h'       */      L('h'), 'H',      A('h'), 'h',      'H',      C('H'),
/* 36 - 'j'       */      L('j'), 'J',      A('j'), 'j',      'J',      C('J'),
/* 37 - 'k'       */      L('k'), 'K',      A('k'), 'k',      'K',      C('K'),
/* 38 - 'l'       */      L('l'), 'L',      A('l'), 'l',      'L',      C('L'),
/* 39 - ';'       */      L(0224), 0231,      0224,      ';',      ':',      C('@'),
/* 40 - '\\'      */      L(0204), 0216,      0204,      '\\',      '"',      C('@'),
/* 41 - '^'       */      '^',          0370,      A('^'), '^',      '~',      C('^'),
/* 42 - SHIFT     */      SHIFT,  SHIFT,      SHIFT,      SHIFT,      SHIFT,      SHIFT,
/* 43 - '\\\\'    */      '#',      '\\',      A('#'), '\\\\', '|',      C('\\\\'),
/* 44 - 'z'       */      L('y'), 'Y',      A('y'), 'y',      'Y',      C('Y'),
/* 45 - 'x'       */      L('x'), 'X',      A('x'), 'x',      'X',      C('X'),
/* 46 - 'c'       */      L('c'), 'C',      A('c'), 'c',      'C',      C('C'),
/* 47 - 'v'       */      L('v'), 'V',      A('v'), 'v',      'V',      C('V'),
/* 48 - 'b'       */      L('b'), 'B',      A('b'), 'b',      'B',      C('B'),
/* 49 - 'n'       */      L('n'), 'N',      A('n'), 'n',      'N',      C('N'),
/* 50 - 'm'       */      L('m'), 'M',      A('m'), 0346,      'M',      C('M'),
/* 51 - '<'       */      '<',      '<',      A('<'), '<',      '<',      C('@'),
/* 52 - '>'       */      '>',      '>',      A('>'), '>',      '>',      C('@'),
/* 53 - '/'       */      '-',      '-',      A('-'), '/',      '?',      C('_'),
/* 54 - SHIFT     */      SHIFT,  SHIFT,      SHIFT,      SHIFT,      SHIFT,      SHIFT,
/* 55 - '*'       */      '*',      '*',      A('*'), '*',      '*',      C('@'),
/* 56 - ALT       */      ALT,      ALT,      ALT,      ALT,      ALT,      ALT,
/* 57 - ' '       */      ' ',      ' ',      A(' '), ' ',      ' ',      C('@'),
/* 58 - CapsLck   */      CALOCK,  CALOCK,      CALOCK,      CALOCK,      CALOCK,      CALOCK,
/* 59 - F1        */      F1,      SF1,      AF1,      AF1,      ASF1,      CF1,
/* 60 - F2        */      F2,      SF2,      AF2,      AF2,      ASF2,      CF2,
/* 61 - F3        */      F3,      SF3,      AF3,      AF3,      ASF3,      CF3,
/* 62 - F4        */      F4,      SF4,      AF4,      AF4,      ASF4,      CF4,
/* 63 - F5        */      F5,      SF5,      AF5,      AF5,      ASF5,      CF5,
/* 64 - F6        */      F6,      SF6,      AF6,      AF6,      ASF6,      CF6,
/* 65 - F7        */      F7,      SF7,      AF7,      AF7,      ASF7,      CF7,
/* 66 - F8        */      F8,      SF8,      AF8,      AF8,      ASF8,      CF8,
/* 67 - F9        */      F9,      SF9,      AF9,      AF9,      ASF9,      CF9,
```

```

/* 68 - F10 */ F10, SF10, AF10, AF10, ASF10, CF10,
/* 69 - NumLock */ NLOCK, NLOCK, NLOCK, NLOCK, NLOCK, NLOCK,
/* 70 - ScrLock */ SLOCK, SLOCK, SLOCK, SLOCK, SLOCK, SLOCK,
/* 71 - Home */ HOME, '7', AHOME, AHOME, '7', CHOME,
/* 72 - CurUp */ UP, '8', AUP, AUP, '8', CUP,
/* 73 - PgUp */ PGUP, '9', APGUP, APGUP, '9', CPGUP,
/* 74 - '-' */ NMIN, '-', ANMIN, ANMIN, '-', CNMIN,
/* 75 - Left */ LEFT, '4', ALEFT, ALEFT, '4', CLEFT,
/* 76 - MID */ MID, '5', AMID, AMID, '5', CMID,
/* 77 - Right */ RIGHT, '6', ARIGHT, ARIGHT, '6', CRIGHT,
/* 78 - '+' */ PLUS, '+', APLUS, APLUS, '+', CPLUS,
/* 79 - End */ END, '1', AEND, AEND, '1', CEND,
/* 80 - Down */ DOWN, '2', ADOWN, ADOWN, '2', CDOWN,
/* 81 - PgDown */ PGDN, '3', APGDN, APGDN, '3', CPGDN,
/* 82 - Insert */ INSRT, '0', AINSRT, AINSRT, '0', CINSRT,
/* 83 - Delete */ 0177, '.', A(0177), 0177, '.', 0177,
/* 84 - Enter */ C('M'), C('M'), CA('M'), C('M'), C('M'), C('J'),
/* 85 - ??? */ 0, 0, 0, 0, 0, 0,
/* 86 - ??? */ '<', '>', A('<'), '|', '>', C('@'),
/* 87 - F11 */ F11, SF11, AF11, AF11, ASF11, CF11,
/* 88 - F12 */ F12, SF12, AF12, AF12, ASF12, CF12,
/* 89 - ??? */ 0, 0, 0, 0, 0, 0,
/* 90 - ??? */ 0, 0, 0, 0, 0, 0,
/* 91 - ??? */ 0, 0, 0, 0, 0, 0,
/* 92 - ??? */ 0, 0, 0, 0, 0, 0,
/* 93 - ??? */ 0, 0, 0, 0, 0, 0,
/* 94 - ??? */ 0, 0, 0, 0, 0, 0,
/* 95 - ??? */ 0, 0, 0, 0, 0, 0,
/* 96 - EXT_KEY */ EXTKEY, EXTKEY, EXTKEY, EXTKEY, EXTKEY, EXTKEY,
/* 97 - ??? */ 0, 0, 0, 0, 0, 0,
/* 98 - ??? */ 0, 0, 0, 0, 0, 0,
/* 99 - ??? */ 0, 0, 0, 0, 0, 0,
/*100 - ??? */ 0, 0, 0, 0, 0, 0,
/*101 - ??? */ 0, 0, 0, 0, 0, 0,
/*102 - ??? */ 0, 0, 0, 0, 0, 0,
/*103 - ??? */ 0, 0, 0, 0, 0, 0,
/*104 - ??? */ 0, 0, 0, 0, 0, 0,
/*105 - ??? */ 0, 0, 0, 0, 0, 0,
/*106 - ??? */ 0, 0, 0, 0, 0, 0,
/*107 - ??? */ 0, 0, 0, 0, 0, 0,
/*108 - ??? */ 0, 0, 0, 0, 0, 0,
/*109 - ??? */ 0, 0, 0, 0, 0, 0,
/*110 - ??? */ 0, 0, 0, 0, 0, 0,
/*111 - ??? */ 0, 0, 0, 0, 0, 0,
/*112 - ??? */ 0, 0, 0, 0, 0, 0,
/*113 - ??? */ 0, 0, 0, 0, 0, 0,
/*114 - ??? */ 0, 0, 0, 0, 0, 0,
/*115 - ??? */ 0, 0, 0, 0, 0, 0,
/*116 - ??? */ 0, 0, 0, 0, 0, 0,
/*117 - ??? */ 0, 0, 0, 0, 0, 0,
/*118 - ??? */ 0, 0, 0, 0, 0, 0,
/*119 - ??? */ 0, 0, 0, 0, 0, 0,
/*120 - ??? */ 0, 0, 0, 0, 0, 0,
/*121 - ??? */ 0, 0, 0, 0, 0, 0,
/*122 - ??? */ 0, 0, 0, 0, 0, 0,
/*123 - ??? */ 0, 0, 0, 0, 0, 0,
/*124 - ??? */ 0, 0, 0, 0, 0, 0,
/*125 - ??? */ 0, 0, 0, 0, 0, 0,
/*126 - ??? */ 0, 0, 0, 0, 0, 0,
/*127 - ??? */ 0, 0, 0, 0, 0, 0
};

```



```

/* Keymap for Italian keyboard. */
/* Modified by Ernesto Del Prete in October 1997 */
/* ernesto@ccclix1.polito.it or s84508@ccclix1.polito.it */

u16_t keymap[NR_SCAN_CODES * MAP_COLS] = {

/* scan-code      !Shift  Shift  Alt      AltGr  Alt+Sh  Ctrl      */
/* ===== */
/* 00 - none      */ 0,      0,      0,      0,      0,      0,
/* 01 - ESC        */ C('[',), C('[',), CA('[',), C('[',), C('[',), C('[',),
/* 02 - '1'        */ '1',    '!',    A('1'), '1',    '!',    C('A'),
/* 03 - '2'        */ '2',    '"',    A('2'), '2',    '@',    C('@'),
/* 04 - '3'        */ '3',    0234,  A('3'), '3',    0234,  C('C'),
/* 05 - '4'        */ '4',    '$',    A('4'), '4',    '$',    C('D'),
/* 06 - '5'        */ '5',    '%',    A('5'), '5',    '%',    C('E'),
/* 07 - '6'        */ '6',    '&',    A('6'), '6',    '&',    C('F'),
/* 08 - '7'        */ '7',    '/',    A('7'), '7',    '/',    C('G'),
/* 09 - '8'        */ '8',    '(',    A('8'), '8',    '(',    C('H'),
/* 10 - '9'        */ '9',    ')',    A('9'), '8',    ')',    C('I'),
/* 11 - '0'        */ '0',    '=',    A('0'), '0',    '=',    C('@'),
/* 12 - '-'        */ '\',    '?',    A('\'), '\',    '?',    C('@'),
/* 13 - '='        */ 0215,  '^',    A('^'), 0215,  '^',    C('^'),
/* 14 - BS         */ C('H'), C('H'), CA('H'), C('H'), C('H'), 0177,
/* 15 - TAB        */ C('I'), C('I'), CA('I'), C('I'), C('I'), C('I'),
/* 16 - 'q'        */ L('q'), 'Q',    A('q'), 'q',    'Q',    C('Q'),
/* 17 - 'w'        */ L('w'), 'W',    A('w'), 'w',    'W',    C('W'),
/* 18 - 'e'        */ L('e'), 'E',    A('e'), 'e',    'E',    C('E'),
/* 19 - 'r'        */ L('r'), 'R',    A('r'), 'r',    'R',    C('R'),
/* 20 - 't'        */ L('t'), 'T',    A('t'), 't',    'T',    C('T'),
/* 21 - 'y'        */ L('y'), 'Y',    A('y'), 'y',    'Y',    C('Y'),
/* 22 - 'u'        */ L('u'), 'U',    A('u'), 'u',    'U',    C('U'),
/* 23 - 'i'        */ L('i'), 'I',    A('i'), 'i',    'I',    C('I'),
/* 24 - 'o'        */ L('o'), 'O',    A('o'), 'o',    'O',    C('O'),
/* 25 - 'p'        */ L('p'), 'P',    A('p'), 'p',    'P',    C('P'),
/* 26 - '['        */ 0212,  0202,  0212,  '[',    '{',    C('{'),
/* 27 - ']'        */ '+',    '*',    A('+'), ']',    '}',    C(']'),
/* 28 - CR/LF      */ C('M'), C('M'), CA('M'), C('M'), C('M'), C('J'),
/* 29 - Ctrl       */ CTRL,  CTRL,  CTRL,  CTRL,  CTRL,  CTRL,
/* 30 - 'a'        */ L('a'), 'A',    A('a'), 'a',    'A',    C('A'),
/* 31 - 's'        */ L('s'), 'S',    A('s'), 's',    'S',    C('S'),
/* 32 - 'd'        */ L('d'), 'D',    A('d'), 'd',    'D',    C('D'),
/* 33 - 'f'        */ L('f'), 'F',    A('f'), 'f',    'F',    C('F'),
/* 34 - 'g'        */ L('g'), 'G',    A('g'), 'g',    'G',    C('G'),
/* 35 - 'h'        */ L('h'), 'H',    A('h'), 'h',    'H',    C('H'),
/* 36 - 'j'        */ L('j'), 'J',    A('j'), 'j',    'J',    C('J'),
/* 37 - 'k'        */ L('k'), 'K',    A('k'), 'k',    'K',    C('K'),
/* 38 - 'l'        */ L('l'), 'L',    A('l'), 'l',    'L',    C('L'),
/* 39 - ';'        */ 0225,  0207,  0225,  '@',    '@',    C('@'),
/* 40 - '\'        */ 0205,  0370,  0205,  '#',    '#',    C('@'),
/* 41 - '`'        */ '\\',  '|',    A('\\'), 0176,  '|',    C('\\'),
/* 42 - l. SHIFT*  */ SHIFT, SHIFT, SHIFT, SHIFT, SHIFT, SHIFT,
/* 43 - '\\\'      */ 0227,  025,    0227,  0227,  025,    C('@'),
/* 44 - 'z'        */ L('z'), 'Z',    A('z'), 'z',    'Z',    C('Z'),
/* 45 - 'x'        */ L('x'), 'X',    A('x'), 'x',    'X',    C('X'),
/* 46 - 'c'        */ L('c'), 'C',    A('c'), 'c',    'C',    C('C'),
/* 47 - 'v'        */ L('v'), 'V',    A('v'), 'v',    'V',    C('V'),
/* 48 - 'b'        */ L('b'), 'B',    A('b'), 'b',    'B',    C('B'),
/* 49 - 'n'        */ L('n'), 'N',    A('n'), 'n',    'N',    C('N'),
/* 50 - 'm'        */ L('m'), 'M',    A('m'), 'm',    'M',    C('M'),
/* 51 - ','        */ ',',    ';',    A(','), ',',    ';',    C('@'),
/* 52 - '.'        */ '.',    ':',    A('.'), '.',    ':',    C('@'),
/* 53 - '/'        */ '-',    '_',    A('-'), '-',    '_',    C('_'),
/* 54 - r. SHIFT*  */ SHIFT, SHIFT, SHIFT, SHIFT, SHIFT, SHIFT,
/* 55 - '*'        */ '*',    '*',    A('*'), '*',    '*',    C('M'),
/* 56 - ALT        */ ALT,    ALT,    ALT,    ALT,    ALT,    ALT,
/* 57 - ' '        */ ' ',    ' ',    A(' '), ' ',    ' ',    C('@'),
/* 58 - CapsLck    */ CALOCK, CALOCK, CALOCK, CALOCK, CALOCK, CALOCK,
/* 59 - F1         */ F1,    SF1,    AF1,    AF1,    ASF1,  CF1,
/* 60 - F2         */ F2,    SF2,    AF2,    AF2,    ASF2,  CF2,
/* 61 - F3         */ F3,    SF3,    AF3,    AF3,    ASF3,  CF3,
/* 62 - F4         */ F4,    SF4,    AF4,    AF4,    ASF4,  CF4,
/* 63 - F5         */ F5,    SF5,    AF5,    AF5,    ASF5,  CF5,
/* 64 - F6         */ F6,    SF6,    AF6,    AF6,    ASF6,  CF6,
/* 65 - F7         */ F7,    SF7,    AF7,    AF7,    ASF7,  CF7,

```

```

/* 66 - F8      */      F8,      SF8,      AF8,      AF8,      ASF8,      CF8,
/* 67 - F9      */      F9,      SF9,      AF9,      AF9,      ASF9,      CF9,
/* 68 - F10     */      F10,     SF10,     AF10,     AF10,     ASF10,     CF10,
/* 69 - NumLock */      NLOCK,    NLOCK,    NLOCK,    NLOCK,    NLOCK,    NLOCK,
/* 70 - ScrLock */      SLOCK,    SLOCK,    SLOCK,    SLOCK,    SLOCK,    SLOCK,
/* 71 - Home    */      HOME,     '7',     AHOME,     AHOME,     '7',     CHOME,
/* 72 - CurUp   */      UP,       '8',     AUP,       AUP,       '8',     CUP,
/* 73 - PgUp    */      PGUP,     '9',     APGUP,    APGUP,    '9',     CPGUP,
/* 74 - '-'     */      NMIN,     '-',     ANMIN,    ANMIN,    '-',     CNMIN,
/* 75 - Left    */      LEFT,     '4',     ALEFT,    ALEFT,    '4',     CLEFT,
/* 76 - MID     */      MID,      '5',     AMID,     AMID,     '5',     CMID,
/* 77 - Right   */      RIGHT,    '6',     ARIGHT,   ARIGHT,   '6',     CRIGHT,
/* 78 - '+'     */      PLUS,     '+',     APLUS,    APLUS,    '+',     CPLUS,
/* 79 - End     */      END,      '1',     AEND,     AEND,     '1',     CEND,
/* 80 - Down    */      DOWN,     '2',     ADOWN,    ADOWN,    '2',     CDOWN,
/* 81 - PgDown  */      PGDN,     '3',     APGDN,    APGDN,    '3',     CPGDN,
/* 82 - Insert  */      INSRT,    '0',     AINSRT,   AINSRT,   '0',     CINSRT,
/* 83 - Delete  */      0177,    '.',    A(0177), 0177,    '.',    0177,
/* 84 - Enter   */      C('M'),  C('M'),  CA('M'),  C('M'),  C('M'),  C('J'),
/* 85 - ???     */      0,       0,       0,       0,       0,       0,
/* 86 - ???     */      '<',     '>',     A('<'),  '|',     '>',     C('@'),
/* 87 - F11     */      F11,     SF11,    AF11,     AF11,     ASF11,    CF11,
/* 88 - F12     */      F12,     SF12,    AF12,     AF12,     ASF12,    CF12,
/* 89 - ???     */      0,       0,       0,       0,       0,       0,
/* 90 - ???     */      0,       0,       0,       0,       0,       0,
/* 91 - ???     */      0,       0,       0,       0,       0,       0,
/* 92 - ???     */      0,       0,       0,       0,       0,       0,
/* 93 - ???     */      0,       0,       0,       0,       0,       0,
/* 94 - ???     */      0,       0,       0,       0,       0,       0,
/* 95 - ???     */      0,       0,       0,       0,       0,       0,
/* 96 - EXT_KEY */      EXTKEY,  EXTKEY,  EXTKEY,  EXTKEY,  EXTKEY,  EXTKEY,
/* 97 - ???     */      0,       0,       0,       0,       0,       0,
/* 98 - ???     */      0,       0,       0,       0,       0,       0,
/* 99 - ???     */      0,       0,       0,       0,       0,       0,
/*100 - ???     */      0,       0,       0,       0,       0,       0,
/*101 - ???     */      0,       0,       0,       0,       0,       0,
/*102 - ???     */      0,       0,       0,       0,       0,       0,
/*103 - ???     */      0,       0,       0,       0,       0,       0,
/*104 - ???     */      0,       0,       0,       0,       0,       0,
/*105 - ???     */      0,       0,       0,       0,       0,       0,
/*106 - ???     */      0,       0,       0,       0,       0,       0,
/*107 - ???     */      0,       0,       0,       0,       0,       0,
/*108 - ???     */      0,       0,       0,       0,       0,       0,
/*109 - ???     */      0,       0,       0,       0,       0,       0,
/*110 - ???     */      0,       0,       0,       0,       0,       0,
/*111 - ???     */      0,       0,       0,       0,       0,       0,
/*112 - ???     */      0,       0,       0,       0,       0,       0,
/*113 - ???     */      0,       0,       0,       0,       0,       0,
/*114 - ???     */      0,       0,       0,       0,       0,       0,
/*115 - ???     */      0,       0,       0,       0,       0,       0,
/*116 - ???     */      0,       0,       0,       0,       0,       0,
/*117 - ???     */      0,       0,       0,       0,       0,       0,
/*118 - ???     */      0,       0,       0,       0,       0,       0,
/*119 - ???     */      0,       0,       0,       0,       0,       0,
/*120 - ???     */      0,       0,       0,       0,       0,       0,
/*121 - ???     */      0,       0,       0,       0,       0,       0,
/*122 - ???     */      0,       0,       0,       0,       0,       0,
/*123 - ???     */      0,       0,       0,       0,       0,       0,
/*124 - ???     */      0,       0,       0,       0,       0,       0,
/*125 - ???     */      0,       0,       0,       0,       0,       0,
/*126 - ???     */      0,       0,       0,       0,       0,       0,
/*127 - ???     */      0,       0,       0,       0,       0,       0
};

```

```

/*
 *      Keymap for Japanese 106 keyboard.
 *      Dec. 31 1995
 *      Toshiya Ogawa    <ogw@shizuokanet.or.jp>
 *                      <GCD02425@niftyserve.or.jp>
 */

/*
 * Japanese 106 keyboard has following additional 5 scan codes
 * compared to US standard 101 keyboard.
 */
/*      scan-code      keytop      effect in this keymap
 *      -----
 *      112(0x70)      KANA          (ignored)
 *      115(0x73)      BackSlash     mapped to '\\' and '_'
 *      121(0x79)      HENKAN        (ignored)
 *      123(0x7B)      MU-HENKAN     (ignored)
 *      125(0x7D)      YEN           mapped to '\\' and '|'
 */

#if (NR_SCAN_CODES != 0x80)
#error NR_SCAN_CODES mis-match
#endif

u16_t keymap[NR_SCAN_CODES * MAP_COLS] = {

/* scan-code      !Shift  Shift  Alt1    Alt2    Alt+Sh  Ctrl    */
/* =====
/* 00 - none      */      0,      0,      0,      0,      0,      0,
/* 01 - ESC       */      C('['), C('['), CA('['), CA('['), CA('['), C('['),
/* 02 - '1'       */      '1',    '!',    A('1'), A('1'), A('!'), C('A'),
/* 03 - '2'       */      '2',    '"',    A('2'), A('2'), A('"'), C('B'),
/* 04 - '3'       */      '3',    '#',    A('3'), A('3'), A('#'), C('C'),
/* 05 - '4'       */      '4',    '$',    A('4'), A('4'), A('$'), C('D'),
/* 06 - '5'       */      '5',    '%',    A('5'), A('5'), A('%'), C('E'),
/* 07 - '6'       */      '6',    '&',    A('6'), A('6'), A('&'), C('F'),
/* 08 - '7'       */      '7',    '\',    A('7'), A('7'), A('\'), C('G'),
/* 09 - '8'       */      '8',    '(',    A('8'), A('8'), A('('), C('H'),
/* 10 - '9'       */      '9',    ')',    A('9'), A('9'), A(')'), C('I'),
/* 11 - '0'       */      '0',    '~',    A('0'), A('0'), A('~'), C('@'),
/* 12 - '-'       */      '-',    '=',    A('-'), A('-'), A('='), C('@'),
/* 13 - '^'       */      '^',    '~',    A('^'), A('^'), A('~'), C('^'),
/* 14 - BS        */      C('H'), C('H'), CA('H'), CA('H'), CA('H'), 0177,
/* 15 - TAB       */      C('I'), C('I'), CA('I'), CA('I'), CA('I'), C('I'),
/* 16 - 'q'       */      L('q'), 'Q',    A('q'), A('q'), A('Q'), C('Q'),
/* 17 - 'w'       */      L('w'), 'W',    A('w'), A('w'), A('W'), C('W'),
/* 18 - 'e'       */      L('e'), 'E',    A('e'), A('e'), A('E'), C('E'),
/* 19 - 'r'       */      L('r'), 'R',    A('r'), A('r'), A('R'), C('R'),
/* 20 - 't'       */      L('t'), 'T',    A('t'), A('t'), A('T'), C('T'),
/* 21 - 'y'       */      L('y'), 'Y',    A('y'), A('y'), A('Y'), C('Y'),
/* 22 - 'u'       */      L('u'), 'U',    A('u'), A('u'), A('U'), C('U'),
/* 23 - 'i'       */      L('i'), 'I',    A('i'), A('i'), A('I'), C('I'),
/* 24 - 'o'       */      L('o'), 'O',    A('o'), A('o'), A('O'), C('O'),
/* 25 - 'p'       */      L('p'), 'P',    A('p'), A('p'), A('P'), C('P'),
/* 26 - '@'       */      '@',    '',    A('@'), A('@'), A(''), C('@'),
/* 27 - '['       */      '[',    '{',    A('['), A('['), A('{'), C('['),
/* 28 - Enter     */      C('M'), C('M'), CA('M'), CA('M'), CA('M'), C('J'),
/* 29 - Ctrl      */      CTRL,  CTRL,  CTRL,  CTRL,  CTRL,  CTRL,
/* 30 - 'a'       */      L('a'), 'A',    A('a'), A('a'), A('A'), C('A'),
/* 31 - 's'       */      L('s'), 'S',    A('s'), A('s'), A('S'), C('S'),
/* 32 - 'd'       */      L('d'), 'D',    A('d'), A('d'), A('D'), C('D'),
/* 33 - 'f'       */      L('f'), 'F',    A('f'), A('f'), A('F'), C('F'),
/* 34 - 'g'       */      L('g'), 'G',    A('g'), A('g'), A('G'), C('G'),
/* 35 - 'h'       */      L('h'), 'H',    A('h'), A('h'), A('H'), C('H'),
/* 36 - 'j'       */      L('j'), 'J',    A('j'), A('j'), A('J'), C('J'),
/* 37 - 'k'       */      L('k'), 'K',    A('k'), A('k'), A('K'), C('K'),
/* 38 - 'l'       */      L('l'), 'L',    A('l'), A('l'), A('L'), C('L'),
/* 39 - ';'       */      ';',    '+',    A(';'), A(';'), A('+'), C('@'),
/* 40 - ':'       */      ':',    '*',    A(':'), A(':'), A('*'), C('@'),
/* 41 - KANJI     */      0,      0,      0,      0,      0,      0,
/* 42 - l. SHIFT  */      SHIFT, SHIFT, SHIFT, SHIFT, SHIFT, SHIFT,
/* 43 - ']'       */      ']',    '}',    A(']'), A(']'), A('}'), C(']'),
/* 44 - 'z'       */      L('z'), 'Z',    A('z'), A('z'), A('Z'), C('Z'),
/* 45 - 'x'       */      L('x'), 'X',    A('x'), A('x'), A('X'), C('X'),

```

```

/* 46 - 'c' */ L('c'), 'C', A('c'), A('c'), A('C'), C('C'),
/* 47 - 'v' */ L('v'), 'V', A('v'), A('v'), A('V'), C('V'),
/* 48 - 'b' */ L('b'), 'B', A('b'), A('b'), A('B'), C('B'),
/* 49 - 'n' */ L('n'), 'N', A('n'), A('n'), A('N'), C('N'),
/* 50 - 'm' */ L('m'), 'M', A('m'), A('m'), A('M'), C('M'),
/* 51 - ',' */ '',' '<', A(','), A(','), A('<'), C('@'),
/* 52 - '.' */ '',' '>', A('.'), A('.'), A('>'), C('@'),
/* 53 - '/' */ '',' '?', A('/'), A('/'), A('?'), C('@'),
/* 54 - r. SHIFT*/ SHIFT, SHIFT, SHIFT, SHIFT, SHIFT, SHIFT,
/* 55 - '*' */ '',' '*', A('*'), A('*'), A('*'), C('@'),
/* 56 - ALT */ ALT, ALT, ALT, ALT, ALT, ALT,
/* 57 - ' ' */ '',' 'A(' '), A(' '), A(' '), C('@'),
/* 58 - CapsLock */ CALOCK, CALOCK, CALOCK, CALOCK, CALOCK, CALOCK,
/* 59 - F1 */ F1, SF1, AF1, AF1, ASF1, CF1,
/* 60 - F2 */ F2, SF2, AF2, AF2, ASF2, CF2,
/* 61 - F3 */ F3, SF3, AF3, AF3, ASF3, CF3,
/* 62 - F4 */ F4, SF4, AF4, AF4, ASF4, CF4,
/* 63 - F5 */ F5, SF5, AF5, AF5, ASF5, CF5,
/* 64 - F6 */ F6, SF6, AF6, AF6, ASF6, CF6,
/* 65 - F7 */ F7, SF7, AF7, AF7, ASF7, CF7,
/* 66 - F8 */ F8, SF8, AF8, AF8, ASF8, CF8,
/* 67 - F9 */ F9, SF9, AF9, AF9, ASF9, CF9,
/* 68 - F10 */ F10, SF10, AF10, AF10, ASF10, CF10,
/* 69 - NumLock */ NLOCK, NLOCK, NLOCK, NLOCK, NLOCK, NLOCK,
/* 70 - ScrLock */ SLOCK, SLOCK, SLOCK, SLOCK, SLOCK, SLOCK,
/* 71 - Home */ HOME, '7', AHOME, AHOME, A('7'), CHOME,
/* 72 - CurUp */ UP, '8', AUP, AUP, A('8'), CUP,
/* 73 - PgUp */ PGUP, '9', APGUP, APGUP, A('9'), CPGUP,
/* 74 - '-' */ NMIN, '- ', ANMIN, ANMIN, A('- '), CNMIN,
/* 75 - Left */ LEFT, '4', ALEFT, ALEFT, A('4'), CLEFT,
/* 76 - MID */ MID, '5', AMID, AMID, A('5'), CMID,
/* 77 - Right */ RIGHT, '6', ARIGHT, ARIGHT, A('6'), CRIGHT,
/* 78 - '+' */ PLUS, '+', APLUS, APLUS, A('+'), CPLUS,
/* 79 - End */ END, '1', AEND, AEND, A('1'), CEND,
/* 80 - Down */ DOWN, '2', ADOWN, ADOWN, A('2'), CDOWN,
/* 81 - PgDown */ PGDN, '3', APGDN, APGDN, A('3'), CPGDN,
/* 82 - Insert */ INSRT, '0', AINSRT, AINSRT, A('0'), CINSRT,
/* 83 - Delete */ 0177, '.', A(0177), A(0177), A('.'), 0177,
/* 84 - Enter */ C('M'), C('M'), CA('M'), CA('M'), CA('M'), C('J'),
/* 85 - ??? */ 0, 0, 0, 0, 0, 0,
/* 86 - ??? */ 0, 0, 0, 0, 0, 0,
/* 87 - F11 */ F11, SF11, AF11, AF11, ASF11, CF11,
/* 88 - F12 */ F12, SF12, AF12, AF12, ASF12, CF12,
/* 89 - ??? */ 0, 0, 0, 0, 0, 0,
/* 90 - ??? */ 0, 0, 0, 0, 0, 0,
/* 91 - ??? */ 0, 0, 0, 0, 0, 0,
/* 92 - ??? */ 0, 0, 0, 0, 0, 0,
/* 93 - ??? */ 0, 0, 0, 0, 0, 0,
/* 94 - ??? */ 0, 0, 0, 0, 0, 0,
/* 95 - ??? */ 0, 0, 0, 0, 0, 0,
/* 96 - EXT_KEY */ EXTKEY, EXTKEY, EXTKEY, EXTKEY, EXTKEY, EXTKEY,
/* 97 - ??? */ 0, 0, 0, 0, 0, 0,
/* 98 - ??? */ 0, 0, 0, 0, 0, 0,
/* 99 - ??? */ 0, 0, 0, 0, 0, 0,
/*100 - ??? */ 0, 0, 0, 0, 0, 0,
/*101 - ??? */ 0, 0, 0, 0, 0, 0,
/*102 - ??? */ 0, 0, 0, 0, 0, 0,
/*103 - ??? */ 0, 0, 0, 0, 0, 0,
/*104 - ??? */ 0, 0, 0, 0, 0, 0,
/*105 - ??? */ 0, 0, 0, 0, 0, 0,
/*106 - ??? */ 0, 0, 0, 0, 0, 0,
/*107 - ??? */ 0, 0, 0, 0, 0, 0,
/*108 - ??? */ 0, 0, 0, 0, 0, 0,
/*109 - ??? */ 0, 0, 0, 0, 0, 0,
/*110 - ??? */ 0, 0, 0, 0, 0, 0,
/*111 - ??? */ 0, 0, 0, 0, 0, 0,
/*112 - KANA */ 0, 0, 0, 0, 0, 0,
/*113 - ??? */ 0, 0, 0, 0, 0, 0,
/*114 - ??? */ 0, 0, 0, 0, 0, 0,
/*115 - '\\ ' */ '\\ ', '_ ', A('\\ '), A('\\ '), A('_ '), C('_ '),
/*116 - ??? */ 0, 0, 0, 0, 0, 0,
/*117 - ??? */ 0, 0, 0, 0, 0, 0,
/*118 - ??? */ 0, 0, 0, 0, 0, 0,
/*119 - ??? */ 0, 0, 0, 0, 0, 0,

```

```
/*120 - ??? */ 0, 0, 0, 0, 0, 0,
/*121 - HENKAN */ 0, 0, 0, 0, 0, 0,
/*122 - ??? */ 0, 0, 0, 0, 0, 0,
/*123 - MU-HENKAN*/ 0, 0, 0, 0, 0, 0,
/*124 - ??? */ 0, 0, 0, 0, 0, 0,
/*125 - YEN */ '\'', '|', A('\'''), A('\'''), A(' | '), C('\'''),
/*126 - ??? */ 0, 0, 0, 0, 0, 0,
/*127 - ??? */ 0, 0, 0, 0, 0, 0
};
```

```
/**
 * Keymap for Latin American keyboard. v1.02
 * Victor A. Rodriguez - El bit Fantasma - Bit-Man@Tasa.Com.AR
 *
 * The Latin American keyboard makes differences between the left and
 * right ALT keys (the right one is so called ALT GR), and uses accent.
 *
 * Release History
 * =====
 * v1.00      Initial version
 * v1.01      Extended ASCII characters replaced by hex. equivalents
 * v1.02      NR_SCAN_CODES has grown to 0x80, required by Toshiya Ogawa
 *            (ogw@shizuokanet.or.jp) and added by Kees J.Bot (kjb@cs.vu.nl)
 *            in MINIX 1.7.2
 */

#if (NR_SCAN_CODES != 0x80)
#error NR_SCAN_CODES mis-match
#endif

u16_t keymap[NR_SCAN_CODES * MAP_COLS] = {
/* scan-code      !Shift  Shift    Alt1     Alt2     Alt+Sh  Ctrl      */
/* ===== */
/* 00 - none      */ 0,        0,        0,        0,        0,        0,
/* 01 - ESC       */ C('['), C('['), CA('['), CA('['), CA('['), C('['),
/* 02 - '1'       */ '1',    '!',    A('1'), A('1'), A('!'), C('A'),
/* 03 - '2'       */ '2',    '"',    A('2'), A('2'), A('"'), C('@'),
/* 04 - '3'       */ '3',    '#',    A('3'), A('3'), A('#'), C('C'),
/* 05 - '4'       */ '4',    '$',    A('4'), A('4'), A('$'), C('D'),
/* 06 - '5'       */ '5',    '%',    A('5'), A('5'), A('%'), C('E'),
/* 07 - '6'       */ '6',    '&',    A('6'), A('6'), A('&'), C('^'),
/* 08 - '7'       */ '7',    '/',    A('7'), A('7'), A('/'), C('G'),
/* 09 - '8'       */ '8',    '(',    A('8'), A('8'), A('('), C('H'),
/* 10 - '9'       */ '9',    ')',    A('9'), A('9'), A(')'), C('I'),
/* 11 - '0'       */ '0',    '=',    A('0'), A('0'), A('='), C('@'),
/* 12 - '-'       */ '\'',    '?',    A('\''), 0x5c, A('?'), C('?'),
/* 13 - '`'       */ 0xa8,    0xad,    A(0xa8), A(0xa8), A(0xad), C('@'),
/* 14 - BS        */ C('H'), C('H'), CA('H'), CA('H'), CA('H'), 0177,
/* 15 - TAB       */ C('I'), C('I'), CA('I'), CA('I'), CA('I'), C('I'),
/* 16 - 'q'       */ L('q'), 'Q',    A('q'), 0x40, A('Q'), C('Q'),
/* 17 - 'w'       */ L('w'), 'W',    A('w'), A('w'), A('W'), C('W'),
/* 18 - 'e'       */ L('e'), 'E',    A('e'), A('e'), A('E'), C('E'),
/* 19 - 'r'       */ L('r'), 'R',    A('r'), A('r'), A('R'), C('R'),
/* 20 - 't'       */ L('t'), 'T',    A('t'), A('t'), A('T'), C('T'),
/* 21 - 'y'       */ L('y'), 'Y',    A('y'), A('y'), A('Y'), C('Y'),
/* 22 - 'u'       */ L('u'), 'U',    A('u'), A('u'), A('U'), C('U'),
/* 23 - 'i'       */ L('i'), 'I',    A('i'), A('i'), A('I'), C('I'),
/* 24 - 'o'       */ L('o'), 'O',    A('o'), A('o'), A('O'), C('O'),
/* 25 - 'p'       */ L('p'), 'P',    A('p'), A('p'), A('P'), C('P'),
/* 26 - 'i'       */ 0xef,    0xf9,    A(0xef), A(0xef), A(0xf9), C(0xef),
/* 27 - '+'       */ '+',    '*',    A('+'), 0x7e, A('*'), C('+'),
/* 28 - CR/LF     */ C('M'), C('M'), CA('M'), CA('M'), CA('M'), C('J'),
/* 29 - Ctrl      */ CTRL, CTRL, CTRL, CTRL, CTRL, CTRL,
/* 30 - 'a'       */ L('a'), 'A',    A('a'), A('a'), A('A'), C('A'),
/* 31 - 's'       */ L('s'), 'S',    A('s'), A('s'), A('S'), C('S'),
/* 32 - 'd'       */ L('d'), 'D',    A('d'), A('d'), A('D'), C('D'),
/* 33 - 'f'       */ L('f'), 'F',    A('f'), A('f'), A('F'), C('F'),
/* 34 - 'g'       */ L('g'), 'G',    A('g'), A('g'), A('G'), C('G'),
/* 35 - 'h'       */ L('h'), 'H',    A('h'), A('h'), A('H'), C('H'),
/* 36 - 'j'       */ L('j'), 'J',    A('j'), A('j'), A('J'), C('J'),
/* 37 - 'k'       */ L('k'), 'K',    A('k'), A('k'), A('K'), C('K'),
/* 38 - 'l'       */ L('l'), 'L',    A('l'), A('l'), A('L'), C('L'),
/* 39 - 'n'       */ L(0xa4), 0xa5,    A(0xa4), A(0xa4), A(0xa5), C('@'),
/* 40 - '{'       */ '{',    '[',    A('{'), 0x5e, A('['), C('@'),
/* 41 - '|'       */ '|',    0xf8,    A('|'), 0xaa, A('\''), C('@'),
/* 42 - l. SHIFT */ SHIFT, SHIFT, SHIFT, SHIFT, SHIFT, SHIFT,
/* 43 - '}'       */ 0x7d, 0x5d,    A('<'), 0x60, A('>'), C('<'),
/* 44 - 'z'       */ L('z'), 'Z',    A('z'), A('z'), A('Z'), C('Z'),
/* 45 - 'x'       */ L('x'), 'X',    A('x'), A('x'), A('X'), C('X'),
/* 46 - 'c'       */ L('c'), 'C',    A('c'), A('c'), A('C'), C('C'),
/* 47 - 'v'       */ L('v'), 'V',    A('v'), A('v'), A('V'), C('V'),
/* 48 - 'b'       */ L('b'), 'B',    A('b'), A('b'), A('B'), C('B'),
/* 49 - 'n'       */ L('n'), 'N',    A('n'), A('n'), A('N'), C('N'),
```

```

/* 50 - 'm' */ L('m'), 'M', A('m'), A('m'), A('M'), C('M'),
/* 51 - ',' */ ',' , ';' , A(','), A(','), A(';'), C('@'),
/* 52 - '.' */ '.' , ':' , A('.'), A('.'), A(':'), C('@'),
/* 53 - '/' */ '-' , '_' , A('-'), A('-'), A('_'), C('@'),
/* 54 - r. SHIFT*/ SHIFT, SHIFT, SHIFT, SHIFT, SHIFT, SHIFT,
/* 55 - '*' */ '*' , '*' , A('*'), A('*'), A('*'), C('@'),
/* 56 - ALT */ ALT, ALT, ALT, ALT, ALT, ALT,
/* 57 - ' ' */ ' ' , ' ' , A(' '), A(' '), A(' '), C('@'),
/* 58 - CapsLck */ CALOCK, CALOCK, CALOCK, CALOCK, CALOCK, CALOCK,
/* 59 - F1 */ F1, SF1, AF1, AF1, ASF1, CF1,
/* 60 - F2 */ F2, SF2, AF2, AF2, ASF2, CF2,
/* 61 - F3 */ F3, SF3, AF3, AF3, ASF3, CF3,
/* 62 - F4 */ F4, SF4, AF4, AF4, ASF4, CF4,
/* 63 - F5 */ F5, SF5, AF5, AF5, ASF5, CF5,
/* 64 - F6 */ F6, SF6, AF6, AF6, ASF6, CF6,
/* 65 - F7 */ F7, SF7, AF7, AF7, ASF7, CF7,
/* 66 - F8 */ F8, SF8, AF8, AF8, ASF8, CF8,
/* 67 - F9 */ F9, SF9, AF9, AF9, ASF9, CF9,
/* 68 - F10 */ F10, SF10, AF10, AF10, ASF10, CF10,
/* 69 - NumLock */ NLOCK, NLOCK, NLOCK, NLOCK, NLOCK, NLOCK,
/* 70 - ScrLock */ SLOCK, SLOCK, SLOCK, SLOCK, SLOCK, SLOCK,
/* 71 - Home */ HOME, '7', AHOME, AHOME, A('7'), CHOME,
/* 72 - CurUp */ UP, '8', AUP, AUP, A('8'), CUP,
/* 73 - PgUp */ PGUP, '9', APGUP, APGUP, A('9'), CPGUP,
/* 74 - '-' */ '-' , '-' , '-' , '-' , '-' , '-' ,
/* 75 - Left */ LEFT, '4', ALEFT, ALEFT, A('4'), CLEFT,
/* 76 - MID */ MID, '5', AMID, AMID, A('5'), CMID,
/* 77 - Right */ RIGHT, '6', ARIGHT, ARIGHT, A('6'), CRIGHT,
/* 78 - '+' */ '+' , '+' , '+' , '+' , '+' , '+' ,
/* 79 - End */ END, '1', AEND, AEND, A('1'), CEND,
/* 80 - Down */ DOWN, '2', ADOWN, ADOWN, A('2'), CDOWN,
/* 81 - PgDown */ PGDN, '3', APGDN, APGDN, A('3'), CPGDN,
/* 82 - Insert */ INSRT, '0', AINSRT, AINSRT, A('0'), CINSRT,
/* 83 - Delete */ 0177, '.' , A(0177), A(0177), A('.'), 0177,
/* 84 - Enter */ C('M'), C('M'), CA('M'), CA('M'), CA('M'), C('J'),
/* 85 - ??? */ 0, 0, 0, 0, 0, 0,
/* 86 - ??? */ '<' , '>' , A('<') , A('>') , A('>') , C('@') ,
/* 87 - F11 */ F11, SF11, AF11, AF11, ASF11, CF11,
/* 88 - F12 */ F12, SF12, AF12, AF12, ASF12, CF12,
/* 89 - ??? */ 0, 0, 0, 0, 0, 0,
/* 90 - ??? */ 0, 0, 0, 0, 0, 0,
/* 91 - ??? */ 0, 0, 0, 0, 0, 0,
/* 92 - ??? */ 0, 0, 0, 0, 0, 0,
/* 93 - ??? */ 0, 0, 0, 0, 0, 0,
/* 94 - ??? */ 0, 0, 0, 0, 0, 0,
/* 95 - ??? */ 0, 0, 0, 0, 0, 0,
/* 96 - EXT_KEY */ EXTKEY, EXTKEY, EXTKEY, EXTKEY, EXTKEY, EXTKEY,
/* 97 - ??? */ 0, 0, 0, 0, 0, 0,
/* 98 - ??? */ 0, 0, 0, 0, 0, 0,
/* 99 - ??? */ 0, 0, 0, 0, 0, 0,
/*100 - ??? */ 0, 0, 0, 0, 0, 0,
/*101 - ??? */ 0, 0, 0, 0, 0, 0,
/*102 - ??? */ 0, 0, 0, 0, 0, 0,
/*103 - ??? */ 0, 0, 0, 0, 0, 0,
/*104 - ??? */ 0, 0, 0, 0, 0, 0,
/*105 - ??? */ 0, 0, 0, 0, 0, 0,
/*106 - ??? */ 0, 0, 0, 0, 0, 0,
/*107 - ??? */ 0, 0, 0, 0, 0, 0,
/*108 - ??? */ 0, 0, 0, 0, 0, 0,
/*109 - ??? */ 0, 0, 0, 0, 0, 0,
/*110 - ??? */ 0, 0, 0, 0, 0, 0,
/*111 - ??? */ 0, 0, 0, 0, 0, 0,
/*112 - ??? */ 0, 0, 0, 0, 0, 0,
/*113 - ??? */ 0, 0, 0, 0, 0, 0,
/*114 - ??? */ 0, 0, 0, 0, 0, 0,
/*115 - ??? */ 0, 0, 0, 0, 0, 0,
/*116 - ??? */ 0, 0, 0, 0, 0, 0,
/*117 - ??? */ 0, 0, 0, 0, 0, 0,
/*118 - ??? */ 0, 0, 0, 0, 0, 0,
/*119 - ??? */ 0, 0, 0, 0, 0, 0,
/*120 - ??? */ 0, 0, 0, 0, 0, 0,
/*121 - ??? */ 0, 0, 0, 0, 0, 0,
/*122 - ??? */ 0, 0, 0, 0, 0, 0,
/*123 - ??? */ 0, 0, 0, 0, 0, 0,

```

```
/*124 - ??? */ 0, 0, 0, 0, 0, 0,
/*125 - ??? */ 0, 0, 0, 0, 0, 0,
/*126 - ??? */ 0, 0, 0, 0, 0, 0,
/*127 - ??? */ 0, 0, 0, 0, 0, 0
};
```



```

/* Keymap for the Olivetti M24. */

u16_t keymap[NR_SCAN_CODES * MAP_COLS] = {

/* scan-code      !Shift  Shift   Alt1    Alt2    Alt+Sh  Ctrl    */
/* ===== */
/* 00 - none      */      0,      0,      0,      0,      0,      0,
/* 01 - ESC       */      C('['), C('['), CA('['), CA('['), CA('['), C('['),
/* 02 - '1'       */      '1',    '!',    A('1'), A('1'), A('!'), C('A'),
/* 03 - '2'       */      '2',    '"',    A('2'), A('2'), A('"'), C('B'),
/* 04 - '3'       */      '3',    '#',    A('3'), A('3'), A('#'), C('C'),
/* 05 - '4'       */      '4',    '$',    A('4'), A('4'), A('$'), C('D'),
/* 06 - '5'       */      '5',    '%',    A('5'), A('5'), A('%'), C('E'),
/* 07 - '6'       */      '6',    '&',    A('6'), A('6'), A('&'), C('F'),
/* 08 - '7'       */      '7',    '\',    A('7'), A('7'), A('\'), C('G'),
/* 09 - '8'       */      '8',    '(',    A('8'), A('8'), A('('), C('H'),
/* 10 - '9'       */      '9',    ')',    A('9'), A('9'), A(')'), C('I'),
/* 11 - '0'       */      '0',    '_',    A('0'), A('0'), A('_'), C('@'),
/* 12 - '-'       */      '-',    '=',    A('-'), A('-'), A('='), C('@'),
/* 13 - '='       */      '^',    '~',    A('^'), A('^'), A('~'), C('^'),
/* 14 - BS        */      C('H'), C('H'), CA('H'), CA('H'), CA('H'), 0177,
/* 15 - TAB       */      C('I'), C('I'), CA('I'), CA('I'), CA('I'), C('I'),
/* 16 - 'q'       */      L('q'), 'Q',    A('q'), A('q'), A('Q'), C('Q'),
/* 17 - 'w'       */      L('w'), 'W',    A('w'), A('w'), A('W'), C('W'),
/* 18 - 'e'       */      L('e'), 'E',    A('e'), A('e'), A('E'), C('E'),
/* 19 - 'r'       */      L('r'), 'R',    A('r'), A('r'), A('R'), C('R'),
/* 20 - 't'       */      L('t'), 'T',    A('t'), A('t'), A('T'), C('T'),
/* 21 - 'y'       */      L('y'), 'Y',    A('y'), A('y'), A('Y'), C('Y'),
/* 22 - 'u'       */      L('u'), 'U',    A('u'), A('u'), A('U'), C('U'),
/* 23 - 'i'       */      L('i'), 'I',    A('i'), A('i'), A('I'), C('I'),
/* 24 - 'o'       */      L('o'), 'O',    A('o'), A('o'), A('O'), C('O'),
/* 25 - 'p'       */      L('p'), 'P',    A('p'), A('p'), A('P'), C('P'),
/* 26 - '['       */      '@',    '\',    A('@'), A('@'), A('\'), C('@'),
/* 27 - ']'       */      '[',    '{',    A('['), A('['), A('{'), C('['),
/* 28 - CR/LF     */      C('M'), C('M'), CA('M'), CA('M'), CA('M'), C('J'),
/* 29 - Ctrl      */      CTRL,  CTRL,  CTRL,  CTRL,  CTRL,  CTRL,
/* 30 - 'a'       */      L('a'), 'A',    A('a'), A('a'), A('A'), C('A'),
/* 31 - 's'       */      L('s'), 'S',    A('s'), A('s'), A('S'), C('S'),
/* 32 - 'd'       */      L('d'), 'D',    A('d'), A('d'), A('D'), C('D'),
/* 33 - 'f'       */      L('f'), 'F',    A('f'), A('f'), A('F'), C('F'),
/* 34 - 'g'       */      L('g'), 'G',    A('g'), A('g'), A('G'), C('G'),
/* 35 - 'h'       */      L('h'), 'H',    A('h'), A('h'), A('H'), C('H'),
/* 36 - 'j'       */      L('j'), 'J',    A('j'), A('j'), A('J'), C('J'),
/* 37 - 'k'       */      L('k'), 'K',    A('k'), A('k'), A('K'), C('K'),
/* 38 - 'l'       */      L('l'), 'L',    A('l'), A('l'), A('L'), C('L'),
/* 39 - ';'       */      ';',    '+',    A(';'), A(';'), A('+'), C('@'),
/* 40 - '\'       */      ':',    '*',    A(':'), A(':'), A('*'), C('@'),
/* 41 - '`'       */      '~',    '~',    A('~'), A('~'), A('~'), C('@'),
/* 42 - l. SHIFT */      SHIFT, SHIFT, SHIFT, SHIFT, SHIFT, SHIFT,
/* 43 - '\\       */      '\\', '|',    A('\\'), A('\\'), A('|'), C('\\'),
/* 44 - 'z'       */      L('z'), 'Z',    A('z'), A('z'), A('Z'), C('Z'),
/* 45 - 'x'       */      L('x'), 'X',    A('x'), A('x'), A('X'), C('X'),
/* 46 - 'c'       */      L('c'), 'C',    A('c'), A('c'), A('C'), C('C'),
/* 47 - 'v'       */      L('v'), 'V',    A('v'), A('v'), A('V'), C('V'),
/* 48 - 'b'       */      L('b'), 'B',    A('b'), A('b'), A('B'), C('B'),
/* 49 - 'n'       */      L('n'), 'N',    A('n'), A('n'), A('N'), C('N'),
/* 50 - 'm'       */      L('m'), 'M',    A('m'), A('m'), A('M'), C('M'),
/* 51 - '<'       */      '<', '<', A('<'), A('<'), A('<'), C('@'),
/* 52 - '>'       */      '>', '>', A('>'), A('>'), A('>'), C('@'),
/* 53 - '/'       */      '/', '?',    A('/'), A('/'), A('?'), C('@'),
/* 54 - r. SHIFT */      SHIFT, SHIFT, SHIFT, SHIFT, SHIFT, SHIFT,
/* 55 - '*'       */      '*', '*',    A('*'), A('*'), A('*'), C('@'),
/* 56 - ALT       */      ALT,  ALT,  ALT,  ALT,  ALT,  ALT,
/* 57 - ' '       */      ' ', ' ',    A(' '), A(' '), A(' '), C('@'),
/* 58 - CapsLock  */      CALOCK, CALOCK, CALOCK, CALOCK, CALOCK, CALOCK,
/* 59 - F1        */      F1,    SF1,  AF1,  AF1,  ASF1, CF1,
/* 60 - F2        */      F2,    SF2,  AF2,  AF2,  ASF2, CF2,
/* 61 - F3        */      F3,    SF3,  AF3,  AF3,  ASF3, CF3,
/* 62 - F4        */      F4,    SF4,  AF4,  AF4,  ASF4, CF4,
/* 63 - F5        */      F5,    SF5,  AF5,  AF5,  ASF5, CF5,
/* 64 - F6        */      F6,    SF6,  AF6,  AF6,  ASF6, CF6,
/* 65 - F7        */      F7,    SF7,  AF7,  AF7,  ASF7, CF7,
/* 66 - F8        */      F8,    SF8,  AF8,  AF8,  ASF8, CF8,
/* 67 - F9        */      F9,    SF9,  AF9,  AF9,  ASF9, CF9,

```

```

/* 68 - F10 */ F10, SF10, AF10, AF10, ASF10, CF10,
/* 69 - NumLock */ C('S'), C('S'), C('S'), C('S'), C('S'), C('S'),
/* 70 - ScrLock */ SLOCK, SLOCK, SLOCK, SLOCK, SLOCK, SLOCK,
/* 71 - Home */ HOME, '7', AHOME, AHOME, A('7'), CHOME,
/* 72 - CurUp */ UP, '8', AUP, AUP, A('8'), CUP,
/* 73 - PgUp */ PGUP, '9', APGUP, APGUP, A('9'), CPGUP,
/* 74 - '-' */ NMIN, '-', ANMIN, ANMIN, A('-'), CNMIN,
/* 75 - Left */ LEFT, '4', ALEFT, ALEFT, A('4'), CLEFT,
/* 76 - MID */ MID, '5', AMID, AMID, A('5'), CMID,
/* 77 - Right */ RIGHT, '6', ARIGHT, ARIGHT, A('6'), CRIGHT,
/* 78 - '+' */ PLUS, '+', APLUS, APLUS, A('+'), CPLUS,
/* 79 - End */ END, '1', AEND, AEND, A('1'), CEND,
/* 80 - Down */ DOWN, '2', ADOWN, ADOWN, A('2'), CDOWN,
/* 81 - PgDown */ PGDN, '3', APGDN, APGDN, A('3'), CPGDN,
/* 82 - Insert */ INSRT, '0', AINSRT, AINSRT, A('0'), CINSRT,
/* 83 - Delete */ 0177, '.', A(0177), A(0177), A('.'), 0177,
/* 84 - Enter */ ' ', ' ', A(' '), A(' '), A(' '), C('@'),
/* 85 - ??? */ LEFT, 014, A(014), A(014), A(014), 014,
/* 86 - ??? */ 0212, 0212, 0212, 0212, 0212, 0212,
/* 87 - F11 */ C('M'), C('M'), CA('M'), CA('M'), CA('M'), C('J'),
/* 88 - F12 */ F12, SF12, AF12, AF12, ASF12, CF12,
/* 89 - ??? */ DOWN, DOWN, DOWN, DOWN, DOWN, DOWN,
/* 90 - ??? */ RIGHT, RIGHT, RIGHT, RIGHT, RIGHT, RIGHT,
/* 91 - ??? */ UP, UP, UP, UP, UP, UP,
/* 92 - ??? */ LEFT, LEFT, LEFT, LEFT, LEFT, LEFT,
/* 93 - ??? */ 0, 0, 0, 0, 0, 0,
/* 94 - ??? */ 0, 0, 0, 0, 0, 0,
/* 95 - ??? */ 0, 0, 0, 0, 0, 0,
/* 96 - EXT_KEY */ EXTKEY, EXTKEY, EXTKEY, EXTKEY, EXTKEY, EXTKEY,
/* 97 - ??? */ 0, 0, 0, 0, 0, 0,
/* 98 - ??? */ 0, 0, 0, 0, 0, 0,
/* 99 - ??? */ 0, 0, 0, 0, 0, 0,
/*100 - ??? */ 0, 0, 0, 0, 0, 0,
/*101 - ??? */ 0, 0, 0, 0, 0, 0,
/*102 - ??? */ 0, 0, 0, 0, 0, 0,
/*103 - ??? */ 0, 0, 0, 0, 0, 0,
/*104 - ??? */ 0, 0, 0, 0, 0, 0,
/*105 - ??? */ 0, 0, 0, 0, 0, 0,
/*106 - ??? */ 0, 0, 0, 0, 0, 0,
/*107 - ??? */ 0, 0, 0, 0, 0, 0,
/*108 - ??? */ 0, 0, 0, 0, 0, 0,
/*109 - ??? */ 0, 0, 0, 0, 0, 0,
/*110 - ??? */ 0, 0, 0, 0, 0, 0,
/*111 - ??? */ 0, 0, 0, 0, 0, 0,
/*112 - ??? */ 0, 0, 0, 0, 0, 0,
/*113 - ??? */ 0, 0, 0, 0, 0, 0,
/*114 - ??? */ 0, 0, 0, 0, 0, 0,
/*115 - ??? */ 0, 0, 0, 0, 0, 0,
/*116 - ??? */ 0, 0, 0, 0, 0, 0,
/*117 - ??? */ 0, 0, 0, 0, 0, 0,
/*118 - ??? */ 0, 0, 0, 0, 0, 0,
/*119 - ??? */ 0, 0, 0, 0, 0, 0,
/*120 - ??? */ 0, 0, 0, 0, 0, 0,
/*121 - ??? */ 0, 0, 0, 0, 0, 0,
/*122 - ??? */ 0, 0, 0, 0, 0, 0,
/*123 - ??? */ 0, 0, 0, 0, 0, 0,
/*124 - ??? */ 0, 0, 0, 0, 0, 0,
/*125 - ??? */ 0, 0, 0, 0, 0, 0,
/*126 - ??? */ 0, 0, 0, 0, 0, 0,
/*127 - ??? */ 0, 0, 0, 0, 0, 0
};

```

```

/*
 * Keymap for US MF-2 keyboard. + Polish national letters.
 * Modified by Antek Sawicki <tenox@tenox.tc>
 * Charset: ISO-8859-2 - [Polska Norma PN-93 T-42118]
 */

u16_t keymap[NR_SCAN_CODES * MAP_COLS] = {

/* scan-code      !Shift  Shift  Alt1    Alt2    Alt+Sh  Ctrl    */
/* ===== */
/* 00 - none      */      0,      0,      0,      0,      0,      0,
/* 01 - ESC       */      C('['), C('['), CA('['), CA('['), CA('['), C('['),
/* 02 - '1'       */      '1',    '!',    A('1'), A('1'), A('!'), C('A'),
/* 03 - '2'       */      '2',    '@',    A('2'), A('2'), A('@'), C('@'),
/* 04 - '3'       */      '3',    '#',    A('3'), A('3'), A('#'), C('C'),
/* 05 - '4'       */      '4',    '$',    A('4'), A('4'), A('$'), C('D'),
/* 06 - '5'       */      '5',    '%',    A('5'), A('5'), A('%'), C('E'),
/* 07 - '6'       */      '6',    '^',    A('6'), A('6'), A('^'), C('^'),
/* 08 - '7'       */      '7',    '&',    A('7'), A('7'), A('&'), C('G'),
/* 09 - '8'       */      '8',    '*',    A('8'), A('8'), A('*'), C('H'),
/* 10 - '9'       */      '9',    '(',    A('9'), A('9'), A('('), C('I'),
/* 11 - '0'       */      '0',    ')',    A('0'), A('0'), A(')'), C('@'),
/* 12 - '-'       */      '-',    '_',    A('-'), A('-'), A('_'), C('_'),
/* 13 - '='       */      '=',    '+',    A('='), A('='), A('+'), C('@'),
/* 14 - BS        */      C('H'), C('H'), CA('H'), CA('H'), CA('H'), 0177,
/* 15 - TAB       */      C('I'), C('I'), CA('I'), CA('I'), CA('I'), C('I'),
/* 16 - 'q'       */      L('q'), 'Q',    A('q'), A('q'), A('Q'), C('Q'),
/* 17 - 'w'       */      L('w'), 'W',    A('w'), A('w'), A('W'), C('W'),
/* 18 - 'e'       */      L('e'), 'E',    A(0xea), A(0xea), A(0xca), C('E'),
/* 19 - 'r'       */      L('r'), 'R',    A('r'), A('r'), A('R'), C('R'),
/* 20 - 't'       */      L('t'), 'T',    A('t'), A('t'), A('T'), C('T'),
/* 21 - 'y'       */      L('y'), 'Y',    A('y'), A('y'), A('Y'), C('Y'),
/* 22 - 'u'       */      L('u'), 'U',    A('u'), A('u'), A('U'), C('U'),
/* 23 - 'i'       */      L('i'), 'I',    A('i'), A('i'), A('I'), C('I'),
/* 24 - 'o'       */      L('o'), 'O',    A(0xf3), A(0xf3), A(0xd3), C('O'),
/* 25 - 'p'       */      L('p'), 'P',    A('p'), A('p'), A('P'), C('P'),
/* 26 - '['       */      '[',    '{',    A('['), A('['), A('{'), C('['),
/* 27 - ']'       */      ']',    '}',    A(']'), A(']'), A('}'), C(']'),
/* 28 - CR/LF     */      C('M'), C('M'), CA('M'), CA('M'), CA('M'), C('J'),
/* 29 - Ctrl      */      CTRL,  CTRL,  CTRL,  CTRL,  CTRL,  CTRL,
/* 30 - 'a'       */      L('a'), 'A',    A(0xb1), A(0xb1), A(0xa1), C('A'),
/* 31 - 's'       */      L('s'), 'S',    A(0xb6), A(0xb6), A(0xa6), C('S'),
/* 32 - 'd'       */      L('d'), 'D',    A('d'), A('d'), A('D'), C('D'),
/* 33 - 'f'       */      L('f'), 'F',    A('f'), A('f'), A('F'), C('F'),
/* 34 - 'g'       */      L('g'), 'G',    A('g'), A('g'), A('G'), C('G'),
/* 35 - 'h'       */      L('h'), 'H',    A('h'), A('h'), A('H'), C('H'),
/* 36 - 'j'       */      L('j'), 'J',    A('j'), A('j'), A('J'), C('J'),
/* 37 - 'k'       */      L('k'), 'K',    A('k'), A('k'), A('K'), C('K'),
/* 38 - 'l'       */      L('l'), 'L',    A(0xb3), A(0xb3), A(0xa3), C('L'),
/* 39 - ';'       */      ';',    ':',    A(';'), A(';'), A(':'), C('@'),
/* 40 - '\'       */      '\',    '"',    A('\'), A('\'), A('"'), C('@'),
/* 41 - '~'       */      '~',    '~',    A('~'), A('~'), A('~'), C('@'),
/* 42 - l. SHIFT */      SHIFT, SHIFT, SHIFT, SHIFT, SHIFT, SHIFT,
/* 43 - '\\       */      '\\',  '|',    A('\\'), A('\\'), A('|'), C('\\'),
/* 44 - 'z'       */      L('z'), 'Z',    A(0xbf), A(0xbf), A(0xaf), C('Z'),
/* 45 - 'x'       */      L('x'), 'X',    A(0xbc), A(0xbc), A(0xac), C('X'),
/* 46 - 'c'       */      L('c'), 'C',    A(0xe6), A(0xe6), A(0xc6), C('C'),
/* 47 - 'v'       */      L('v'), 'V',    A('v'), A('v'), A('V'), C('V'),
/* 48 - 'b'       */      L('b'), 'B',    A('b'), A('b'), A('B'), C('B'),
/* 49 - 'n'       */      L('n'), 'N',    A(0xf1), A(0xf1), A(0xd1), C('N'),
/* 50 - 'm'       */      L('m'), 'M',    A('m'), A('m'), A('M'), C('M'),
/* 51 - ','       */      ',',    '<',    A(','), A(','), A('<'), C('@'),
/* 52 - '.'       */      '.',    '>',    A('.'), A('.'), A('>'), C('@'),
/* 53 - '/'       */      '/',    '?',    A('/'), A('/'), A('?'), C('@'),
/* 54 - r. SHIFT */      SHIFT, SHIFT, SHIFT, SHIFT, SHIFT, SHIFT,
/* 55 - '*'       */      '*',    '*',    A('*'), A('*'), A('*'), C('@'),
/* 56 - ALT       */      ALT,  ALT,  ALT,  ALT,  ALT,  ALT,
/* 57 - ' '       */      ' ',  ' ',  A(' '), A(' '), A(' '), C('@'),
/* 58 - CapsLck   */      CALOCK, CALOCK, CALOCK, CALOCK, CALOCK, CALOCK,
/* 59 - F1        */      F1,    SF1,   AF1,   AF1,   ASF1,  CF1,
/* 60 - F2        */      F2,    SF2,   AF2,   AF2,   ASF2,  CF2,
/* 61 - F3        */      F3,    SF3,   AF3,   AF3,   ASF3,  CF3,
/* 62 - F4        */      F4,    SF4,   AF4,   AF4,   ASF4,  CF4,

```

```

/* 63 - F5      */      F5,      SF5,      AF5,      AF5,      ASF5,      CF5,
/* 64 - F6      */      F6,      SF6,      AF6,      AF6,      ASF6,      CF6,
/* 65 - F7      */      F7,      SF7,      AF7,      AF7,      ASF7,      CF7,
/* 66 - F8      */      F8,      SF8,      AF8,      AF8,      ASF8,      CF8,
/* 67 - F9      */      F9,      SF9,      AF9,      AF9,      ASF9,      CF9,
/* 68 - F10     */      F10,     SF10,     AF10,     AF10,     ASF10,     CF10,
/* 69 - NumLock */      NLOCK,   NLOCK,   NLOCK,   NLOCK,   NLOCK,   NLOCK,
/* 70 - ScrLock */      SLOCK,   SLOCK,   SLOCK,   SLOCK,   SLOCK,   SLOCK,
/* 71 - Home    */      HOME,    '7',    AHOME,    AHOME,    A('7'),  CHOME,
/* 72 - CurUp   */      UP,      '8',    AUP,      AUP,      A('8'),  CUP,
/* 73 - PgUp    */      PGUP,    '9',    APGUP,   APGUP,   A('9'),  CPGUP,
/* 74 - '-'     */      NMIN,    '-',    ANMIN,   ANMIN,   A('-'),  CNMIN,
/* 75 - Left    */      LEFT,    '4',    ALEFT,   ALEFT,   A('4'),  CLEFT,
/* 76 - MID     */      MID,     '5',    AMID,    AMID,    A('5'),  CMID,
/* 77 - Right   */      RIGHT,   '6',    ARIGHT,  ARIGHT,  A('6'),  CRIGHT,
/* 78 - '+'     */      PLUS,    '+',    APLUS,   APLUS,   A('+'),  CPLUS,
/* 79 - End     */      END,     '1',    AEND,    AEND,    A('1'),  CEND,
/* 80 - Down    */      DOWN,    '2',    ADOWN,   ADOWN,   A('2'),  CDOWN,
/* 81 - PgDown  */      PGDN,    '3',    APGDN,   APGDN,   A('3'),  CPGDN,
/* 82 - Insert  */      INSRT,   '0',    AINSRT,  AINSRT,  A('0'),  CINSRT,
/* 83 - Delete  */      0177,   '.',   A(0177), A(0177), A('.'),  0177,
/* 84 - Enter   */      C('M'),  C('M'), CA('M'), CA('M'), CA('M'), C('J'),
/* 85 - ???     */      0,        0,      0,        0,        0,        0,
/* 86 - ???     */      '<',     '>',    A('<'),  A('|'),  A('>'),  C('@'),
/* 87 - F11     */      F11,     SF11,   AF11,    AF11,    ASF11,   CF11,
/* 88 - F12     */      F12,     SF12,   AF12,    AF12,    ASF12,   CF12,
/* 89 - ???     */      0,        0,      0,        0,        0,        0,
/* 90 - ???     */      0,        0,      0,        0,        0,        0,
/* 91 - ???     */      0,        0,      0,        0,        0,        0,
/* 92 - ???     */      0,        0,      0,        0,        0,        0,
/* 93 - ???     */      0,        0,      0,        0,        0,        0,
/* 94 - ???     */      0,        0,      0,        0,        0,        0,
/* 95 - ???     */      0,        0,      0,        0,        0,        0,
/* 96 - EXT_KEY */      EXTKEY, EXTKEY, EXTKEY, EXTKEY, EXTKEY, EXTKEY,
/* 97 - ???     */      0,        0,      0,        0,        0,        0,
/* 98 - ???     */      0,        0,      0,        0,        0,        0,
/* 99 - ???     */      0,        0,      0,        0,        0,        0,
/*100 - ???     */      0,        0,      0,        0,        0,        0,
/*101 - ???     */      0,        0,      0,        0,        0,        0,
/*102 - ???     */      0,        0,      0,        0,        0,        0,
/*103 - ???     */      0,        0,      0,        0,        0,        0,
/*104 - ???     */      0,        0,      0,        0,        0,        0,
/*105 - ???     */      0,        0,      0,        0,        0,        0,
/*106 - ???     */      0,        0,      0,        0,        0,        0,
/*107 - ???     */      0,        0,      0,        0,        0,        0,
/*108 - ???     */      0,        0,      0,        0,        0,        0,
/*109 - ???     */      0,        0,      0,        0,        0,        0,
/*110 - ???     */      0,        0,      0,        0,        0,        0,
/*111 - ???     */      0,        0,      0,        0,        0,        0,
/*112 - ???     */      0,        0,      0,        0,        0,        0,
/*113 - ???     */      0,        0,      0,        0,        0,        0,
/*114 - ???     */      0,        0,      0,        0,        0,        0,
/*115 - ???     */      0,        0,      0,        0,        0,        0,
/*116 - ???     */      0,        0,      0,        0,        0,        0,
/*117 - ???     */      0,        0,      0,        0,        0,        0,
/*118 - ???     */      0,        0,      0,        0,        0,        0,
/*119 - ???     */      0,        0,      0,        0,        0,        0,
/*120 - ???     */      0,        0,      0,        0,        0,        0,
/*121 - ???     */      0,        0,      0,        0,        0,        0,
/*122 - ???     */      0,        0,      0,        0,        0,        0,
/*123 - ???     */      0,        0,      0,        0,        0,        0,
/*124 - ???     */      0,        0,      0,        0,        0,        0,
/*125 - ???     */      0,        0,      0,        0,        0,        0,
/*126 - ???     */      0,        0,      0,        0,        0,        0,
/*127 - ???     */      0,        0,      0,        0,        0,        0
};

```

```

/* Keymap for Scandinavian keyboard.
 * by Oliver Reitalu, nolx@nolx.tartu.ee
 * preliminary version, 8 Sept 1996
 */

```

```

ul6_t keymap[NR_SCAN_CODES * MAP_COLS] = {

```

```

/* scan-code      unsh      Shift      Alt      AltGr      Alt+Sh      Ctrl      */
/* ===== */
/* 00 - none      */      0,          0,          0,          0,          0,          0,
/* 01 - ESC       */      C('['), C('['), CA('['), C('['), C('['), C('['),
/* 02 - '1'       */      '1',    '!',    A('1'), '1',    '!',    C('A'),
/* 03 - '2'       */      '2',    '"',    A('2'), '@',    '@',    C('@'),
/* 04 - '3'       */      '3',    '#',    A('3'), 156,  '#',    C('C'),
/* 05 - '4'       */      '4',    '$',    A('4'), '$',    '$',    C('D'),
/* 06 - '5'       */      '5',    '%',    A('5'), '%',    '%',    C('E'),
/* 07 - '6'       */      '6',    '&',    A('6'), '^',    '^',    C('^'),
/* 08 - '7'       */      '7',    '/',    A('7'), '{',    '&',    C('G'),
/* 09 - '8'       */      '8',    '(',    A('8'), '[',    '*',    C('H'),
/* 10 - '9'       */      '9',    ')',    A('9'), ']',    '(',    C('I'),
/* 11 - '0'       */      '0',    '=',    A('0'), '}',    ')',    C('@'),
/* 12 - '_'       */      '+',    '?',    0341,  '\\',  '-',    C('_'),
/* 13 - '='       */      '\\',    '~',    A('\\'), '=',  '+',    C('@'),
/* 14 - BS        */      C('H'), C('H'), CA('H'), C('H'), C('H'), 0177,
/* 15 - TAB       */      C('I'), C('I'), CA('I'), C('I'), C('I'), C('I'),
/* 16 - 'q'       */      L('q'), 'Q',    A('q'), '@',    'Q',    C('Q'),
/* 17 - 'w'       */      L('w'), 'W',    A('w'), 'w',    'W',    C('W'),
/* 18 - 'e'       */      L('e'), 'E',    A('e'), 'e',    'E',    C('E'),
/* 19 - 'r'       */      L('r'), 'R',    A('r'), 'r',    'R',    C('R'),
/* 20 - 't'       */      L('t'), 'T',    A('t'), 't',    'T',    C('T'),
/* 21 - 'y'       */      L('y'), 'Y',    A('y'), 'y',    'Y',    C('Y'),
/* 22 - 'u'       */      L('u'), 'U',    A('u'), 'u',    'U',    C('U'),
/* 23 - 'i'       */      L('i'), 'I',    A('i'), 'i',    'I',    C('I'),
/* 24 - 'o'       */      L('o'), 'O',    A('o'), 'o',    'O',    C('O'),
/* 25 - 'p'       */      L('p'), 'P',    A('p'), 'p',    'P',    C('P'),
/* 26 - '['       */      L(134), 143,  0201,  '[',    '{',    C('['),
/* 27 - ']'       */      '\\',    '^',    A('+'), '~',    ']',    C(']'),
/* 28 - CR/LF     */      C('M'), C('M'), CA('M'), C('M'), C('M'), C('J'),
/* 29 - Ctrl      */      CTRL,   CTRL,   CTRL,   CTRL,   CTRL,   CTRL,
/* 30 - 'a'       */      L('a'), 'A',    A('a'), 'a',    'A',    C('A'),
/* 31 - 's'       */      L('s'), 'S',    A('s'), 's',    'S',    C('S'),
/* 32 - 'd'       */      L('d'), 'D',    A('d'), 'd',    'D',    C('D'),
/* 33 - 'f'       */      L('f'), 'F',    A('f'), 'f',    'F',    C('F'),
/* 34 - 'g'       */      L('g'), 'G',    A('g'), 'g',    'G',    C('G'),
/* 35 - 'h'       */      L('h'), 'H',    A('h'), 'h',    'H',    C('H'),
/* 36 - 'j'       */      L('j'), 'J',    A('j'), 'j',    'J',    C('J'),
/* 37 - 'k'       */      L('k'), 'K',    A('k'), 'k',    'K',    C('K'),
/* 38 - 'l'       */      L('l'), 'L',    A('l'), 'l',    'L',    C('L'),
/* 39 - ';'       */      L(0224), 0231,  0224,  ';',    ':',    C('@'),
/* 40 - '\\'      */      L(0204), 0216,  0204,  '\\',   '"',    C('@'),
/* 41 - '`'       */      L(21),   171,   A('^'), '~',    '~',    C('^'),
/* 42 - SHIFT     */      SHIFT,   SHIFT,   SHIFT,   SHIFT,   SHIFT,   SHIFT,
/* 43 - '\\\\'    */      39,      '*',    A('#'), '\\\\', '|',    C('\\\\'),
/* 44 - 'z'       */      L('z'), 'Z',    A('z'), 'z',    'Z',    C('Z'),
/* 45 - 'x'       */      L('x'), 'X',    A('x'), 'x',    'X',    C('X'),
/* 46 - 'c'       */      L('c'), 'C',    A('c'), 'c',    'C',    C('C'),
/* 47 - 'v'       */      L('v'), 'V',    A('v'), 'v',    'V',    C('V'),
/* 48 - 'b'       */      L('b'), 'B',    A('b'), 'b',    'B',    C('B'),
/* 49 - 'n'       */      L('n'), 'N',    A('n'), 'n',    'N',    C('N'),
/* 50 - 'm'       */      L('m'), 'M',    A('m'), 0346,  'M',    C('M'),
/* 51 - ','       */      ',',    ';',    A(','), '<',    '<',    C('@'),
/* 52 - '.'       */      '.',    ':',    A('.'), '>',    '>',    C('@'),
/* 53 - '/'       */      '-',    '_',    A('-'), '?',    '?',    C('_'),
/* 54 - SHIFT     */      SHIFT,   SHIFT,   SHIFT,   SHIFT,   SHIFT,   SHIFT,
/* 55 - '*'       */      '*',    '*',    A('*'), '*',    '*',    C('@'),
/* 56 - ALT       */      ALT,     ALT,     ALT,     ALT,     ALT,     ALT,
/* 57 - ' '       */      ' ',    ' ',    A(' '), ' ',    ' ',    C('@'),
/* 58 - CapsLck   */      CALOCK,  CALOCK,  CALOCK,  CALOCK,  CALOCK,  CALOCK,
/* 59 - F1        */      F1,     SF1,    AF1,    AF1,    ASF1,   CF1,
/* 60 - F2        */      F2,     SF2,    AF2,    AF2,    ASF2,   CF2,
/* 61 - F3        */      F3,     SF3,    AF3,    AF3,    ASF3,   CF3,
/* 62 - F4        */      F4,     SF4,    AF4,    AF4,    ASF4,   CF4,
/* 63 - F5        */      F5,     SF5,    AF5,    AF5,    ASF5,   CF5,
/* 64 - F6        */      F6,     SF6,    AF6,    AF6,    ASF6,   CF6,

```

```

/* 65 - F7      */      F7,      SF7,      AF7,      AF7,      ASF7,      CF7,
/* 66 - F8      */      F8,      SF8,      AF8,      AF8,      ASF8,      CF8,
/* 67 - F9      */      F9,      SF9,      AF9,      AF9,      ASF9,      CF9,
/* 68 - F10     */      F10,     SF10,     AF10,     AF10,     ASF10,     CF10,
/* 69 - NumLock */      NLOCK,    NLOCK,    NLOCK,    NLOCK,    NLOCK,    NLOCK,
/* 70 - ScrLock */      SLOCK,    SLOCK,    SLOCK,    SLOCK,    SLOCK,    SLOCK,
/* 71 - Home    */      HOME,     '7',    AHOME,    AHOME,    '7',    CHOME,
/* 72 - CurUp   */      UP,       '8',    AUP,     AUP,     '8',    CUP,
/* 73 - PgUp    */      PGUP,     '9',    APGUP,   APGUP,   '9',    CPGUP,
/* 74 - '-'     */      NMIN,     '-',    ANMIN,   ANMIN,   '-',    CNMIN,
/* 75 - Left    */      LEFT,     '4',    ALEFT,   ALEFT,   '4',    CLEFT,
/* 76 - MID     */      MID,      '5',    AMID,    AMID,    '5',    CMID,
/* 77 - Right   */      RIGHT,    '6',    ARIGHT,  ARIGHT,  '6',    CRIGHT,
/* 78 - '+'     */      PLUS,     '+',    APLUS,   APLUS,   '+',    CPLUS,
/* 79 - End     */      END,      '1',    AEND,    AEND,    '1',    CEND,
/* 80 - Down    */      DOWN,     '2',    ADOWN,   ADOWN,   '2',    CDOWN,
/* 81 - PgDown  */      PGDN,     '3',    APGDN,   APGDN,   '3',    CPGDN,
/* 82 - Insert  */      INSRT,    '0',    AINSRT,  AINSRT,  '0',    CINSRT,
/* 83 - Delete  */      0177,    '.',   A(0177), 0177,    '.',   0177,
/* 84 - Enter   */      C('M'),  C('M'), CA('M'), C('M'), C('M'), C('J'),
/* 85 - ???     */      0,        0,     0,       0,       0,       0,
/* 86 - ???     */      '<',     '>',   A('<'), '|',     '>',   C('@'),
/* 87 - F11     */      F11,     SF11,   AF11,    AF11,    ASF11,   CF11,
/* 88 - F12     */      F12,     SF12,   AF12,    AF12,    ASF12,   CF12,
/* 89 - ???     */      0,        0,     0,       0,       0,       0,
/* 90 - ???     */      0,        0,     0,       0,       0,       0,
/* 91 - ???     */      0,        0,     0,       0,       0,       0,
/* 92 - ???     */      0,        0,     0,       0,       0,       0,
/* 93 - ???     */      0,        0,     0,       0,       0,       0,
/* 94 - ???     */      0,        0,     0,       0,       0,       0,
/* 95 - ???     */      0,        0,     0,       0,       0,       0,
/* 96 - EXT_KEY */      EXTKEY,  EXTKEY, EXTKEY,  EXTKEY,  EXTKEY,  EXTKEY,
/* 97 - ???     */      0,        0,     0,       0,       0,       0,
/* 98 - ???     */      0,        0,     0,       0,       0,       0,
/* 99 - ???     */      0,        0,     0,       0,       0,       0,
/*100 - ???     */      0,        0,     0,       0,       0,       0,
/*101 - ???     */      0,        0,     0,       0,       0,       0,
/*102 - ???     */      0,        0,     0,       0,       0,       0,
/*103 - ???     */      0,        0,     0,       0,       0,       0,
/*104 - ???     */      0,        0,     0,       0,       0,       0,
/*105 - ???     */      0,        0,     0,       0,       0,       0,
/*106 - ???     */      0,        0,     0,       0,       0,       0,
/*107 - ???     */      0,        0,     0,       0,       0,       0,
/*108 - ???     */      0,        0,     0,       0,       0,       0,
/*109 - ???     */      0,        0,     0,       0,       0,       0,
/*110 - ???     */      0,        0,     0,       0,       0,       0,
/*111 - ???     */      0,        0,     0,       0,       0,       0,
/*112 - ???     */      0,        0,     0,       0,       0,       0,
/*113 - ???     */      0,        0,     0,       0,       0,       0,
/*114 - ???     */      0,        0,     0,       0,       0,       0,
/*115 - ???     */      0,        0,     0,       0,       0,       0,
/*116 - ???     */      0,        0,     0,       0,       0,       0,
/*117 - ???     */      0,        0,     0,       0,       0,       0,
/*118 - ???     */      0,        0,     0,       0,       0,       0,
/*119 - ???     */      0,        0,     0,       0,       0,       0,
/*120 - ???     */      0,        0,     0,       0,       0,       0,
/*121 - ???     */      0,        0,     0,       0,       0,       0,
/*122 - ???     */      0,        0,     0,       0,       0,       0,
/*123 - ???     */      0,        0,     0,       0,       0,       0,
/*124 - ???     */      0,        0,     0,       0,       0,       0,
/*125 - ???     */      0,        0,     0,       0,       0,       0,
/*126 - ???     */      0,        0,     0,       0,       0,       0,
/*127 - ???     */      0,        0,     0,       0,       0,       0
};

```

```

/* Keymap for Spanish MF-2 keyboard. */
/* Modified by Javier Garcia Martin jawa@inf.deusto.es */

u16_t keymap[NR_SCAN_CODES * MAP_COLS] = {

/* scan-code      !Shift  Shift   Alt     AltGr   Alt+Sh  Ctrl    */
/* ===== */
/* 00 - none      */      0,        0,        0,        0,        0,        0,
/* 01 - ESC       */      C('['), C('['), CA('['), C('['), C('['), C('['),
/* 02 - '1'       */      '1',      '!',      A('1'),  '|',      '!',      C('A'),
/* 03 - '2'       */      '2',      '"',      A('2'),  '@',      '"',      C('@'),
/* 04 - '3'       */      '3',      0372,    A('3'),  '#',      0372,    C('C'),
/* 05 - '4'       */      '4',      '$',      A('4'),  '4',      '$',      C('D'),
/* 06 - '5'       */      '5',      '%',      A('5'),  '5',      '%',      C('E'),
/* 07 - '6'       */      '6',      '&',      A('6'),  0252,    '&',      C('F'),
/* 08 - '7'       */      '7',      '/',      A('7'),  '{',      '/',      C('G'),
/* 09 - '8'       */      '8',      '(',      A('8'),  '(',      '(',      C('H'),
/* 10 - '9'       */      '9',      ')',      A('9'),  ')',      ')',      C('I'),
/* 11 - '0'       */      '0',      '=',      A('0'),  '=',      '=',      C('@'),
/* 12 - '-'       */      '\'',    '?',      A('\'),  '?',      '?',      C('_'),
/* 13 - '='       */      0255,    0250,    A(0255), 0250,    0250,    C('@'),
/* 14 - BS        */      C('H'), C('H'), CA('H'), C('H'), C('H'), 0177,
/* 15 - TAB       */      C('I'), C('I'), CA('I'), C('I'), C('I'), C('I'),
/* 16 - 'q'       */      L('q'), 'Q',      A('q'), 'q',      'Q',      C('Q'),
/* 17 - 'w'       */      L('w'), 'W',      A('w'), 'w',      'W',      C('W'),
/* 18 - 'e'       */      L('e'), 'E',      A('e'), 'e',      'E',      C('E'),
/* 19 - 'r'       */      L('r'), 'R',      A('r'), 'r',      'R',      C('R'),
/* 20 - 't'       */      L('t'), 'T',      A('t'), 't',      'T',      C('T'),
/* 21 - 'y'       */      L('y'), 'Y',      A('y'), 'Y',      'Y',      C('Y'),
/* 22 - 'u'       */      L('u'), 'U',      A('u'), 'u',      'U',      C('U'),
/* 23 - 'i'       */      L('i'), 'I',      A('i'), 'i',      'I',      C('I'),
/* 24 - 'o'       */      L('o'), 'O',      A('o'), 'o',      'O',      C('O'),
/* 25 - 'p'       */      L('p'), 'P',      A('p'), 'p',      'P',      C('P'),
/* 26 - '['       */      '\'',    '^',      A('\'),  '^',      '^',      C('['),
/* 27 - ']'       */      '+',      '*',      A('+'),  ']',      '*',      C(']'),
/* 28 - CR/LF     */      C('M'), C('M'), CA('M'), C('M'), C('M'), C('J'),
/* 29 - Ctrl      */      CTRL,  CTRL,    CTRL,    CTRL,    CTRL,    CTRL,
/* 30 - 'a'       */      L('a'), 'A',      A('a'), 'a',      'A',      C('A'),
/* 31 - 's'       */      L('s'), 'S',      A('s'), 's',      'S',      C('S'),
/* 32 - 'd'       */      L('d'), 'D',      A('d'), 'd',      'D',      C('D'),
/* 33 - 'f'       */      L('f'), 'F',      A('f'), 'f',      'F',      C('F'),
/* 34 - 'g'       */      L('g'), 'G',      A('g'), 'g',      'G',      C('G'),
/* 35 - 'h'       */      L('h'), 'H',      A('h'), 'h',      'H',      C('H'),
/* 36 - 'j'       */      L('j'), 'J',      A('j'), 'j',      'J',      C('J'),
/* 37 - 'k'       */      L('k'), 'K',      A('k'), 'k',      'K',      C('K'),
/* 38 - 'l'       */      L('l'), 'L',      A('l'), 'l',      'L',      C('L'),
/* 39 - ';'       */      L(0244), 0245, A(0244), 0244,    0245,    C('@'),
/* 40 - '\'       */      '\'',    '"',      A('\'),  '{',      '"',      C('@'),
/* 41 - '`'       */      0247,    0246,    A(0247), '\'',    0246,    C('@'),
/* 42 - l. SHIFT */      SHIFT,  SHIFT,  SHIFT,  SHIFT,  SHIFT,  SHIFT,
/* 43 - '\'       */      L(0207), 0200, A(0207), '}',    0200,    C('@'),
/* 44 - 'z'       */      L('z'), 'Z',      A('z'), 'z',      'Z',      C('Z'),
/* 45 - 'x'       */      L('x'), 'X',      A('x'), 'x',      'X',      C('X'),
/* 46 - 'c'       */      L('c'), 'C',      A('c'), 'c',      'C',      C('C'),
/* 47 - 'v'       */      L('v'), 'V',      A('v'), 'v',      'V',      C('V'),
/* 48 - 'b'       */      L('b'), 'B',      A('b'), 'b',      'B',      C('B'),
/* 49 - 'n'       */      L('n'), 'N',      A('n'), 'n',      'N',      C('N'),
/* 50 - 'm'       */      L('m'), 'M',      A('m'), 'm',      'M',      C('M'),
/* 51 - ','       */      ',',      ';',      A(','),  ';',      ';',      C('@'),
/* 52 - '.'       */      '.',      ':',      A('.'),  ':',      ':',      C('@'),
/* 53 - '/'       */      '-',      '_',      A('-'),  '-',      '-',      C('@'),
/* 54 - r. SHIFT */      SHIFT,  SHIFT,  SHIFT,  SHIFT,  SHIFT,  SHIFT,
/* 55 - '*'       */      '*',      '*',      A('*'),  '*',      '*',      C('M'),
/* 56 - ALT       */      ALT,      ALT,      ALT,      ALT,      ALT,      ALT,
/* 57 - ' '       */      ' ',      ' ',      A(' '),  ' ',      ' ',      C('@'),
/* 58 - CapsLck   */      CALOCK,  CALOCK,  CALOCK,  CALOCK,  CALOCK,  CALOCK,
/* 59 - F1        */      F1,      SF1,     AF1,     AF1,     ASF1,    CF1,
/* 60 - F2        */      F2,      SF2,     AF2,     AF2,     ASF2,    CF2,
/* 61 - F3        */      F3,      SF3,     AF3,     AF3,     ASF3,    CF3,
/* 62 - F4        */      F4,      SF4,     AF4,     AF4,     ASF4,    CF4,
/* 63 - F5        */      F5,      SF5,     AF5,     AF5,     ASF5,    CF5,
/* 64 - F6        */      F6,      SF6,     AF6,     AF6,     ASF6,    CF6,

```

```

/* 65 - F7      */      F7,      SF7,      AF7,      AF7,      ASF7,      CF7,
/* 66 - F8      */      F8,      SF8,      AF8,      AF8,      ASF8,      CF8,
/* 67 - F9      */      F9,      SF9,      AF9,      AF9,      ASF9,      CF9,
/* 68 - F10     */      F10,     SF10,     AF10,     AF10,     ASF10,     CF10,
/* 69 - NumLock */      NLOCK,   NLOCK,   NLOCK,   NLOCK,   NLOCK,   NLOCK,
/* 70 - ScrLock */      SLOCK,   SLOCK,   SLOCK,   SLOCK,   SLOCK,   SLOCK,
/* 71 - Home    */      HOME,    '7',    AHOME,    AHOME,    '7',    CHOME,
/* 72 - CurUp   */      UP,      '8',    AUP,      AUP,      '8',    CUP,
/* 73 - PgUp    */      PGUP,    '9',    APGUP,   APGUP,   '9',    CPGUP,
/* 74 - '-'     */      NMIN,    '-',    ANMIN,   ANMIN,   '-',    CNMIN,
/* 75 - Left    */      LEFT,    '4',    ALEFT,   ALEFT,   '4',    CLEFT,
/* 76 - MID     */      MID,     '5',    AMID,    AMID,    '5',    CMID,
/* 77 - Right   */      RIGHT,   '6',    ARIGHT,  ARIGHT,  '6',    CRIGHT,
/* 78 - '+'     */      PLUS,    '+',    APLUS,   APLUS,   '+',    CPLUS,
/* 79 - End     */      END,     '1',    AEND,    AEND,    '1',    CEND,
/* 80 - Down    */      DOWN,    '2',    ADOWN,   ADOWN,   '2',    CDOWN,
/* 81 - PgDown  */      PGDN,    '3',    APGDN,   APGDN,   '3',    CPGDN,
/* 82 - Insert  */      INSRT,   '0',    AINSRT,  AINSRT,  '0',    CINSRT,
/* 83 - Delete  */      0177,   '.',   A(0177), 0177,   '.',   0177,
/* 84 - Enter   */      C('M'),  C('M'), CA('M'), C('M'), C('M'), C('J'),
/* 85 - ???     */      0,        0,      0,        0,      0,        0,
/* 86 - ???     */      '<',     '>',    A('<'),  '<',     '>',    C('@'),
/* 87 - F11     */      F11,     SF11,   AF11,    AF11,    ASF11,   CF11,
/* 88 - F12     */      F12,     SF12,   AF12,    AF12,    ASF12,   CF12,
/* 89 - ???     */      0,        0,      0,        0,      0,        0,
/* 90 - ???     */      0,        0,      0,        0,      0,        0,
/* 91 - ???     */      0,        0,      0,        0,      0,        0,
/* 92 - ???     */      0,        0,      0,        0,      0,        0,
/* 93 - ???     */      0,        0,      0,        0,      0,        0,
/* 94 - ???     */      0,        0,      0,        0,      0,        0,
/* 95 - ???     */      0,        0,      0,        0,      0,        0,
/* 96 - EXT_KEY */      EXTKEY,  EXTKEY, EXTKEY,  EXTKEY,  EXTKEY,  EXTKEY,
/* 97 - ???     */      0,        0,      0,        0,      0,        0,
/* 98 - ???     */      0,        0,      0,        0,      0,        0,
/* 99 - ???     */      0,        0,      0,        0,      0,        0,
/*100 - ???     */      0,        0,      0,        0,      0,        0,
/*101 - ???     */      0,        0,      0,        0,      0,        0,
/*102 - ???     */      0,        0,      0,        0,      0,        0,
/*103 - ???     */      0,        0,      0,        0,      0,        0,
/*104 - ???     */      0,        0,      0,        0,      0,        0,
/*105 - ???     */      0,        0,      0,        0,      0,        0,
/*106 - ???     */      0,        0,      0,        0,      0,        0,
/*107 - ???     */      0,        0,      0,        0,      0,        0,
/*108 - ???     */      0,        0,      0,        0,      0,        0,
/*109 - ???     */      0,        0,      0,        0,      0,        0,
/*110 - ???     */      0,        0,      0,        0,      0,        0,
/*111 - ???     */      0,        0,      0,        0,      0,        0,
/*112 - ???     */      0,        0,      0,        0,      0,        0,
/*113 - ???     */      0,        0,      0,        0,      0,        0,
/*114 - ???     */      0,        0,      0,        0,      0,        0,
/*115 - ???     */      0,        0,      0,        0,      0,        0,
/*116 - ???     */      0,        0,      0,        0,      0,        0,
/*117 - ???     */      0,        0,      0,        0,      0,        0,
/*118 - ???     */      0,        0,      0,        0,      0,        0,
/*119 - ???     */      0,        0,      0,        0,      0,        0,
/*120 - ???     */      0,        0,      0,        0,      0,        0,
/*121 - ???     */      0,        0,      0,        0,      0,        0,
/*122 - ???     */      0,        0,      0,        0,      0,        0,
/*123 - ???     */      0,        0,      0,        0,      0,        0,
/*124 - ???     */      0,        0,      0,        0,      0,        0,
/*125 - ???     */      0,        0,      0,        0,      0,        0,
/*126 - ???     */      0,        0,      0,        0,      0,        0,
/*127 - ???     */      0,        0,      0,        0,      0,        0
};

```



```

/* Keymap for standard UK keyboard.                                Author: Darren Mason */

u16_t keymap[NR_SCAN_CODES * MAP_COLS] = {

/* scan-code      !Shift  Shift    Alt1     Alt2     Alt+Sh  Ctrl     */
/* ===== */
/* 00 - none      */      0,        0,        0,        0,        0,        0,
/* 01 - ESC       */      C('['), C('['), CA('['), CA('['), CA('['), C('['),
/* 02 - '1'       */      '1',      '!',      A('1'), A('1'), A('!'), C('A'),
/* 03 - '2'       */      '2',      '"',      A('2'), A('2'), A('@'), C('@'),
/* 04 - '3'       */      '3',      156,      A('3'), A('3'), A(156), C('C'),
/* 05 - '4'       */      '4',      '$',      A('4'), A('4'), A('$'), C('D'),
/* 06 - '5'       */      '5',      '%',      A('5'), A('5'), A('%'), C('E'),
/* 07 - '6'       */      '6',      '^',      A('6'), A('6'), A('^'), C('^'),
/* 08 - '7'       */      '7',      '&',      A('7'), A('7'), A('&'), C('G'),
/* 09 - '8'       */      '8',      '*',      A('8'), A('8'), A('*'), C('H'),
/* 10 - '9'       */      '9',      '(',      A('9'), A('9'), A('('), C('I'),
/* 11 - '0'       */      '0',      ')',      A('0'), A('0'), A(')'), C('@'),
/* 12 - '-'       */      '-',      '_',      A('-'), A('-'), A('_'), C('_'),
/* 13 - '='       */      '=',      '+',      A('='), A('='), A('+'), C('@'),
/* 14 - BS        */      C('H'), C('H'), CA('H'), CA('H'), CA('H'), 0177,
/* 15 - TAB       */      C('I'), C('I'), CA('I'), CA('I'), CA('I'), C('I'),
/* 16 - 'q'       */      L('q'), 'Q',      A('q'), A('q'), A('Q'), C('Q'),
/* 17 - 'w'       */      L('w'), 'W',      A('w'), A('w'), A('W'), C('W'),
/* 18 - 'e'       */      L('e'), 'E',      A('e'), A('e'), A('E'), C('E'),
/* 19 - 'r'       */      L('r'), 'R',      A('r'), A('r'), A('R'), C('R'),
/* 20 - 't'       */      L('t'), 'T',      A('t'), A('t'), A('T'), C('T'),
/* 21 - 'y'       */      L('y'), 'Y',      A('y'), A('y'), A('Y'), C('Y'),
/* 22 - 'u'       */      L('u'), 'U',      A('u'), A('u'), A('U'), C('U'),
/* 23 - 'i'       */      L('i'), 'I',      A('i'), A('i'), A('I'), C('I'),
/* 24 - 'o'       */      L('o'), 'O',      A('o'), A('o'), A('O'), C('O'),
/* 25 - 'p'       */      L('p'), 'P',      A('p'), A('p'), A('P'), C('P'),
/* 26 - '['       */      '[',      '{',      A('['), A('['), A('{'), C('['),
/* 27 - ']'       */      ']',      '}',      A(']'), A(']'), A('}'), C(']'),
/* 28 - CR/LF     */      C('M'), C('M'), CA('M'), CA('M'), CA('M'), C('J'),
/* 29 - Ctrl      */      CTRL,   CTRL,   CTRL,   CTRL,   CTRL,   CTRL,
/* 30 - 'a'       */      L('a'), 'A',      A('a'), A('a'), A('A'), C('A'),
/* 31 - 's'       */      L('s'), 'S',      A('s'), A('s'), A('S'), C('S'),
/* 32 - 'd'       */      L('d'), 'D',      A('d'), A('d'), A('D'), C('D'),
/* 33 - 'f'       */      L('f'), 'F',      A('f'), A('f'), A('F'), C('F'),
/* 34 - 'g'       */      L('g'), 'G',      A('g'), A('g'), A('G'), C('G'),
/* 35 - 'h'       */      L('h'), 'H',      A('h'), A('h'), A('H'), C('H'),
/* 36 - 'j'       */      L('j'), 'J',      A('j'), A('j'), A('J'), C('J'),
/* 37 - 'k'       */      L('k'), 'K',      A('k'), A('k'), A('K'), C('K'),
/* 38 - 'l'       */      L('l'), 'L',      A('l'), A('l'), A('L'), C('L'),
/* 39 - ';'       */      ';',      ':',      A(';'), A(';'), A(':'), C('@'),
/* 40 - '\'       */      '\',      '@',      A('\'), A('\'), A('@'), C('@'),
/* 41 - '`'       */      '`',      '~',      A('`'), A('`'), A('~'), C('@'),
/* 42 - l. SHIFT */      SHIFT,  SHIFT,  SHIFT,  SHIFT,  SHIFT,  SHIFT,
/* 43 - '#'       */      '#',      '~',      A('#'), A('#'), A('~'), C('#'),
/* 44 - 'z'       */      L('z'), 'Z',      A('z'), A('z'), A('Z'), C('Z'),
/* 45 - 'x'       */      L('x'), 'X',      A('x'), A('x'), A('X'), C('X'),
/* 46 - 'c'       */      L('c'), 'C',      A('c'), A('c'), A('C'), C('C'),
/* 47 - 'v'       */      L('v'), 'V',      A('v'), A('v'), A('V'), C('V'),
/* 48 - 'b'       */      L('b'), 'B',      A('b'), A('b'), A('B'), C('B'),
/* 49 - 'n'       */      L('n'), 'N',      A('n'), A('n'), A('N'), C('N'),
/* 50 - 'm'       */      L('m'), 'M',      A('m'), A('m'), A('M'), C('M'),
/* 51 - '<'       */      '<',      '<',      A('<'), A('<'), A('<'), C('@'),
/* 52 - '>'       */      '>',      '>',      A('>'), A('>'), A('>'), C('@'),
/* 53 - '/'       */      '/',      '?',      A('/'), A('/'), A('?'), C('@'),
/* 54 - r. SHIFT */      SHIFT,  SHIFT,  SHIFT,  SHIFT,  SHIFT,  SHIFT,
/* 55 - '*'       */      '*',      '*',      A('*'), A('*'), A('*'), C('@'),
/* 56 - ALT       */      ALT,     ALT,     ALT,     ALT,     ALT,     ALT,
/* 57 - ' '       */      ' ',      ' ',      A(' '), A(' '), A(' '), C('@'),
/* 58 - CapsLock  */      CALOCK,  CALOCK,  CALOCK,  CALOCK,  CALOCK,  CALOCK,
/* 59 - F1        */      F1,      SF1,     AF1,     AF1,     ASF1,    CF1,
/* 60 - F2        */      F2,      SF2,     AF2,     AF2,     ASF2,    CF2,
/* 61 - F3        */      F3,      SF3,     AF3,     AF3,     ASF3,    CF3,
/* 62 - F4        */      F4,      SF4,     AF4,     AF4,     ASF4,    CF4,
/* 63 - F5        */      F5,      SF5,     AF5,     AF5,     ASF5,    CF5,
/* 64 - F6        */      F6,      SF6,     AF6,     AF6,     ASF6,    CF6,
/* 65 - F7        */      F7,      SF7,     AF7,     AF7,     ASF7,    CF7,
/* 66 - F8        */      F8,      SF8,     AF8,     AF8,     ASF8,    CF8,
/* 67 - F9        */      F9,      SF9,     AF9,     AF9,     ASF9,    CF9,

```

```

/* 68 - F10 */ F10, SF10, AF10, AF10, ASF10, CF10,
/* 69 - NumLock */ NLOCK, NLOCK, NLOCK, NLOCK, NLOCK, NLOCK,
/* 70 - ScrLock */ SLOCK, SLOCK, SLOCK, SLOCK, SLOCK, SLOCK,
/* 71 - Home */ HOME, '7', AHOME, AHOME, A('7'), CHOME,
/* 72 - CurUp */ UP, '8', AUP, AUP, A('8'), CUP,
/* 73 - PgUp */ PGUP, '9', APGUP, APGUP, A('9'), CPGUP,
/* 74 - '-' */ NMIN, '-', ANMIN, ANMIN, A('-'), CNMIN,
/* 75 - Left */ LEFT, '4', ALEFT, ALEFT, A('4'), CLEFT,
/* 76 - MID */ MID, '5', AMID, AMID, A('5'), CMID,
/* 77 - Right */ RIGHT, '6', ARIGHT, ARIGHT, A('6'), CRIGHT,
/* 78 - '+' */ PLUS, '+', APLUS, APLUS, A('+'), CPLUS,
/* 79 - End */ END, '1', AEND, AEND, A('1'), CEND,
/* 80 - Down */ DOWN, '2', ADOWN, ADOWN, A('2'), CDOWN,
/* 81 - PgDown */ PGDN, '3', APGDN, APGDN, A('3'), CPGDN,
/* 82 - Insert */ INSRT, '0', AINSRT, AINSRT, A('0'), CINSRT,
/* 83 - Delete */ 0177, '.', A(0177), A(0177), A('.'), 0177,
/* 84 - Enter */ C('M'), C('M'), CA('M'), CA('M'), CA('M'), C('J'),
/* 85 - ??? */ 0, 0, 0, 0, 0, 0,
/* 86 - ??? */ '\\', '|', A('\\'), A('|'), A('|'), C('@'),
/* 87 - F11 */ F11, SF11, AF11, AF11, ASF11, CF11,
/* 88 - F12 */ F12, SF12, AF12, AF12, ASF12, CF12,
/* 89 - ??? */ 0, 0, 0, 0, 0, 0,
/* 90 - ??? */ 0, 0, 0, 0, 0, 0,
/* 91 - ??? */ 0, 0, 0, 0, 0, 0,
/* 92 - ??? */ 0, 0, 0, 0, 0, 0,
/* 93 - ??? */ 0, 0, 0, 0, 0, 0,
/* 94 - ??? */ 0, 0, 0, 0, 0, 0,
/* 95 - ??? */ 0, 0, 0, 0, 0, 0,
/* 96 - EXT_KEY */ EXTKEY, EXTKEY, EXTKEY, EXTKEY, EXTKEY, EXTKEY,
/* 97 - ??? */ 0, 0, 0, 0, 0, 0,
/* 98 - ??? */ 0, 0, 0, 0, 0, 0,
/* 99 - ??? */ 0, 0, 0, 0, 0, 0,
/*100 - ??? */ 0, 0, 0, 0, 0, 0,
/*101 - ??? */ 0, 0, 0, 0, 0, 0,
/*102 - ??? */ 0, 0, 0, 0, 0, 0,
/*103 - ??? */ 0, 0, 0, 0, 0, 0,
/*104 - ??? */ 0, 0, 0, 0, 0, 0,
/*105 - ??? */ 0, 0, 0, 0, 0, 0,
/*106 - ??? */ 0, 0, 0, 0, 0, 0,
/*107 - ??? */ 0, 0, 0, 0, 0, 0,
/*108 - ??? */ 0, 0, 0, 0, 0, 0,
/*109 - ??? */ 0, 0, 0, 0, 0, 0,
/*110 - ??? */ 0, 0, 0, 0, 0, 0,
/*111 - ??? */ 0, 0, 0, 0, 0, 0,
/*112 - ??? */ 0, 0, 0, 0, 0, 0,
/*113 - ??? */ 0, 0, 0, 0, 0, 0,
/*114 - ??? */ 0, 0, 0, 0, 0, 0,
/*115 - ??? */ 0, 0, 0, 0, 0, 0,
/*116 - ??? */ 0, 0, 0, 0, 0, 0,
/*117 - ??? */ 0, 0, 0, 0, 0, 0,
/*118 - ??? */ 0, 0, 0, 0, 0, 0,
/*119 - ??? */ 0, 0, 0, 0, 0, 0,
/*120 - ??? */ 0, 0, 0, 0, 0, 0,
/*121 - ??? */ 0, 0, 0, 0, 0, 0,
/*122 - ??? */ 0, 0, 0, 0, 0, 0,
/*123 - ??? */ 0, 0, 0, 0, 0, 0,
/*124 - ??? */ 0, 0, 0, 0, 0, 0,
/*125 - ??? */ 0, 0, 0, 0, 0, 0,
/*126 - ??? */ 0, 0, 0, 0, 0, 0,
/*127 - ??? */ 0, 0, 0, 0, 0, 0
};

```

```

/* Keymap for US MF-2 keyboard. */

u16_t keymap[NR_SCAN_CODES * MAP_COLS] = {

/* scan-code      !Shift  Shift    Alt1     Alt2     Alt+Sh  Ctrl     */
/* ===== */
/* 00 - none      */      0,      0,      0,      0,      0,      0,
/* 01 - ESC       */      C('['), C('['), CA('['), CA('['), CA('['), C('['),
/* 02 - '1'       */      '1',    '!',    A('1'), A('1'), A('!'), C('A'),
/* 03 - '2'       */      '2',    '@',    A('2'), A('2'), A('@'), C('@'),
/* 04 - '3'       */      '3',    '#',    A('3'), A('3'), A('#'), C('C'),
/* 05 - '4'       */      '4',    '$',    A('4'), A('4'), A('$'), C('D'),
/* 06 - '5'       */      '5',    '%',    A('5'), A('5'), A('%'), C('E'),
/* 07 - '6'       */      '6',    '^',    A('6'), A('6'), A('^'), C('^'),
/* 08 - '7'       */      '7',    '&',    A('7'), A('7'), A('&'), C('G'),
/* 09 - '8'       */      '8',    '*',    A('8'), A('8'), A('*'), C('H'),
/* 10 - '9'       */      '9',    '(',    A('9'), A('9'), A('('), C('I'),
/* 11 - '0'       */      '0',    ')',    A('0'), A('0'), A(')'), C('@'),
/* 12 - '-'       */      '-',    '_',    A('-'), A('-'), A('_'), C('_'),
/* 13 - '='       */      '=',    '+',    A('='), A('='), A('+'), C('@'),
/* 14 - BS        */      C('H'), C('H'), CA('H'), CA('H'), CA('H'), 0177,
/* 15 - TAB       */      C('I'), C('I'), CA('I'), CA('I'), CA('I'), C('I'),
/* 16 - 'q'       */      L('q'), 'Q',    A('q'), A('q'), A('Q'), C('Q'),
/* 17 - 'w'       */      L('w'), 'W',    A('w'), A('w'), A('W'), C('W'),
/* 18 - 'e'       */      L('e'), 'E',    A('e'), A('e'), A('E'), C('E'),
/* 19 - 'r'       */      L('r'), 'R',    A('r'), A('r'), A('R'), C('R'),
/* 20 - 't'       */      L('t'), 'T',    A('t'), A('t'), A('T'), C('T'),
/* 21 - 'y'       */      L('y'), 'Y',    A('y'), A('y'), A('Y'), C('Y'),
/* 22 - 'u'       */      L('u'), 'U',    A('u'), A('u'), A('U'), C('U'),
/* 23 - 'i'       */      L('i'), 'I',    A('i'), A('i'), A('I'), C('I'),
/* 24 - 'o'       */      L('o'), 'O',    A('o'), A('o'), A('O'), C('O'),
/* 25 - 'p'       */      L('p'), 'P',    A('p'), A('p'), A('P'), C('P'),
/* 26 - '['       */      '[',    '{',    A('['), A('['), A('{'), C('['),
/* 27 - ']'       */      ']',    '}',    A(']'), A(']'), A('}'), C(']'),
/* 28 - CR/LF     */      C('M'), C('M'), CA('M'), CA('M'), CA('M'), C('J'),
/* 29 - Ctrl      */      CTRL,  CTRL,  CTRL,  CTRL,  CTRL,  CTRL,
/* 30 - 'a'       */      L('a'), 'A',    A('a'), A('a'), A('A'), C('A'),
/* 31 - 's'       */      L('s'), 'S',    A('s'), A('s'), A('S'), C('S'),
/* 32 - 'd'       */      L('d'), 'D',    A('d'), A('d'), A('D'), C('D'),
/* 33 - 'f'       */      L('f'), 'F',    A('f'), A('f'), A('F'), C('F'),
/* 34 - 'g'       */      L('g'), 'G',    A('g'), A('g'), A('G'), C('G'),
/* 35 - 'h'       */      L('h'), 'H',    A('h'), A('h'), A('H'), C('H'),
/* 36 - 'j'       */      L('j'), 'J',    A('j'), A('j'), A('J'), C('J'),
/* 37 - 'k'       */      L('k'), 'K',    A('k'), A('k'), A('K'), C('K'),
/* 38 - 'l'       */      L('l'), 'L',    A('l'), A('l'), A('L'), C('L'),
/* 39 - ';'       */      ';',    ':',    A(';'), A(';'), A(':'), C('@'),
/* 40 - '\'       */      '\',    '"',    A('\'), A('\'), A('"'), C('@'),
/* 41 - '`'       */      '`',    '~',    A('`'), A('`'), A('~'), C('@'),
/* 42 - l. SHIFT */      SHIFT, SHIFT, SHIFT, SHIFT, SHIFT, SHIFT,
/* 43 - '\\       */      '\\',  '|',    A('\\'), A('\\'), A('|'), C('\\'),
/* 44 - 'z'       */      L('z'), 'Z',    A('z'), A('z'), A('Z'), C('Z'),
/* 45 - 'x'       */      L('x'), 'X',    A('x'), A('x'), A('X'), C('X'),
/* 46 - 'c'       */      L('c'), 'C',    A('c'), A('c'), A('C'), C('C'),
/* 47 - 'v'       */      L('v'), 'V',    A('v'), A('v'), A('V'), C('V'),
/* 48 - 'b'       */      L('b'), 'B',    A('b'), A('b'), A('B'), C('B'),
/* 49 - 'n'       */      L('n'), 'N',    A('n'), A('n'), A('N'), C('N'),
/* 50 - 'm'       */      L('m'), 'M',    A('m'), A('m'), A('M'), C('M'),
/* 51 - ','       */      ',',    '<',    A(','), A(','), A('<'), C('@'),
/* 52 - '.'       */      '.',    '>',    A('.'), A('.'), A('>'), C('@'),
/* 53 - '/'       */      '/',    '?',    A('/'), A('/'), A('?'), C('@'),
/* 54 - r. SHIFT */      SHIFT, SHIFT, SHIFT, SHIFT, SHIFT, SHIFT,
/* 55 - '*'       */      '*',    '*',    A('*'), A('*'), A('*'), C('@'),
/* 56 - ALT       */      ALT,   ALT,   ALT,   ALT,   ALT,   ALT,
/* 57 - ' '       */      ' ',   ' ',   A(' '), A(' '), A(' '), C('@'),
/* 58 - CapsLock  */      CALOCK, CALOCK, CALOCK, CALOCK, CALOCK, CALOCK,
/* 59 - F1        */      F1,    SF1,   AF1,   AF1,   ASF1,  CF1,
/* 60 - F2        */      F2,    SF2,   AF2,   AF2,   ASF2,  CF2,
/* 61 - F3        */      F3,    SF3,   AF3,   AF3,   ASF3,  CF3,
/* 62 - F4        */      F4,    SF4,   AF4,   AF4,   ASF4,  CF4,
/* 63 - F5        */      F5,    SF5,   AF5,   AF5,   ASF5,  CF5,
/* 64 - F6        */      F6,    SF6,   AF6,   AF6,   ASF6,  CF6,
/* 65 - F7        */      F7,    SF7,   AF7,   AF7,   ASF7,  CF7,
/* 66 - F8        */      F8,    SF8,   AF8,   AF8,   ASF8,  CF8,
/* 67 - F9        */      F9,    SF9,   AF9,   AF9,   ASF9,  CF9,

```

```

/* 68 - F10 */ F10, SF10, AF10, AF10, ASF10, CF10,
/* 69 - NumLock */ NLOCK, NLOCK, NLOCK, NLOCK, NLOCK, NLOCK,
/* 70 - ScrLock */ SLOCK, SLOCK, SLOCK, SLOCK, SLOCK, SLOCK,
/* 71 - Home */ HOME, '7', AHOME, AHOME, A('7'), CHOME,
/* 72 - CurUp */ UP, '8', AUP, AUP, A('8'), CUP,
/* 73 - PgUp */ PGUP, '9', APGUP, APGUP, A('9'), CPGUP,
/* 74 - '-' */ NMIN, '-', ANMIN, ANMIN, A('-'), CNMIN,
/* 75 - Left */ LEFT, '4', ALEFT, ALEFT, A('4'), CLEFT,
/* 76 - MID */ MID, '5', AMID, AMID, A('5'), CMID,
/* 77 - Right */ RIGHT, '6', ARIGHT, ARIGHT, A('6'), CRIGHT,
/* 78 - '+' */ PLUS, '+', APLUS, APLUS, A('+'), CPLUS,
/* 79 - End */ END, '1', AEND, AEND, A('1'), CEND,
/* 80 - Down */ DOWN, '2', ADOWN, ADOWN, A('2'), CDOWN,
/* 81 - PgDown */ PGDN, '3', APGDN, APGDN, A('3'), CPGDN,
/* 82 - Insert */ INSRT, '0', AINSRT, AINSRT, A('0'), CINSRT,
/* 83 - Delete */ 0177, '.', A(0177), A(0177), A('.'), 0177,
/* 84 - Enter */ C('M'), C('M'), CA('M'), CA('M'), CA('M'), C('J'),
/* 85 - ??? */ 0, 0, 0, 0, 0, 0,
/* 86 - ??? */ '<', '>', A('<'), A('|'), A('>'), C('@'),
/* 87 - F11 */ F11, SF11, AF11, AF11, ASF11, CF11,
/* 88 - F12 */ F12, SF12, AF12, AF12, ASF12, CF12,
/* 89 - ??? */ 0, 0, 0, 0, 0, 0,
/* 90 - ??? */ 0, 0, 0, 0, 0, 0,
/* 91 - ??? */ 0, 0, 0, 0, 0, 0,
/* 92 - ??? */ 0, 0, 0, 0, 0, 0,
/* 93 - ??? */ 0, 0, 0, 0, 0, 0,
/* 94 - ??? */ 0, 0, 0, 0, 0, 0,
/* 95 - ??? */ 0, 0, 0, 0, 0, 0,
/* 96 - EXT_KEY */ EXTKEY, EXTKEY, EXTKEY, EXTKEY, EXTKEY, EXTKEY,
/* 97 - ??? */ 0, 0, 0, 0, 0, 0,
/* 98 - ??? */ 0, 0, 0, 0, 0, 0,
/* 99 - ??? */ 0, 0, 0, 0, 0, 0,
/*100 - ??? */ 0, 0, 0, 0, 0, 0,
/*101 - ??? */ 0, 0, 0, 0, 0, 0,
/*102 - ??? */ 0, 0, 0, 0, 0, 0,
/*103 - ??? */ 0, 0, 0, 0, 0, 0,
/*104 - ??? */ 0, 0, 0, 0, 0, 0,
/*105 - ??? */ 0, 0, 0, 0, 0, 0,
/*106 - ??? */ 0, 0, 0, 0, 0, 0,
/*107 - ??? */ 0, 0, 0, 0, 0, 0,
/*108 - ??? */ 0, 0, 0, 0, 0, 0,
/*109 - ??? */ 0, 0, 0, 0, 0, 0,
/*110 - ??? */ 0, 0, 0, 0, 0, 0,
/*111 - ??? */ 0, 0, 0, 0, 0, 0,
/*112 - ??? */ 0, 0, 0, 0, 0, 0,
/*113 - ??? */ 0, 0, 0, 0, 0, 0,
/*114 - ??? */ 0, 0, 0, 0, 0, 0,
/*115 - ??? */ 0, 0, 0, 0, 0, 0,
/*116 - ??? */ 0, 0, 0, 0, 0, 0,
/*117 - ??? */ 0, 0, 0, 0, 0, 0,
/*118 - ??? */ 0, 0, 0, 0, 0, 0,
/*119 - ??? */ 0, 0, 0, 0, 0, 0,
/*120 - ??? */ 0, 0, 0, 0, 0, 0,
/*121 - ??? */ 0, 0, 0, 0, 0, 0,
/*122 - ??? */ 0, 0, 0, 0, 0, 0,
/*123 - ??? */ 0, 0, 0, 0, 0, 0,
/*124 - ??? */ 0, 0, 0, 0, 0, 0,
/*125 - ??? */ 0, 0, 0, 0, 0, 0,
/*126 - ??? */ 0, 0, 0, 0, 0, 0,
/*127 - ??? */ 0, 0, 0, 0, 0, 0
};

```

```
/* Keymap for US MF-2 keyboard with the Caps Lock and Control key swapped. */
```

```
u16_t keymap[NR_SCAN_CODES * MAP_COLS] = {
```

```
/* scan-code      !Shift  Shift    Alt1     Alt2     Alt+Sh  Ctrl     */
/* ===== */
/* 00 - none      */      0,        0,        0,        0,        0,        0,
/* 01 - ESC       */      C('['), C('['), CA('['), CA('['), CA('['), C('['),
/* 02 - '1'       */      '1',      '!',      A('1'), A('1'), A('!'), C('A'),
/* 03 - '2'       */      '2',      '@',      A('2'), A('2'), A('@'), C('@'),
/* 04 - '3'       */      '3',      '#',      A('3'), A('3'), A('#'), C('C'),
/* 05 - '4'       */      '4',      '$',      A('4'), A('4'), A('$'), C('D'),
/* 06 - '5'       */      '5',      '%',      A('5'), A('5'), A('%'), C('E'),
/* 07 - '6'       */      '6',      '^',      A('6'), A('6'), A('^'), C('^'),
/* 08 - '7'       */      '7',      '&',      A('7'), A('7'), A('&'), C('G'),
/* 09 - '8'       */      '8',      '*',      A('8'), A('8'), A('*'), C('H'),
/* 10 - '9'       */      '9',      '(',      A('9'), A('9'), A('('), C('I'),
/* 11 - '0'       */      '0',      ')',      A('0'), A('0'), A(')'), C('@'),
/* 12 - '-'       */      '-',      '_',      A('-'), A('-'), A('_'), C('_'),
/* 13 - '='       */      '=',      '+',      A('='), A('='), A('+'), C('@'),
/* 14 - BS        */      C('H'), C('H'), CA('H'), CA('H'), CA('H'), 0177,
/* 15 - TAB       */      C('I'), C('I'), CA('I'), CA('I'), CA('I'), C('I'),
/* 16 - 'q'       */      L('q'), 'Q',      A('q'), A('q'), A('Q'), C('Q'),
/* 17 - 'w'       */      L('w'), 'W',      A('w'), A('w'), A('W'), C('W'),
/* 18 - 'e'       */      L('e'), 'E',      A('e'), A('e'), A('E'), C('E'),
/* 19 - 'r'       */      L('r'), 'R',      A('r'), A('r'), A('R'), C('R'),
/* 20 - 't'       */      L('t'), 'T',      A('t'), A('t'), A('T'), C('T'),
/* 21 - 'y'       */      L('y'), 'Y',      A('y'), A('y'), A('Y'), C('Y'),
/* 22 - 'u'       */      L('u'), 'U',      A('u'), A('u'), A('U'), C('U'),
/* 23 - 'i'       */      L('i'), 'I',      A('i'), A('i'), A('I'), C('I'),
/* 24 - 'o'       */      L('o'), 'O',      A('o'), A('o'), A('O'), C('O'),
/* 25 - 'p'       */      L('p'), 'P',      A('p'), A('p'), A('P'), C('P'),
/* 26 - '['       */      '[',      '{',      A('['), A('['), A('{'), C('['),
/* 27 - ']'       */      ']',      '}',      A(']'), A(']'), A('}'), C(']'),
/* 28 - CR/LF     */      C('M'), C('M'), CA('M'), CA('M'), CA('M'), C('J'),
/* 29 - Ctrl      */      CALOCK, CALOCK, CALOCK, CALOCK, CALOCK, CALOCK,
/* 30 - 'a'       */      L('a'), 'A',      A('a'), A('a'), A('A'), C('A'),
/* 31 - 's'       */      L('s'), 'S',      A('s'), A('s'), A('S'), C('S'),
/* 32 - 'd'       */      L('d'), 'D',      A('d'), A('d'), A('D'), C('D'),
/* 33 - 'f'       */      L('f'), 'F',      A('f'), A('f'), A('F'), C('F'),
/* 34 - 'g'       */      L('g'), 'G',      A('g'), A('g'), A('G'), C('G'),
/* 35 - 'h'       */      L('h'), 'H',      A('h'), A('h'), A('H'), C('H'),
/* 36 - 'j'       */      L('j'), 'J',      A('j'), A('j'), A('J'), C('J'),
/* 37 - 'k'       */      L('k'), 'K',      A('k'), A('k'), A('K'), C('K'),
/* 38 - 'l'       */      L('l'), 'L',      A('l'), A('l'), A('L'), C('L'),
/* 39 - ';'       */      ';',      ':',      A(';'), A(';'), A(':'), C('@'),
/* 40 - '\'       */      '\',      '"',      A('\'), A('\'), A('"'), C('@'),
/* 41 - '`'       */      '`',      '~',      A('`'), A('`'), A('~'), C('@'),
/* 42 - l. SHIFT */      SHIFT, SHIFT, SHIFT, SHIFT, SHIFT, SHIFT,
/* 43 - '\\       */      '\\', '|',      A('\\'), A('\\'), A('|'), C('\\'),
/* 44 - 'z'       */      L('z'), 'Z',      A('z'), A('z'), A('Z'), C('Z'),
/* 45 - 'x'       */      L('x'), 'X',      A('x'), A('x'), A('X'), C('X'),
/* 46 - 'c'       */      L('c'), 'C',      A('c'), A('c'), A('C'), C('C'),
/* 47 - 'v'       */      L('v'), 'V',      A('v'), A('v'), A('V'), C('V'),
/* 48 - 'b'       */      L('b'), 'B',      A('b'), A('b'), A('B'), C('B'),
/* 49 - 'n'       */      L('n'), 'N',      A('n'), A('n'), A('N'), C('N'),
/* 50 - 'm'       */      L('m'), 'M',      A('m'), A('m'), A('M'), C('M'),
/* 51 - ','       */      ',',      '<',      A(','), A(','), A('<'), C('@'),
/* 52 - '.'       */      '.',      '>',      A('.'), A('.'), A('>'), C('@'),
/* 53 - '/'       */      '/',      '?',      A('/'), A('/'), A('?'), C('@'),
/* 54 - r. SHIFT */      SHIFT, SHIFT, SHIFT, SHIFT, SHIFT, SHIFT,
/* 55 - '*'       */      '*',      '*',      A('*'), A('*'), A('*'), C('@'),
/* 56 - ALT       */      ALT,      ALT,      ALT,      ALT,      ALT,      ALT,
/* 57 - ' '       */      ' ',      ' ',      A(' '), A(' '), A(' '), C('@'),
/* 58 - CapsLck   */      CTRL,      CTRL,      CTRL,      CTRL,      CTRL,      CTRL,
/* 59 - F1        */      F1,        SF1,       AF1,       AF1,       ASF1,      CF1,
/* 60 - F2        */      F2,        SF2,       AF2,       AF2,       ASF2,      CF2,
/* 61 - F3        */      F3,        SF3,       AF3,       AF3,       ASF3,      CF3,
/* 62 - F4        */      F4,        SF4,       AF4,       AF4,       ASF4,      CF4,
/* 63 - F5        */      F5,        SF5,       AF5,       AF5,       ASF5,      CF5,
/* 64 - F6        */      F6,        SF6,       AF6,       AF6,       ASF6,      CF6,
/* 65 - F7        */      F7,        SF7,       AF7,       AF7,       ASF7,      CF7,
/* 66 - F8        */      F8,        SF8,       AF8,       AF8,       ASF8,      CF8,
/* 67 - F9        */      F9,        SF9,       AF9,       AF9,       ASF9,      CF9,
```

```

/* 68 - F10 */ F10, SF10, AF10, AF10, ASF10, CF10,
/* 69 - NumLock */ NLOCK, NLOCK, NLOCK, NLOCK, NLOCK, NLOCK,
/* 70 - ScrLock */ SLOCK, SLOCK, SLOCK, SLOCK, SLOCK, SLOCK,
/* 71 - Home */ HOME, '7', AHOME, AHOME, A('7'), CHOME,
/* 72 - CurUp */ UP, '8', AUP, AUP, A('8'), CUP,
/* 73 - PgUp */ PGUP, '9', APGUP, APGUP, A('9'), CPGUP,
/* 74 - '-' */ NMIN, '-', ANMIN, ANMIN, A('-'), CNMIN,
/* 75 - Left */ LEFT, '4', ALEFT, ALEFT, A('4'), CLEFT,
/* 76 - MID */ MID, '5', AMID, AMID, A('5'), CMID,
/* 77 - Right */ RIGHT, '6', ARIGHT, ARIGHT, A('6'), CRIGHT,
/* 78 - '+' */ PLUS, '+', APLUS, APLUS, A('+'), CPLUS,
/* 79 - End */ END, '1', AEND, AEND, A('1'), CEND,
/* 80 - Down */ DOWN, '2', ADOWN, ADOWN, A('2'), CDOWN,
/* 81 - PgDown */ PGDN, '3', APGDN, APGDN, A('3'), CPGDN,
/* 82 - Insert */ INSRT, '0', AINSRT, AINSRT, A('0'), CINSRT,
/* 83 - Delete */ 0177, '.', A(0177), A(0177), A('.'), 0177,
/* 84 - Enter */ C('M'), C('M'), CA('M'), CA('M'), CA('M'), C('J'),
/* 85 - ??? */ 0, 0, 0, 0, 0, 0,
/* 86 - ??? */ '<', '>', A('<'), A('|'), A('>'), C('@'),
/* 87 - F11 */ F11, SF11, AF11, AF11, ASF11, CF11,
/* 88 - F12 */ F12, SF12, AF12, AF12, ASF12, CF12,
/* 89 - ??? */ 0, 0, 0, 0, 0, 0,
/* 90 - ??? */ 0, 0, 0, 0, 0, 0,
/* 91 - ??? */ 0, 0, 0, 0, 0, 0,
/* 92 - ??? */ 0, 0, 0, 0, 0, 0,
/* 93 - ??? */ 0, 0, 0, 0, 0, 0,
/* 94 - ??? */ 0, 0, 0, 0, 0, 0,
/* 95 - ??? */ 0, 0, 0, 0, 0, 0,
/* 96 - EXT_KEY */ EXTKEY, EXTKEY, EXTKEY, EXTKEY, EXTKEY, EXTKEY,
/* 97 - ??? */ 0, 0, 0, 0, 0, 0,
/* 98 - ??? */ 0, 0, 0, 0, 0, 0,
/* 99 - ??? */ 0, 0, 0, 0, 0, 0,
/*100 - ??? */ 0, 0, 0, 0, 0, 0,
/*101 - ??? */ 0, 0, 0, 0, 0, 0,
/*102 - ??? */ 0, 0, 0, 0, 0, 0,
/*103 - ??? */ 0, 0, 0, 0, 0, 0,
/*104 - ??? */ 0, 0, 0, 0, 0, 0,
/*105 - ??? */ 0, 0, 0, 0, 0, 0,
/*106 - ??? */ 0, 0, 0, 0, 0, 0,
/*107 - ??? */ 0, 0, 0, 0, 0, 0,
/*108 - ??? */ 0, 0, 0, 0, 0, 0,
/*109 - ??? */ 0, 0, 0, 0, 0, 0,
/*110 - ??? */ 0, 0, 0, 0, 0, 0,
/*111 - ??? */ 0, 0, 0, 0, 0, 0,
/*112 - ??? */ 0, 0, 0, 0, 0, 0,
/*113 - ??? */ 0, 0, 0, 0, 0, 0,
/*114 - ??? */ 0, 0, 0, 0, 0, 0,
/*115 - ??? */ 0, 0, 0, 0, 0, 0,
/*116 - ??? */ 0, 0, 0, 0, 0, 0,
/*117 - ??? */ 0, 0, 0, 0, 0, 0,
/*118 - ??? */ 0, 0, 0, 0, 0, 0,
/*119 - ??? */ 0, 0, 0, 0, 0, 0,
/*120 - ??? */ 0, 0, 0, 0, 0, 0,
/*121 - ??? */ 0, 0, 0, 0, 0, 0,
/*122 - ??? */ 0, 0, 0, 0, 0, 0,
/*123 - ??? */ 0, 0, 0, 0, 0, 0,
/*124 - ??? */ 0, 0, 0, 0, 0, 0,
/*125 - ??? */ 0, 0, 0, 0, 0, 0,
/*126 - ??? */ 0, 0, 0, 0, 0, 0,
/*127 - ??? */ 0, 0, 0, 0, 0, 0
};

```

## Table of Contents

1	Makefile.....	sheets	1 to	1 ( 1)	pages	1- 1	46 lines
2	drivers.h.....	sheets	2 to	2 ( 1)	pages	2- 2	34 lines
3	Makefile.....	sheets	3 to	3 ( 1)	pages	3- 3	48 lines
4	at_wini.c.....	sheets	4 to	37 (34)	pages	4- 37	2486 lines
5	at_wini.h.....	sheets	38 to	38 ( 1)	pages	38- 38	9 lines
6	Makefile.....	sheets	39 to	39 ( 1)	pages	39- 39	47 lines
7	bios_wini.c.....	sheets	40 to	46 ( 7)	pages	40- 46	511 lines
8	Makefile.....	sheets	47 to	47 ( 1)	pages	47- 47	49 lines
9	cmos.c.....	sheets	48 to	51 ( 4)	pages	48- 51	266 lines
10	3c503.c.....	sheets	52 to	54 ( 3)	pages	52- 54	199 lines
11	3c503.h.....	sheets	55 to	55 ( 1)	pages	55- 55	65 lines
12	Makefile.....	sheets	56 to	56 ( 1)	pages	56- 56	41 lines
13	dp8390.c.....	sheets	57 to	83 (27)	pages	57- 83	1983 lines
14	dp8390.h.....	sheets	84 to	88 ( 5)	pages	84- 88	298 lines
15	local.h.....	sheets	89 to	89 ( 1)	pages	89- 89	31 lines
16	ne2000.c.....	sheets	90 to	94 ( 5)	pages	90- 94	331 lines
17	ne2000.h.....	sheets	95 to	95 ( 1)	pages	95- 95	29 lines
18	rtl8029.c.....	sheets	96 to	101 ( 6)	pages	96-101	379 lines
19	rtl8029.h.....	sheets	102 to	102 ( 1)	pages	102-102	16 lines
20	wdeth.c.....	sheets	103 to	107 ( 5)	pages	103-107	363 lines
21	wdeth.h.....	sheets	108 to	109 ( 2)	pages	108-109	95 lines
22	3c501.c.....	sheets	110 to	115 ( 6)	pages	110-115	421 lines
23	3c501.h.....	sheets	116 to	116 ( 1)	pages	116-116	72 lines
24	3c503.c.....	sheets	117 to	119 ( 3)	pages	117-119	168 lines
25	3c503.h.....	sheets	120 to	120 ( 1)	pages	120-120	73 lines
26	3c509.c.....	sheets	121 to	129 ( 9)	pages	121-129	596 lines
27	3c509.h.....	sheets	130 to	132 ( 3)	pages	130-132	165 lines
28	8390.c.....	sheets	133 to	143 (11)	pages	133-143	738 lines
29	8390.h.....	sheets	144 to	146 ( 3)	pages	144-146	175 lines
30	Makefile.....	sheets	147 to	147 ( 1)	pages	147-147	60 lines
31	README.....	sheets	148 to	148 ( 1)	pages	148-148	39 lines
32	devio.c.....	sheets	149 to	150 ( 2)	pages	149-150	132 lines
33	dp.c.....	sheets	151 to	159 ( 9)	pages	151-159	662 lines
34	dp.h.....	sheets	160 to	163 ( 4)	pages	160-163	282 lines
35	ne.c.....	sheets	164 to	166 ( 3)	pages	164-166	194 lines
36	ne.h.....	sheets	167 to	167 ( 1)	pages	167-167	38 lines
37	netbuff.c.....	sheets	168 to	170 ( 3)	pages	168-170	169 lines
38	wd.c.....	sheets	171 to	175 ( 5)	pages	171-175	315 lines
39	wd.h.....	sheets	176 to	177 ( 2)	pages	176-177	106 lines
40	Makefile.....	sheets	178 to	178 ( 1)	pages	178-178	46 lines
41	floppy.c.....	sheets	179 to	196 (18)	pages	179-196	1297 lines
42	floppy.h.....	sheets	197 to	197 ( 1)	pages	197-197	7 lines
43	Makefile.....	sheets	198 to	198 ( 1)	pages	198-198	41 lines
44	fxp.c.....	sheets	199 to	232 (34)	pages	199-232	2504 lines
45	fxp.h.....	sheets	233 to	240 ( 8)	pages	233-240	577 lines
46	mii.c.....	sheets	241 to	243 ( 3)	pages	241-243	199 lines
47	mii.h.....	sheets	244 to	245 ( 2)	pages	244-245	117 lines
48	Makefile.....	sheets	246 to	246 ( 1)	pages	246-246	41 lines
49	lance.c.....	sheets	247 to	271 (25)	pages	247-271	1833 lines
50	lance.h.....	sheets	272 to	273 ( 2)	pages	272-273	104 lines
51	Makefile.....	sheets	274 to	274 ( 1)	pages	274-274	31 lines
52	driver.c.....	sheets	275 to	280 ( 6)	pages	275-280	377 lines
53	driver.h.....	sheets	281 to	282 ( 2)	pages	281-282	92 lines
54	drvlib.c.....	sheets	283 to	285 ( 3)	pages	283-285	200 lines
55	drvlib.h.....	sheets	286 to	286 ( 1)	pages	286-286	28 lines
56	Makefile.....	sheets	287 to	287 ( 1)	pages	287-287	46 lines
57	diag.c.....	sheets	288 to	289 ( 2)	pages	288-289	120 lines
58	kputc.c.....	sheets	290 to	290 ( 1)	pages	290-290	36 lines
59	log.c.....	sheets	291 to	297 ( 7)	pages	291-297	481 lines
60	log.h.....	sheets	298 to	298 ( 1)	pages	298-298	38 lines
61	Makefile.....	sheets	299 to	299 ( 1)	pages	299-299	62 lines
62	imgrd.c.....	sheets	300 to	300 ( 1)	pages	300-300	15 lines
63	imgrd_s.s.....	sheets	301 to	301 ( 1)	pages	301-301	16 lines
64	local.h.....	sheets	302 to	302 ( 1)	pages	302-302	7 lines
65	memory.c.....	sheets	303 to	308 ( 6)	pages	303-308	438 lines
66	Makefile.....	sheets	309 to	310 ( 2)	pages	309-310	102 lines
67	bintoc.c.....	sheets	311 to	312 ( 2)	pages	311-312	130 lines
68	mtab.....	sheets	313 to	313 ( 1)	pages	313-313	2 lines
69	proto.....	sheets	314 to	314 ( 1)	pages	314-314	25 lines
70	proto.sh.....	sheets	315 to	315 ( 1)	pages	315-315	11 lines
71	rc.....	sheets	316 to	316 ( 1)	pages	316-316	39 lines
72	allocmem.c.....	sheets	317 to	317 ( 1)	pages	317-317	15 lines
73	Makefile.....	sheets	318 to	318 ( 1)	pages	318-318	41 lines

74	<i>main.c</i> .....	sheets	319 to 325 ( 7)	pages	319-325	448 lines
75	<i>pci.c</i> .....	sheets	326 to 358 (33)	pages	326-358	2434 lines
76	<i>pci.h</i> .....	sheets	359 to 360 ( 2)	pages	359-360	91 lines
77	<i>pci_amd.h</i> .....	sheets	361 to 361 ( 1)	pages	361-361	22 lines
78	<i>pci_intel.h</i> .....	sheets	362 to 362 ( 1)	pages	362-362	59 lines
79	<i>pci_sis.h</i> .....	sheets	363 to 363 ( 1)	pages	363-363	18 lines
80	<i>pci_table.c</i> .....	sheets	364 to 368 ( 5)	pages	364-368	299 lines
81	<i>pci_via.h</i> .....	sheets	369 to 369 ( 1)	pages	369-369	28 lines
82	<i>Makefile</i> .....	sheets	370 to 370 ( 1)	pages	370-370	42 lines
83	<i>printer.c</i> .....	sheets	371 to 376 ( 6)	pages	371-376	422 lines
84	<i>Makefile</i> .....	sheets	377 to 377 ( 1)	pages	377-377	54 lines
85	<i>main.c</i> .....	sheets	378 to 381 ( 4)	pages	378-381	247 lines
86	<i>random.c</i> .....	sheets	382 to 385 ( 4)	pages	382-385	239 lines
87	<i>random.h</i> .....	sheets	386 to 386 ( 1)	pages	386-386	13 lines
88	<i>sha2.c</i> .....	sheets	387 to 401 (15)	pages	387-401	1087 lines
89	<i>sha2.h</i> .....	sheets	402 to 404 ( 3)	pages	402-404	166 lines
90	<i>boxes.dat</i> .....	sheets	405 to 427 (23)	pages	405-427	919 lines
91	<i>rijndael_alg.h</i> .....	sheets	428 to 428 ( 1)	pages	428-428	37 lines
92	<i>rijndael_api.h</i> .....	sheets	429 to 430 ( 2)	pages	429-430	79 lines
93	<i>rijndael.h</i> .....	sheets	431 to 432 ( 2)	pages	431-432	81 lines
94	<i>rijndael_alg.c</i> .....	sheets	433 to 439 ( 7)	pages	433-439	450 lines
95	<i>rijndael_api.c</i> .....	sheets	440 to 447 ( 8)	pages	440-447	587 lines
96	<i>word_i386.h</i> .....	sheets	448 to 448 ( 1)	pages	448-448	10 lines
97	<i>Makefile</i> .....	sheets	449 to 449 ( 1)	pages	449-449	49 lines
98	<i>rescue.c</i> .....	sheets	450 to 452 ( 3)	pages	450-452	223 lines
99	<i>Makefile</i> .....	sheets	453 to 453 ( 1)	pages	453-453	42 lines
100	<i>rtl8139.c</i> .....	sheets	454 to 490 (37)	pages	454-490	2676 lines
101	<i>rtl8139.h</i> .....	sheets	491 to 496 ( 6)	pages	491-496	430 lines
102	<i>Makefile</i> .....	sheets	497 to 497 ( 1)	pages	497-497	42 lines
103	<i>README</i> .....	sheets	498 to 498 ( 1)	pages	498-498	22 lines
104	<i>sb16.c</i> .....	sheets	499 to 499 ( 1)	pages	499-499	48 lines
105	<i>sb16.h</i> .....	sheets	500 to 502 ( 3)	pages	500-502	184 lines
106	<i>sb16_dsp.c</i> .....	sheets	503 to 512 (10)	pages	503-512	654 lines
107	<i>sb16_mixer.c</i> .....	sheets	513 to 518 ( 6)	pages	513-518	388 lines
108	<i>Makefile</i> .....	sheets	519 to 519 ( 1)	pages	519-519	41 lines
109	<i>i82365.h</i> .....	sheets	520 to 522 ( 3)	pages	520-522	168 lines
110	<i>ti1225.c</i> .....	sheets	523 to 529 ( 7)	pages	523-529	500 lines
111	<i>ti1225.h</i> .....	sheets	530 to 530 ( 1)	pages	530-530	72 lines
112	<i>Makefile</i> .....	sheets	531 to 531 ( 1)	pages	531-531	46 lines
113	<i>console.c</i> .....	sheets	532 to 549 (18)	pages	532-549	1274 lines
114	<i>keyboard.c</i> .....	sheets	550 to 566 (17)	pages	550-566	1218 lines
115	<i>pty.c</i> .....	sheets	567 to 575 ( 9)	pages	567-575	613 lines
116	<i>rs232.c</i> .....	sheets	576 to 589 (14)	pages	576-589	992 lines
117	<i>tty.c</i> .....	sheets	590 to 617 (28)	pages	590-617	2026 lines
118	<i>tty.h</i> .....	sheets	618 to 620 ( 3)	pages	618-620	196 lines
119	<i>vidcopy.s</i> .....	sheets	621 to 623 ( 3)	pages	621-623	161 lines
120	<i>Makefile</i> .....	sheets	624 to 625 ( 2)	pages	624-625	87 lines
121	<i>dvorak.src</i> .....	sheets	626 to 627 ( 2)	pages	626-627	140 lines
122	<i>french.src</i> .....	sheets	628 to 629 ( 2)	pages	628-629	136 lines
123	<i>genmap.c</i> .....	sheets	630 to 630 ( 1)	pages	630-630	59 lines
124	<i>german.src</i> .....	sheets	631 to 632 ( 2)	pages	631-632	136 lines
125	<i>italian.src</i> .....	sheets	633 to 634 ( 2)	pages	633-634	138 lines
126	<i>japanese.src</i> .....	sheets	635 to 637 ( 3)	pages	635-637	158 lines
127	<i>latin-america.src</i> ...	sheets	638 to 640 ( 3)	pages	638-640	154 lines
128	<i>olivetti.src</i> .....	sheets	641 to 642 ( 2)	pages	641-642	136 lines
129	<i>polish.src</i> .....	sheets	643 to 644 ( 2)	pages	643-644	141 lines
130	<i>scandinavian.src</i> ....	sheets	645 to 646 ( 2)	pages	645-646	139 lines
131	<i>spanish.src</i> .....	sheets	647 to 648 ( 2)	pages	647-648	139 lines
132	<i>uk.src</i> .....	sheets	649 to 650 ( 2)	pages	649-650	136 lines
133	<i>us-std.src</i> .....	sheets	651 to 652 ( 2)	pages	651-652	136 lines
134	<i>us-swap.src</i> .....	sheets	653 to 654 ( 2)	pages	653-654	136 lines