BOOT.C 5,31,85

```
/*
 * Commodore M-series Z8001 ROM bootstrap.
 * Boot a Coherent load module from an external
 * disc (or other) device:
 *      (xx,n,m)pathname
 * where:
 *      xx      is a disc name such as 'hd'
 *      n       unit number (default 0)
 *      m       offset in blocks into disc (default 0)
 *      pathname is an arbitrary filename on that file system.
 */
#define LADDR   1                       /* true for 32 bit l.out */

#include <ino.h>
#include <filsys.h>
#include <dir.h>
#include <canon.h>
#if     LADDR
#include "n.out.h"
#else
#include <l.out.h>
#endif
#include "rom.h"

#define bread(b,bn)     (*dread)(unit, (bn)+doffset, b)
#define min(a,b)        ((a)<(b)?(a):(b))

/* Error messages */
char    bootsyn[] = "boot command syntax error";
char    baddev[] = "bad boot device";
char    badlout[] = "l.out header error";
char    badtype[] = "bad file type";

static  unsigned lbn;                   /* Logical block number for vread */
static  long    doffset;                /* Disc block offset */
static  int     unit;                   /* Current logical unit # */
static  int     (*dread)();             /* Read routine */
char    *fsboot();
char    *fsload();
char    *dirlook();
struct dinode *iread();
static  char    buf[512];
long    vblocks[NBN+ND];

extern  unsigned        seg;
extern  int     wdlflag;                /* load from WD hard disk ? */

/*
 * Configuration of devices known to
 * the boot ROM.
 */
int     hdload();
int     hdread();
int     wdload();
int     wdread();
struct  devs    {
        char    dv_name[2];
        int     (*dv_load)();
        int     (*dv_read)();
        int     dv_uoff;
}       devs[] = {
        "hd", hdload, hdread, 0,
        "fd", hdload, hdread, 1,
```

```
};

#define NDEV    (sizeof devs/sizeof devs[0])

/*
 * Parse boot string and read the filesystem to find
 * the driver.
 */
char *
boot(s)
register char *s;
{
        static char dev[2];
        register int c;
        register struct devs *dp;

        if (*s++ != '(')
                return (bootsyn);
        if (*s=='\0' || *s==',')
                return (bootsyn);
        dev[0] = *s++;
        if (*s=='\0' || *s==',')
                return (bootsyn);
        dev[1] = *s++;
        for (dp = devs; dp < &devs[NDEV]; dp++)
                if (dp->dv_name[0]==dev[0] && dp->dv_name[1]==dev[1])
                        break;
        if (dp == &devs[NDEV])
                return (baddev);
        if (dev[0] == 'w' && dev[1] == 'd')
                wdlflag = 1;
        else
                wdlflag = 0;
        dread = dp->dv_read;
        /*
         * Parse unit number
         * and disc block offset.
         */
        doffset = unit = 0;
        if (*s == ',') {
                s++;
                while (*s>='0' && *s<='9')
                        unit = unit*10 + *s++-'0';
        }
        if (*s == ',') {
                s++;
                while (*s>='0' && *s<='9')
                        doffset = doffset*10 + *s++-'0';
        }
        unit += dp->dv_uoff;
        if (*s++ != ')')
                return (bootsyn);
        (*dp->dv_load)();
        return (fsboot(s));
}

/*
 * Boot a name from the file system.
 */
char *
fsboot(s)
register char *s;
{
```

```
        register char *cp;
        ino_t cino = ROOTIN;
        static char name[DIRSIZ];

        for (;;) {
                if (*s == '/')
                        s++;
                if ((dip = iread(cino)) == NULL)
                        return ("inode read error");
                switch (dip->di_mode & IFMT) {
                case IFDIR:
                        for (cp = name; cp < &name[DIRSIZ]; cp++)
                                *cp++ = 0;
                        for (cp = name; *s!='/' && *s!='\0'; s++)
                                if (cp < &name[DIRSIZ])
                                        *cp++ = *s;
                        if (cp == name)
                                return (badtype);
                        if ((cp = dirlook(dip, name, &cino)) != NULL)
                                return (cp);
                        break;

                case IFREG:
                        return (fsload(dip));

                default:
                        return (badtype);
                }
        }
}

/*
 * Look up the given DIRSIZ padded with nulls
 * name in the directory that vread currently
 * understands.
 */
char *
dirlook(ip, name, inop)
register struct dinode *ip;
register char *name;
ino_t *inop;
{
        register char *cp1, *cp2;
        register unsigned n;
        register struct direct *dp;

        while (ip->di_size > 0) {
                if (vread() == 0)
                        return ("directory read error");
                for (dp = &buf[0]; dp < &buf[512]; dp++) {
                        canino(dp->d_ino);
                        if (dp->d_ino == 0)
                                continue;
                        n = DIRSIZ;
                        cp1 = dp->d_name;
                        cp2 = name;
                        do {
                                if (*cp1++ != *cp2++)
                                        break;
                        } while (--n);
                        if (cp2 >= &name[DIRSIZ]) {
                                *inop = dp->d_ino;
                                return (NULL);
```

```
                }
                ip->di_size -= 512;
        }
        return ("file not found");
}

/*
 * Load the load module file into memory.
 * Set up segmentation in segments LDSEG and LDSEG+1.
 * Branch to its entry point to run it.
 */
char *
fsload(dip)
struct dinode *dip;
{
        unsigned s, o;
        unsigned bis, bds, is, ds;
        unsigned start;
        unsigned code, data;
        register struct ldheader *lp;
        register int i;

        if (vread() == 0)
                return (badlout);
        lp = buf;
#if     LADDR
        canshort(lp->l_magic);
        canshort(lp->l_flag);
        canshort(lp->l_machine);
        canshort(lp->l_tbase);
        canlong(lp->l_entry);
        for (i=0; i<NXSEG; i++)
                canlong(lp->l_ssize[i]);
#else
        canint(lp->l_magic);
        canint(lp->l_flag);
        canint(lp->l_machine);
        canvaddr(lp->l_entry);
        for (i=0; i<NXSEG; i++)
                cansize(lp->l_ssize[i]);
#endif
        if (lp->l_magic != L_MAGIC
#if     LADDR
         || (lp->l_flag & (LF_SEP|LF_32)) != (LF_SEP|LF_32)
         || lp->l_machine != M_Z8001)
#else
         || (lp->l_flag & LF_SEP) == 0
         || lp->l_machine != M_Z8002)
#endif
                return (badlout);
        start = lp->l_entry;
        code = romconf.rom_bram<<2;
        s = code & ~0xFF;
        if (wdlflag)
                s += 0x100;                     /* bump to next higher phys seg */
        o = 0;
        bds = lp->l_ssize[L_BSSD];
        bis = lp->l_ssize[L_BSSI];
        is = lp->l_ssize[L_SHRI] + lp->l_ssize[L_PRVI];
        ds = lp->l_ssize[L_SHRD] + lp->l_ssize[L_PRVD];
        /* BSSI + roundup to 512-byte click boundary */
        bis += 1024 - (bis+is)%1024;
#if     LADDR
```

```
#else
        i = fload(&s, &o, sizeof *lp, is, bis);
#endif
        fload(&s, &o, i, ds, bds);
        /*
         * this really ugly code handles the case where the command
         * block for the Western Digital controller is fixed at the
         * base of RAM (RAMBASE). Since this is where we want to load
         * Coherent, we have to kludge a bit by loading it one physical
         * segment higher in memory and then block moving it down into
         * place after all disk I/O is completed. Then we have to clear
         * the memory that we stepped on.
         */
        if (wdlflag) {
                s = romconf.rom_bram << 2;
                s &= ~0xFF;
                ppcopy(laddr((s+0x100), 0), laddr(s, 0), (unsigned)0xFFFFL);
                pclear(laddr((s+0x100), 0), (unsigned)0xFFFFL);
        }
        data = code;
        data += ((bis+is+1023)&~1023)>>8;
        segload(LDSEG, code, data);
        jump(laddr(LDSEG<<8, start), 1);
        /* NOTREACHED */
}

/*
 * Load data from a file.
 * The 'segp' and 'offp' are call by reference.
 * The offset is into the block already read.
 * And offset of 0, means do another read.
 * Return a new offset.
 */
fload(segp, offp, offset, size, bsize)
unsigned *segp, *offp;
unsigned offset;
unsigned size;
unsigned bsize;
{
        register char *bp;
        register unsigned s, o;
        register unsigned count;

        s = *segp;
        o = *offp;
        for ( ; size!=0; size -= count) {
                if (offset == 0)
                        vread();
                count = 512-offset;
                bp = &buf[offset];
                count = min(count, size);
                ppcopy(bp, laddr(s, o), count);
                if (o + count  < o)
                        s++;
                o += count;
                offset = 0;
        }
        offset = count;
        if (bsize != 0) {
                pclear(laddr(s, o), bsize);
                if (o + bsize < o)
                        s++;
                o += bsize;
```

```
        *segp = s;
        *offp = o;
        return (offset);
}

/*
 * Read in an inode.
 */
struct dinode *
iread(ino)
register ino_t ino;
{
        register struct dinode *dip;
        register int i;

        if (bread(buf, (long)iblockn(ino)) == 0)
                return (NULL);
        dip = ((struct dinode *)buf) + iblocko(ino);
        canshort(dip->di_mode);
        cansize(dip->di_size);
        l3tol(vblocks, dip->di_addr, ND+1);
        if (vblocks[ND] != 0) {
                if (bread((char *)&vblocks[ND], vblocks[ND]) == 0)
                        return (NULL);
                for (i=ND; i<ND+NBN; i++)
                        cansize(vblocks[i]);
        } else
                for (i=ND; i<ND+NBN; i++)
                        vblocks[i] = 0;
        lbn = 0;
        return (dip);
}

/*
 * Read virtual blocks sequentially from a file.
 * The block number is only an integer as it
 * can handle only 138 blocks of a file.
 * However, this handles sparse files.
 */
vread()
{
        long pb;

        if (lbn >= ND+NBN || (pb = vblocks[lbn])==0) {
                register char *cp;

                for (cp = buf; cp < &buf[512]; cp++)
                        *cp = 0;
                lbn++;
                return (1);
        } else {
                lbn++;
                return (bread(buf, pb));
        }
}
```

CONF.C
5.31.85

```
/*
 * Commodore M-series boot ROM.
 * Establish machine configuration parameters.
 */

#include "rom.h"

configure(rp, ramlow, ramhigh)
register struct romconf *rp;
unsigned ramlow, ramhigh;
{
        rp->rom_bram = ramlow;
        rp->rom_eram = ramhigh;
        rp->rom_ctype = CTYPE;
}
```

```
CFLAGS=-O
OPTIONS=-DZ8000HR -DFIVEINCH=1 -DDDT -DAUTOBOOT
CONFIG=-DRAMBASE=0x08000000 -DCMDBLKPADDR="(0x8000000L)"\
 -DBUFPADDR="(0x8000400L)"
CC=cc -c $(OPTIONS) $(CONFIG) -I/usr/sys/z8001/h
LD=/bin/nld -i -R 0x00000000 -o rom
CCP=cc -E $(OPTIONS) $(CONFIG)
LIB=/lib/libc.a
OBJECTS=crt.o rom.o ramtest.o sio.o hd.o wd.o boot.o diag.o trap.o conf.o\
 ddt.o hires.o io.o loadseg.o lceddt.o mbittest.o diagdisp.o porttest.o
rom: $(OBJECTS)
        $(LD) $(OBJECTS) $(LIB)
        bootfix rom
lceddt.o : lceddt.c
        cc -c -DK1 -DNLD -DDDT -DFIVEINCH -I/usr/sys/z8001/h -I /usr/sys/h\
 lceddt.c |tee lceddt.err
diagdisp.o : diagdisp.c cioports.h
        $(CC) diagdisp.c |tee diagdisp.err
porttest.o : porttest.c cioports.h
        $(CC) porttest.c |tee porttest.err
loadseg.o : loadseg.s
        /lib/cpp -E -DNLD -DDDT loadseg.s >/tmp/loadseg.i
        as -gox loadseg.o /tmp/loadseg.i |tee loadseg.err
        rm -f /tmp/loadseg.i
hires.o : hires.s
        /lib/cpp -E -DNLD -DDDT hires.s >/tmp/hires.i
        as -gox hires.o /tmp/hires.i |tee hires.err
        rm -f /tmp/hires.i
rom.o: romconf.h rom.c
        $(CC) rom.c |tee rom.err
trap.o: trap.c
        $(CC) trap.c |tee trap.err
hd.o :  hd.c
        $(CC) hd.c |tee hd.err
wd.o : wd.c
        $(CC) wd.c |tee wd.err
diag.o : diag.c
        $(CC) diag.c |tee diag.err
boot.o : romconf.h boot.c
        $(CC) boot.c |tee boot.err
sio.o : romconf.h sio.c
        $(CC) sio.c |tee sio.err
conf.o : romconf.h conf.c
        $(CC) conf.c |tee conf.err
bootfix : bootfix.c
        cc -o bootfix -I/usr/sys/z8001/h bootfix.c |tee bootfix.err
prom : prom.c
        cc -o prom -I/usr/sys/z8001/h prom.c |tee prom.err
crt.o : 1kcrt.s
        /lib/cpp -E  -DDDT -DRAMBASE=0x08000000 1kcrt.s > crt.i
        as -go crt.o crt.i |tee 1kcrt.err
        rm -f crt.i
ddt.o : ddt.s
        /lib/cpp -E  -DDDT -DRAMBASE=0x08000000 ddt.s >ddt.i
        as -go ddt.o ddt.i |tee ddt.err
        rm -f ddt.i
io.o : io.s
        as -go io.o io.s |tee io.err
ramtest.o : ramtest.s
        as -go ramtest.o ramtest.s |tee ramtest.err
mbittest.o : mbittest.s
        as -go mbittest.o mbittest.s |tee mbittest.err
```