```
/* AR.c for the archive 1/4" TAPE cartridge drive.
 *
 *  For use with the Archive smart controller and AMD95/3310
 *   parallel interface to the multibus.
 *
 *  Written for Forward Technology
 *  July 17, 1982
 *  Bruce Borden
 *
 *   I hope it's modified by chase b.
 *  July 21, 1982
 */
/* first pass 7/11/82 chase b. */
/* chase b. 7/11/82 */
/* AR.C for the archive tape drive */
/* BB started on it 7/17/82 */
/* 7/19/82 added ECTISELECT */
/* 7/19/82 removed arwt(READY) from the start of ar_command */
/* for power up reasons chase b. */
/* 7/19/82 changed aropen and arwrite to test for more inline code */
/* 7/20/82 rewrote  open write and read and added some for error */
/* need to add check at first for not a drive selected */
/* need to time out in arwt and arwtn */
/* 7/22/82 added timeout to almost all of the while routines with */
/* error notification to all calling routines, chase */
/* removed need for T_init ???? dont know if it helped, I will just
 * reset everytime.
 * made reg = 800000 in 2 instances for arrd and arwrt - had timed out
 * twice.
 * removed some of the timeouts and also remade calls to arwt and arwtn
```

```
 * found ERROR in our code with error types ** corrected 7/24/82
 * Bullet proofed on 7/27/82 and will release as slow standalone
 */


#include        "../h/param.h"
#include        "../h/buf.h"
#include        "../h/dir.h"
#include        "../h/conf.h"
#include        "../h/file.h"
#include        "../h/user.h"

struct  {
        unsigned char   arst0;  /* first byte of status */
        unsigned char   arst1;  /* second byte of status */
        short           arrt;   /* retry count */
        short           arur;   /* underrun count */
} arstat;

/* The following h/w addresses are for the amd 3310 i/f board
   and are byte twisted for the multibus */

#define ARPAD   0x8d    /* Data address to port A */
#define ARPBD   0x8c    /* Handshaking port from archive port B */
#define ARPCD   0x8f    /* Handshaking port to archive port C */
#define ARPDD   0x8e    /* Command port to the 8255 for port A out or in */

/* Commands for the ARCMD */

#define AINCMD  0x92    /* turns the 8255 port A for data in */
```

```
#define AOUTCMD 0x82     /* turns the 8255 port A for data out */
#define CHAIN   0        /* signals the latch input for data in */
#define CHAOUT  0x40     /* signals the latch input for data out */

/* Inputs from ARPBD */
#define ACK     1        /* Acknowledge signal from the Archive */
#define READY   2        /* Ready signal from the Archive */
#define EXCEPTION 4      /* Exception signal from the Archive */
#define DIRC    8        /* DIRC signal from the Archive */

/* Commands for the ARchive via ARPCD */
/* make sure that you and or or with CDATAIN or CDATAOUT */

#define TONLINE 1        /* Online signal to the ARChive */
#define TREQ    2        /* Request signal to the ARchive */
#define TRESET  4        /* Reset signal to the ARchive */
#define TXFER   8        /* Transfer signal to the ARchive */

/* Commands for the ARchive unit via ARADDR (port A) */

#define TREAD   0x80
#define TWRITE  0x40
#define TERASE  0x22
#define TRETEN  0x24
#define TRSTAT  0xc0
#define TSELECT 0x11

/* definition of ARST byte 0 */

#define ANOTIN  0x40     /* no tape inserted */
```

```
#define ANOTSLT 0x20     /* unit not selected */
#define AWRPROT 0x10     /* Write protected */
#define AENDTRK 0x08     /* End of Tape */ /* Track */
#define AUNERR  0x04     /* Unrecoverable Error */
#define AFLMRK  0x01     /* File mark encountered */

/* definition of ARST byte 1 */

#define AILLCMD 0x40     /* Illegal command */
#define ANODATA 0x20     /* No data found */
#define A8RETRY 0x10     /* 8 or more retries (BAD TAPE) */
#define ABEGIN  0x08     /* BOT */

char    ar_flags;        /* flags */
char    ar_buf[512];     /* Temp rd/wrt buffer for I/O */

/* Definition of FLAG BITS */
#define T_OPEN  1        /* Unit has been opened and reset */
#define T_INIT  2        /* Unit has been inited */

/* ar_command() commands */
#define CGETSTS 1        /* Read Archive status into arstat */
#define COFFLINE 2       /* Take unit offline */
/* #define      CREAD   3 */    /* Condition unit for reading */
/* #define      CWRITE  4 */    /* Condition unit for writing */
#define CERASE  5        /* Erase tape (tbd) */
#define CTENSION 6       /* Tension tape (tbd) */
#define CRESET  7        /* Reset the drive */
#define CSELECT 8        /* select the drive */
```

```
/* ARGSUSED */
aropen(dev, flag)
dev_t dev;
int flag;
{

        if(ar_flags & T_OPEN) {
                u.u_error = ENXIO;
                return;
        }
        /* removed reference to T_INIT */
        ar_command(CRESET);
        ar_command(CGETSTS);            /* Get the initial status of tape */
        ar_command(CSELECT);            /* select the drive */
        if(arstat.arst0 & ANOTIN) {
                printf("No tape in Archive drive\n");
                u.u_error = ENXIO;
                return;
        } else if(flag && (arstat.arst0 & AWRPROT)) {
                printf("Archive Tape is WRITE PROTECTED\n");
                u.u_error = ENXIO;
                return;
        }
        if(flag == 0)
                arrdset();
        else
                arwrset();
        if (u.u_error == 0)
                ar_flags = T_OPEN;              /* set flag */
```

```
}

/* Close routine */
/* On the Archive, dropping ONLINE will automatically write a FileMark
 * (if the tape is being written) and rewind the tape.
 */

/* ARGSUSED */  /* Tell LINT I know I'm ignoring my args */
arclose(dev, flag)
dev_t dev;
int flag;
{
        outb(ARPDD, AOUTCMD);   /* recondition the 8255 */
        ar_command(COFFLINE);   /* Take offline (may be NOP) */
        ar_flags = 0;           /* clear t_open flag */
}
```

```
#define WT20Us  for(wait = 50; --wait; )


ar_command(cmd)
{
        register reg;
        register wait;                          /* added for CSELECT */
/*      register struct buf *ar; */
        register st;

        switch(cmd) {
        case CRESET:                /* Must be followed by a CGETSTS!! */
                outb(ARPDD, AOUTCMD);    /* Condition 8255 for output */
                outb(ARPCD, CHAOUT);     /* Condition latch */
                outb(ARPCD, TRESET);     /* Do the reset */
                break;
        case CSELECT:
                arwt(READY|EXCEPTION, "CSELECT1");/* wait for ready */
                outb(ARPAD, TSELECT);    /* put the select command on bus */
                outb(ARPCD, TREQ|CHAOUT); /* put out request */
                WT20Us;                  /* wait to assure that he saw it */
                arwt(READY|EXCEPTION, "CSELECT2");/* wait for ready */
                outb(ARPCD, CHAOUT);     /* remove request */
                break;
        case CGETSTS:
                if(arstin())             /* Get 6 bytes of status in */
                        return;
                break;
        case COFFLINE:
                outb(ARPCD, CHAOUT);     /* Clear ONLINE (if it was) */
```

```
                break;
        case CERASE:
        case CTENSION:
                ;                               /* TBD */
        }
}


/* Copy 6 bytes of status in from Archive */
arstin()
{
        register reg;
        register char *cp = (char *)&arstat;
        register i, wait;
        register st;

        outb(ARPDD, AOUTCMD);    /* make sure the 8255 is for out */
        outb(ARPAD, TRSTAT);     /* Send rd status to Port AD */
        outb(ARPCD, TREQ|CHAOUT);
        if(arwt(READY, "arstin") == 0)/* Tells me AR has command */
                return 1;
        outb(ARPDD, AINCMD);     /* Condition 8255 for input */
        outb(ARPCD, CHAIN);      /* Condition latch for input (clrs REQUEST)*/
        if((st = arwt(DIRC|EXCEPTION, "arstin2")) == 0)
                return 1;
        if(st & EXCEPTION) {
                arsterr();
                return 1;
        }
        for(i = 6; i--; ) {
```

```
        if((st = arwt(READY|EXCEPTION, "arstin3")) == 0)
                return 1;
        if(st & EXCEPTION) {
                arsterr();
                return 1;
        }
        *cp++ = inb(ARPAD);       /* Fetch a status byte */
        outb(ARPCD, TREQ);        /* Set Request again */
/*      arwtn(READY); */          /* Wait for ready low */
        reg = 500000;
        while((st = inb(ARPBD)) & READY) {
                if(st & EXCEPTION) {
                        printf(" arstin EXCEPTION\n");
                        u.u_error = EIO;
                        return 1;
                }
                if(--reg == 0){
                        printf(" ar0 not ready(arstin4)\n");
                        u.u_error = EIO;
                        return 1;
                }
        }
        WT20Us;
        outb(ARPCD, 0);           /* Clear Request */
}
outb(ARPDD, AOUTCMD);    /* Recondition 8255 for output */
printf("ARST=%x,%x\n",arstat.arst0, arstat.arst1);
return 0;
}
```

```
arwt(bits, msg)
{
        register st, reg = 500000;

        while(((st = inb(ARPBD)) & bits) == 0)
                if(--reg == 0) {
                        printf("TIMEOUT arwt: %s\n", msg);
                        u.u_error = EIO;
                        return 0;
                }
        return st;
}
/*
arwtn(bit)
{
        register st;

        while((st = inb(ARPBD)) & bit);
        return st;
}
*/
```

```
/* ARGSUSED */
arread(dev)
dev_t dev;
{
        while(u.u_count) {              /* While the user wants more */
                if(arrd())              /* Read a block */
                        break;          /* EOF or an error */
                                        /* Copy the data to the user */
                iomove(ar_buf, min(sizeof ar_buf, u.u_count), B_READ);
        }
}

/* set up the archive for reading */
arrdset()
{
        register reg = 200;

        outb(ARPDD, AOUTCMD);           /* condition the 8255 */
        outb(ARPCD, TONLINE|CHAOUT);     /* Set ONLINE    */
        outb(ARPAD, TREAD);             /* Command to A bus */
        outb(ARPCD, TONLINE|CHAOUT|TREQ); /* Set ONLINE&REQUEST */
        while(inb(ARPBD) & READY)       /* Wait for not ready */
                if(--reg == 0) break;   /* But only for a while */
        arwt(READY|EXCEPTION, "arrdset1");/* Wait for ready */
        outb(ARPDD, AINCMD);
        outb(ARPCD, TONLINE|CHAIN);      /* Clr REQUEST */
        arwt(DIRC|EXCEPTION, "arrdset2");
}
```

```
/* Read one 512 byte block into ar_buf */

arrd()

{
        register reg;
        register char *cp = ar_buf;
        register st;
        register i;


        i = 512;                        /* Fixed by the hardware!! */
        if((st = arwt(READY|EXCEPTION, "arrd")) == 0)
                return 1;
        if(st & EXCEPTION) {
                arerr();
                return 1;
        }
        do {
                reg = 200;
                while(((inb(ARPBD)) & ACK) == 0)
                        if(--reg == 0)
                                return 1;
                *cp++ = inb(ARPAD);     /* Grab the byte */
                outb(ARPCD, TXFER|CHAINITONLINE); /* Set XFER to notify ctlr */
                reg = 200;
                while(inb(ARPBD) & ACK)
                        if(--reg == 0)
                                return 1;
                outb(ARPCD, CHAINITONLINE);     /* Clear XFER */
        } while(--i);                   /* And loop for 512 bytes */
        return 0;

}
```

```
/* Write routine */
/* ARGSUSED */
arwrite(dev)
dev_t dev;
{
        while(u.u_count) {              /* While the user has more to write */
                                        /* Copy the data from the user */

                iomove(ar_buf, min(sizeof ar_buf, u.u_count), B_WRITE);
                if(arwrt())             /* Write the block */
                        break;          /* EOF or an error */
        }
}

/* set up the archive for writing */
arwrset()
{
        register reg;

        outb(ARPDD, AOUTCMD);           /* condition the 8255 */
        outb(ARPCD, TONLINE|CHAOUT);     /* Set ONLINE    */
        outb(ARPAD, TWRITE);            /* Command to A bus */
        outb(ARPCD, TONLINE|CHAOUT|TREQ); /* Set ONLINE&REQUEST */
        reg = 200;
        while(inb(ARPBD) & READY)       /* Wait for not ready */
                if(--reg == 0) break;   /* ...only for a while */
        arwt(READY|EXCEPTION, "arwrset1");
        outb(ARPCD, TONLINE|CHAOUT);     /* Clr REQUEST */
        reg = 200;
```

```
        while(inb(ARPBD) & READY)       /* Wait for not ready */
                if(--reg == 0) break;   /* ...only for a while */
        arwt(READY|EXCEPTION, "arwrset2");/* Wait for ready */
}

/* Write one 512 byte block from ar_buf to the tape */
arwrt()
{
        register reg;
        register char *cp = ar_buf;
        register st;
        register i;

        i = 512;                        /* fix the number of bytes */
        if((st = arwt(READY|EXCEPTION, "arwrt")) == 0)
                return 1;
        if(st & EXCEPTION) {            /* Some error indicated */
                arerr();                /* Handle the error */
                return 1;
        }
        do {
                outb(ARPAD, *cp++);     /* Output one byte */
                outb(ARPCD, TXFER|CHAOUT|ONLINE); /* Set XFER to notify ctlr */
                reg = 200;
                while(((inb(ARPBD)) & ACK) == 0)
                        if(--reg == 0)
                                return 1;
                outb(ARPCD, CHAOUT|ONLINE);     /* Clear XFER */
                reg = 200;
                while((inb(ARPBD)) & ACK)
```

```
                if(--reg == 0)
                        return 1;
        } while(--i);                   /* And loop for 512 bytes */
        return 0;
}
```

```
arerr()
{
        arstin();                   /* Get the status */
        if(arstat.arst0 & AFLMRK) {     /* File mark?? */
                u.u_error = 0;  /* This is NOT an error */
                return;         /* Yes, normal eot */
        }
        if(arstat.arst0 & AENDTRK) {    /* End of tape?? */
                printf("AR EOT\n");
                u.u_error = EIO;
                return;
        }
        if(arstat.arst0 & AUNERR) {     /* Unrecoverable error?? */
                printf("AR ERR %x\n", arstat.arst1);
                u.u_error = EIO;
                return;
        }
        if(arstat.arst1 & AILLCMD) {
                printf("AR ILLEGAL COMMAND %x\n", arstat.arst1);
                u.u_error = EIO;
                return;
        }
        if(arstat.arst1 & ANODATA) {
                printf("AR NO DATA FOUND %x\n", arstat.arst1);
                u.u_error = EIO;
                return;
        }
        if(arstat.arst0 & ANOTIN) {
                printf("No tape in Archive drive\n");
```

```
                u.u_error = ENXIO;
                return;
        }
}

/* Routine to print error in arstin() */

arsterr()
{

        if(arstat.arst0 & AENDTRK) {     /* End of tape?? */
                printf("AR EOT\n");
                u.u_error = EIO;
                return;
        }
        if(arstat.arst0 & AUNERR) {      /* Unrecoverable error?? */
                printf("AR ERR %x\n", arstat.arst1);
                u.u_error = EIO;
                return;
        }
        if(arstat.arst1 & AILLCMD) {
                printf("AR ILLEGAL COMMAND %x\n", arstat.arst1);
                u.u_error = EIO;
                return;
        }
        if(arstat.arst1 & ANODATA) {
                printf("AR NO DATA FOUND %x\n", arstat.arst1);
                u.u_error = EIO;
                return;
        }
```

```
        if(arstat.arst0 & ANOTIN) {
                printf("No tape in Archive drive\n");
                u.u_error = ENXIO;
                return;
        }
}
```