```
/*
 * Interphase Corp. SMD 2180 Disk Controler XENIX Driver
 * Henry Burgess    March 15, 1982
 * Modified from WDC 2880 Controller by Jerry Dunietz, March 29, 1982
 *       Version for only 1 drive, with debugging code.
 *       Limited support for second drive, so that we can to disk-to
 *               disk backup
 */

/*
 * Modifications
 *       M000    10 May 1982     JJD
 *               Reconfiguration of multibus memory.
 *       M001    20 May 1982     JJD
 *               Modified so that ip can share interrupt line with other
 *               device.
 */

/*
 *       Modification History
 *
 *    XXX         00/00/00
 *       Comments .....
 */

#include "../h/param.h"
#include "../h/systm.h"
#include "../h/buf.h"
#include "../h/conf.h"
#include "../h/dir.h"
```

```
#include "../h/user.h"
#include "../h/dma.h"                                                  /*M000*/

#ifdef DEBUG
int      ip_debug = 0;
#endif

#define DK_N     0

#define NIP      1               /* # of drives supported      */
#define NIPSEC   34              /* # of sectors per track     */
#define NIPTRK   7               /* # of tracks per cylinder (heads) */
#define NBPC     NIPSEC*NIPTRK   /* blocks / cylinder */
#define NIPCYL   589             /* cylinders */
#define NIPCYLH  NIPCYL/2        /* cylinders/2 */


/*
 *       IPsizes - Sizes and locations of each file structure
 *
 */

#define LOGCNT2 3                /* log2 of cnt of logical devices per phys */
#define UNITCNT 8                /* count of logical devices per physical */

struct  size
{
        daddr_t nblocks;
        int     cyloff;
} ip_sizes[UNITCNT] =
{
/*0*/   NBPC*60,        0,              /* cyl 0- 59, root on FUJI       */
/*1*/   NBPC*30,        60,             /* cyl   60-89, swap on FUJI     */
/*2*/   NBPC*(NIPCYL-90), 90,           /* cyl  90-end, user + rest of disk*/
                                        /* BUT USE ONLY HALF OF IT       */
/*3*/   NBPC*60,        NIPCYLH+0,      /* second half - root            */
/*4*/   NBPC*30,        NIPCYLH+60,     /* second half - swap            */
/*5*/   NBPC*(NIPCYLH-90), NIPCYLH+90,  /* seconf half - user            */
/*6*/   NBPC,           NIPCYL-1,       /* last cylinder (for boot)      */
/*7*/   NBPC*NIPCYL,    0,              /* cyl 0-end, all of FUJI        */
};


struct  ip_iopb
{
        unsigned char
                /* NOTE  *** TWISTED BYTES *** */
        ip_stat,                /*  1 Status Code       */
        ip_comm,                /*  0 Disk Command      */
        ip_unit,                /*  3 UNIT/CYLHI SELETC */
        ip_error,               /*  2 Error code                */
        ip_sec,                 /*  5 STARTING SECTOR */
        ip_cyl,                 /*  4 CYLINDER SELECT */
        ip_xmb,                 /*  7 BUFFER MEMORY ADDRESS */
        ip_count,               /*  6 SECTOR COUNT (1-256) */
        ip_lsb,                 /*  9 BUFFER MEMORY ADDRESS */
        ip_msb,                 /*  8 BUFFER MEMORY ADDRESS */
        ip_cioaddr,             /*  B CONTROLER I/O ADDRESS */
        ip_head,                /*  A Head Address              */
        ip_niopx,               /*  D Next IOP Address EXTEND   */
        ip_burst,               /*  C BUS BURST LENGTH (1-256) */
        ip_niopl,                   /*  F Next IOP Address Low       */
        ip_nioph,                   /*  E Next IOP Address High      */
        ip_asegl,                   /* 11 LSB of ADDRESS SEGMENT */
        ip_asegm                    /* 10 MSB of ADDRESS SEGMENT */
} *ip_iopbl;


struct  buf     iptab;
struct  buf     ripbuf;


#define IOADDR  0xf0                /* controller address */
/* I/O REGISTER LOCATIONS */
/*#define IOCOMM        0xf0                /* write command register */
/*#define IOSTAT        0xf0                /* read status register */
/*#define IOXSB 0xf1                /* write IOPB extended memory (4 bits) */
/*#define IOMSB 0xf2                /* write IOPB hi memory address bits */
/*#define IOLSB 0xf3                /* write IOPB low memory address bits */


/* NOTE  *** TWISTED ***  ADDRESSES */
#define IOCOMM  0xf1                /* write command register */
#define IOSTAT  0xf1                /* read status register */
#define IOXSB   0xf0                /* write IOPB extended memory (4 bits) */
#define IOMSB   0xf3                /* write IOPB hi memory address bits */
#define IOLSB   0xf2                /* write IOPB low memory address bits */


#define XMB16BIT        0x20                /* 16bit bus bit on */


#define GO      0x01                /* set to begin a transfer */
#define CLRINT  0x02                /* clear interrupt */
#define DONE    0x02                /* done */
#define BUSY    0x01                /* controller busy */
```

```
                /* COMMAND BYTE (IOPB BYTE 0) */

#define READ    0x81                /* read command */

#define WRITE   0x82                /* write command */

#define SEEK    0x8A                /* seek command */

#define INITIALIZE   0x87                /* drive reset */

#define RESET   0x8F                /* drive reset */

#define RESTORE 0x89               /* restor (seek 0) */

#define WFMT    0x84                /* write format */

#define VERIFY  0x83                /* verify format */



                /* STATUS BYTE 1  (IOPB BYTE 1) */

#define STOK    0x80               /* Opperation Successful */

#define STBUSY  0x81               /* Opperation in progress */

#define STERR   0x82               /* Error on last command */


/*#define       b_cylin b_resid */


struct ip {

        int ipbn;                  /* starting block number for remainder of

                                   /* multi-cylinder transfer.  -1 indicates

                                   /* that we're not in the middle of such

                                   /* a transfer.

                                   /* */

        unsigned ipbc;             /* remaining byte count */

        unsigned long ipma;        /* physical address for remaining transfer */

} ip = {-1, 0, 0};
```

```
ipstrategy(bp)
register struct buf *bp;
{
	register struct buf *dp;
	register int unit;
	long sz;

	if(ip_iopb1 == NULL)	/* initialize, if necessary */
		ipinitl();
	if (bp->b_flags & B_PHYS)			/*M000*/
		mapalloc(bp);				/*M000*/
	unit = minor(bp->b_dev);
	sz = bp->b_bcount;
	sz = (sz+BMASK)>>BSHIFT;
	if (unit >= (NIP<<LOGCNT2) ||
	   bp->b_blkno+sz > ip_sizes[unit&(UNITCNT-1)].nblocks) {
		bp->b_flags |= B_ERROR;
		iodone(bp);
		return;
	}
	bp->av_forw = NULL;
	bp->b_cylin = (bp->b_blkno + ip_sizes[unit].cyloff)/NBPC;
	spl5();
	dp = & iptab;

	/* if (dp->b_actf == NULL)
	/*	dp->b_actf = bp;
	/* else
	/*	dp->b_actl->av_forw = bp;
	/* dp->b_actl = bp;
	*/

	disksort(dp, bp);
	if (dp->b_active == NULL)
		ipstart();
	spl0();
}

ipstart()
{
	register struct buf *bp;
	register int unit;
	register struct ip_iopb *iopbp;
	register unsigned sn;
	register unsigned addr;
	register unsigned count;
	int cn,tn,dn;
	daddr_t bn;

	if ((bp = iptab.b_actf) == NULL)
		return;
	iptab.b_active++;
	unit = minor(bp->b_dev);
	dn = unit>>LOGCNT2;
	if (ip.ipbn < 0) {
		/*
		 * No remaining bytes from previous multi-cylinder transfer.
		 */
		bn = bp->b_blkno;
		addr = (bp->b_flags & B_MAP) ?			/*M000*/
```

```
                (unsigned) bp->b_un.b_addr :                          /*M000*/
                DMA_ADDR(bp->b_un.b_addr);                            /*M000*/
            count = bp->b_bcount;
        } else {
            /*
             * Handle rest of multi-cylinder transfer.
             */
            bn = ip.ipbn;
            addr = ip.ipma;
            count = ip.ipbc;
#ifdef DEBUG
            if (ip_debug)
                    printf("*** continuing multi-cylinder transfer bn=%d addr=%
#endif
        }
        cn = bn/(NIPTRK*NIPSEC) + ip_sizes[unit&(UNITCNT-1)].cyloff;
        sn = bn%(NIPTRK*NIPSEC);
        tn = sn/NIPSEC;
        sn = sn%NIPSEC;


        iopbp = ip_iopb1;
        /* setup the iopb */

        iopbp->ip_stat          = 0;
        iopbp->ip_error         = 0;
        iopbp->ip_unit          = (1 << (dn + 4)) | ((cn>>8)&0x0f);
        iopbp->ip_cyl           = cn & 0x00ff;
        iopbp->ip_sec           = sn;
```

```
	/*
	 * Use sn as a temporary to hold the number of sectors to be
	 * transferred.  (Note that we need a temporary because the
	 * number of sectors may be greater than 255, the largest possible
	 * unsigned char.)
	 *
	 * CODE DEPENDS ON FACT THAT NUMBER OF BLOCKS PER CYLINDER
	 * IS LESS THAN LARGEST UNSIGNED CHAR.
	 *
	 * The first part of the conditional is there to speed
	 * up the code for the most common case -- transfer of 1 block.
	 */
	if ((sn = count/512) <= 1
	    || bn/(NIPSEC*NIPTRK) == (bn+sn-1)/(NIPSEC*NIPTRK)) {
		ip.ipbn = -1;	/* doesn't cross cylinder boundary */
		iopbp->ip_count = sn;
	} else {
#ifdef DEBUG
		if (ip_debug)
			printf("*** multi-cylinder transfer bn=%d addr=%X count=%d\
		if (ip_debug && sn > 255)
			printf("***	transfer %d > 255 blocks\n", sn);
#endif
		/*
		 * Transfer crosses cylinder boundary.
		 * Correct so that it doesn't cross a boundary.
		 * Save info so that we may resume transfer.
		 */
		iopbp->ip_count=sn= (NIPSEC * NIPTRK)-(bn % (NIPSEC * NIPTRK));
		ip.ipbn = bn + sn;		/* where next piece starts */
```

```
        /* use unit as a temporary */
        unit = 512 * iopbp->ip_count;   /* MAGIC NUMBER */
        ip.ipbc = count - unit;         /* remaining byte count */
        ip.ipma = addr + unit;          /* address of remaining xfer */
}

iopbp->ip_xmb           = ((addr>>16)&0xf) | XMB16BIT;
iopbp->ip_lsb           = addr & 0xff;
iopbp->ip_msb           = addr>>8;
iopbp->ip_head          = tn;
iopbp->ip_cioaddr       = IOADDR;
iopbp->ip_burst         = 8;            /* burst size */
iopbp->ip_niopx         = XMB16BIT;
iopbp->ip_nioph         = 0;
iopbp->ip_niopl         = 0;
iopbp->ip_asegm         = 0;
iopbp->ip_asegl         = 0;

if (bp->b_flags & B_READ)
        iopbp->ip_comm  = READ;
else
        iopbp->ip_comm  = WRITE;

iopbp = (struct ip_iopb *) DMA_ADDR(iopbp);             /*M000*/
/* start off the device */
outb(IOXSB      , ((unsigned int)(iopbp)>>16) | XMB16BIT);
outb(IOMSB      , ((unsigned int)iopbp)>>8);
outb(IOLSB      , ((unsigned int)iopbp)&0xff);
outb(IOSTAT     , GO);
```

```
	dk_busy |= 1<<DK_N;
	dk_numb[DK_N] += 1;
	unit = bp->b_bcount>>6;
	dk_wds[DK_N] += unit;
}


ipintr()
{
	register struct buf *bp;
	register struct ip_iopb *iopbp;
	register int csr;

	/* After getting interrupt make sure you have a done bit */
	csr = inb(IOSTAT);

	if(ip_iopb1 == NULL) {  /* initalize the drive when you get the */
		outb(IOCOMM, CLRINT);                           /*M001*/
		ipinit1();      /* power up=interrupt */
		csr = inb(IOSTAT);                              /*M001*/
	}

	/* Begin M001 ..... */
	if ((csr & DONE) == 0) {
#ifdef DEBUG
		if (ip_debug) {
			printf("ipintr: DONE not set, csr=%x\n", csr);
			if (ip_debug > 1)
				debug(0);
		}
#endif
```

```
		return;
	}
	/* .....End M001 */

	outb(IOCOMM,CLRINT);	/* clear interrupt */

	if (iptab.b_active == NULL)
		return;

	csr = inb(IOSTAT);
	iopbp = ip_iopb1;

	if (iopbp->ip_stat == STBUSY) {
#ifdef DEBUG
		printf("ipintr: STBUSY, csr=%x\n", csr);
		if (ip_debug)
			debug(0);
#endif
		return;
	}

	dk_busy &= ~(1<<DK_N);

	bp = iptab.b_actf;
	iptab.b_active = NULL;

	if (iopbp->ip_stat != STOK) {
		deverror(bp, iopbp->ip_comm,
			(iopbp->ip_stat << 8) | iopbp->ip_error);
		printf("csr = %x,%x\n",csr,inb(IOSTAT));
```

```
#ifdef DEBUG
                printf("ipintr: unrecoverable error\n");
                if (ip_debug)
                        debug(0);
#endif
                bp->b_flags |= B_ERROR;
                ip.ipbn = -1;   /* don't try to transfer more */
        }
        if (iopbp->ip_error)    /* if required retries */
                deverror(bp, iopbp->ip_comm,
                        (iopbp->ip_stat << 8) | iopbp->ip_error);
        if (ip.ipbn >= 0) {
                ipstart();
                return;
        }
        iptab.b_errcnt = 0;
        iptab.b_actf = bp->av_forw;
        bp->b_resid = 0;
        iodone(bp);
        ipstart();
}

ipread(dev)
{
        physio(ipstrategy, &ripbuf, dev, B_READ);
}

ipwrite(dev)
{
        physio(ipstrategy, &ripbuf, dev, B_WRITE);
}


ipinitl()               /* call me once to initalize the controller */
{
        register struct ip_iopb *iopbp;
        register unit;
        register x;
        register i;
        register unsigned int addr;                     /*M000*/
        char *multimem();

        x = spl7();
        if ((ip_iopb1 = (struct ip_iopb *) multimem(sizeof(*ip_iopb1))) == NULL)
                panic("ip_iopb1");

        iopbp = ip_iopb1;
        addr = DMA_ADDR(iopbp);                         /*M000*/

#ifdef DEBUG
        printf("*** initializing disk: ");
#endif
        outb(IOCOMM,0);

        /* RESET, RESTOR */

        /* initalize each drive */
        for(unit=0;unit<NIP;unit++) {
                /* build iopb */
                iopbp->ip_stat          = 0;
                iopbp->ip_error           = 0;
                iopbp->ip_unit            = 1 << (unit + 4);
                iopbp->ip_cyl             = 0;
                iopbp->ip_sec             = NIPSEC;
                iopbp->ip_count           = NIPSEC;
                iopbp->ip_xmb             = XMB16BIT;
                iopbp->ip_lsb             = 0;
                iopbp->ip_msb             = 0;
                iopbp->ip_head            = 0;
                iopbp->ip_cioaddr         = IOADDR;
                iopbp->ip_burst           = 8;            /* burst size */
                iopbp->ip_niopx           = XMB16BIT;
                iopbp->ip_nioph           = 0;
                iopbp->ip_niopl           = 0;
                iopbp->ip_asegm           = 0;
                iopbp->ip_asegl           = 0;

                iopbp->ip_comm  = RESET;

                outb(IOXSB      , (addr >> 16) | XMB16BIT);
                outb(IOMSB      , addr >> 8);
                outb(IOLSB      , addr & 0xff);

                outb(IOCOMM,GO);                          /* RUN THE IOPB */

                while ((i = iopbp->ip_stat) != STOK && i != STERR)
                        ;
                if (i != STOK) {
                        printf("ip INITIALIZE error %x\n",iopbp->ip_error);
#ifdef DEBUG
```

```
                                debug(0);
#endif
                        }

                        /* DO RESTOR */

                        iopbp->ip_comm  = RESTORE;
                        iopbp->ip_stat          = 0;

                        outb(IOXSB      , (addr >> 16) | XMB16BIT);     /*M000*/
                        outb(IOMSB      , addr >> 8);                   /*M000*/
                        outb(IOLSB      , addr & 0xff);                 /*M000*/
                        outb(IOCOMM,GO);                        /* RUN THE IOPB */

                        while ((i = iopbp->ip_stat) != STOK && i != STERR)
                                ;
                        if (i != STOK) {
                                printf("ip RESTOR error %x\n",iopbp->ip_error);
#ifdef DEBUG
                                debug(0);
#endif
                        }
                }
                splx(x);
#ifdef DEBUG
                printf("initialization complete\n");
#endif
}

ipformat()      /* Format Unit 0 */
```

```
{
        int track,cylinder;
        register struct ip_iopb *iopbp;
        register int unit;
        register int x;
        register int i;
        register unsigned int addr;                         /*M000*/

        if(ip_iopb1 == NULL)    /* initalize the drive if necessary */
                ipinit1();
        x = spl7();                     /* so no interrupts from ip */
        iopbp = ip_iopb1;
        addr = DMA_ADDR(iopbp);                             /*M000*/
        unit = 0;


        iopbp->ip_stat          = 0;
        iopbp->ip_error         = 0;
        iopbp->ip_xmb           = XMB16BIT;
        iopbp->ip_lsb           = 0;
        iopbp->ip_msb           = 0;
        iopbp->ip_cioaddr       = IOADDR;
        iopbp->ip_burst         = 8;
        iopbp->ip_niopx         = XMB16BIT;
        iopbp->ip_nioph         = 0;
        iopbp->ip_niopl         = 0;
        iopbp->ip_asegm         = 0;
        iopbp->ip_asegl         = 0;
#ifdef DEBUG
        printf("*** formatting disk");
#endif
```

```
        for(cylinder=0;cylinder < NIPCYL;cylinder++) {
#ifdef DEBUG
        printf(" %d", cylinder);
#endif
        for(track=0; track< NIPTRK; ) {
                iopbp->ip_sec           = 0;
                iopbp->ip_count         = NIPSEC;
                iopbp->ip_unit          = (1<<(unit+4)) | ((cylinder>>8)&0x0f);
                iopbp->ip_cyl           = cylinder & 0x00ff;
                iopbp->ip_head          = track;
                iopbp->ip_stat          = 0;
                iopbp->ip_comm          = WFMT;

                outb(IOXSB      , (addr >> 16) | XMB16BIT);     /*M000*/
                outb(IOMSB      , addr >> 8);                   /*M000*/
                outb(IOLSB      , addr & 0xff);                 /*M000*/
                outb(IOCOMM,GO);                        /* RUN THE IOPB */

                while ((i = iopbp->ip_stat) != STOK && i != STERR)
                        ;
                if(i != STOK) {
                        printf("FORMAT ERROR CYL=%x TRK=%x\n",cylinder,track);
                        printf("\tiopbp=%X read stat=%x\n", iopbp, i & 0xff);
                        printf("\tstat=%x error=%x\n", iopbp->ip_stat,
                                iopbp->ip_error);
#ifdef DEBUG
                        printf("        &track = %X, &cylinder=%X\n",
                                &track, &cylinder);
                        debug(0);
#endif
```

```
                        continue;
                }
                iopbp->ip_sec           = 0;
                iopbp->ip_count         = NIPSEC;
                iopbp->ip_unit          = (1<<(unit+4)) | ((cylinder>>8)&0x0f);
                iopbp->ip_cyl           = cylinder & 0x00ff;
                iopbp->ip_head          = track;
                iopbp->ip_stat          = 0;
                iopbp->ip_comm          = VERIFY;

                outb(IOXSB      , (addr >> 16) | XMB16BIT);     /*M000*/
                outb(IOMSB      , addr >> 8);                   /*M000*/
                outb(IOLSB      , addr & 0xff);                 /*M000*/
                outb(IOCOMM,GO);                        /* RUN THE IOPB */

                while ((i = iopbp->ip_stat) != STOK && i != STERR)
                        ;
                if(i != STOK) {
                        continue;
                }
                track++;
        }
        }
#ifdef DEBUG
        printf("\n***   End of ipformat\n");
#endif
#ifdef DEBUG
        printf("\n*** verifying disk");
#endif
```

```
        for(cylinder=0;cylinder < NIPCYL;cylinder++) {
#ifdef DEBUG
        printf(" %d",cylinder);
#endif
        for(track=0;track< NIPTRK;track++) {
                iopbp->ip_sec           = 0;
                iopbp->ip_count         = NIPSEC;
                iopbp->ip_unit          = (1<<(unit+4)) | ((cylinder>>8)&0x0f);
                iopbp->ip_cyl           = cylinder & 0x00ff;
                iopbp->ip_head          = track;
                iopbp->ip_stat          = 0;
                iopbp->ip_comm          = VERIFY;

                outb(IOXSB      , (addr >> 16) | XMB16BIT);     /*M000*/
                outb(IOMSB      , addr >> 8);                   /*M000*/
                outb(IOLSB      , addr & 0xff);                 /*M000*/
                outb(IOCOMM,GO);                        /* RUN THE IOPB */

                while ((i = iopbp->ip_stat) != STOK && i != STERR)
                        ;
                if(i != STOK) {
                        printf("VERIFY ERROR CYL=%x TRK=%x\n",cylinder,track);
                        printf("\tiopbp=%X read stat=%x\n", iopbp, i & 0xff);
                        printf("\tstat=%x error=%x\n", iopbp->ip_stat,
                                iopbp->ip_error);
 #ifdef DEBUG
                        debug(0);
 #endif
                }
         }
        }
#ifdef DEBUG
        printf("\n***   End of ipverify\n");
#endif
        splx(x);
}
```