```
/*  $Id$
 *  1993/03/31
 *  linux/kernel/aha1740.c
 *
 *  Based loosely on aha1542.c which is
 *  Copyright (C) 1992  Tommy Thorn and
 *  Modified by Eric Youngdale
 *
 *  This file is aha1740.c, written and
 *  Copyright (C) 1992,1993  Brad McLean
 *
 *  Modifications to makecode and queuecommand
 *  for proper handling of multiple devices courteously
 *  provided by Michael Weller, March, 1993
 *
 * aha1740_makecode may still need even more work
 * if it doesn't work for your devices, take a look.
 */

#ifdef MODULE
#include <linux/module.h>
#endif

#include <linux/kernel.h>
#include <linux/head.h>
#include <linux/types.h>
#include <linux/string.h>
#include <linux/ioport.h>
#include <linux/proc_fs.h>
#include <linux/sched.h>
#include <asm/dma.h>

#include <asm/system.h>
#include <asm/io.h>
#include <linux/blk.h>
#include "scsi.h"
#include "hosts.h"
#include "sd.h"

#include "aha1740.h"
#include<linux/stat.h>

struct proc_dir_entry proc_scsi_aha1740 = {
    PROC_SCSI_AHA1740, 7, "aha1740",
    S_IFDIR | S_IRUGO | S_IXUGO, 2
};

/* IF YOU ARE HAVING PROBLEMS WITH THIS DRIVER, AND WANT TO WATCH
   IT WORK, THEN:
#define DEBUG
*/
#ifdef DEBUG
#define DEB(x) x
#else
#define DEB(x)
#endif

/*
 * $Header: aha1740.c,v 1.1 1992/07/24 06:27:38 root Exp root $
 */
```

```
static unsigned int slot, base;
static unsigned char irq_level;

static struct ecb ecb[AHA1740_ECBS];    /* One for each queued operation */

static int aha1740_last_ecb_used  = 0;  /* optimization */

int aha1740_makecode(unchar *sense, unchar *status)
{
    struct statusword
    {
        ushort  don:1,  /* Command Done - No Error */
                du:1,   /* Data underrun */
        :1,     qf:1,   /* Queue full */
                sc:1,   /* Specification Check */
                dor:1,  /* Data overrun */
                ch:1,   /* Chaining Halted */
                intr:1, /* Interrupt issued */
                asa:1,  /* Additional Status Available */
                sns:1,  /* Sense information Stored */
        :1,     ini:1,  /* Initialization Required */
                me:1,   /* Major error or exception */
        :1,     eca:1,  /* Extended Contingent alliance */
        :1;
    } status_word;
    int retval = DID_OK;

    status_word = * (struct statusword *) status;
#ifdef DEBUG
    printk("makecode from %x,%x,%x,%x %x,%x,%x,%x",
           status[0],status[1],status[2],status[3],
           sense[0],sense[1],sense[2],sense[3]);
#endif
    if (!status_word.don) /* Anything abnormal was detected */
    {
        /*A dditional info available? */
        if ( (status[1]&0x18) || status_word.sc )
        {
            /* Use the supplied info for further diagnostics */
            switch ( status[2] )
            {
            case 0x12:
                if ( status_word.dor )
                    retval=DID_ERROR;   /* It’s an Overrun */
                /* If not overrun, assume underrun and ignore it! */
            case 0x00: /* No info, assume no error, should not occur */
                break;
            case 0x11:
            case 0x21:
                retval=DID_TIME_OUT;
                break;
            case 0x0a:
                retval=DID_BAD_TARGET;
                break;
            case 0x04:
            case 0x05:
                retval=DID_ABORT; /* Either by this driver or the AHA1740
                                         itself */
                break;
```

```
            default:
                retval=DID_ERROR; /* No further diagnostics possible */
            }
        }
        else
        { /* Michael suggests, and Brad concurs: */
            if ( status_word.qf )
            {
                retval = DID_TIME_OUT; /* forces a redo */
                /* I think this specific one should not happen -Brad */
                printk("aha1740.c: WARNING: AHA1740 queue overflow!\n");
            }
            else if ( status[0]&0x60 )
            {
                retval = DID_ERROR; /* Didn’t find a better error */
            }
            /* In any other case return DID_OK so for example
               CONDITION_CHECKS make it through to the appropriate
               device driver */
        }
    }
    /* Under all circumstances supply the target status -Michael */
    return status[3] | retval << 16;
}

int aha1740_test_port(void)
{
    char    name[4],tmp;

    /* Okay, look for the EISA ID’s */
    name[0]= ’A’ -1 + ((tmp = inb(HID0)) >> 2); /* First character */
    name[1]= ’A’ -1 + ((tmp & 3) << 3);
    name[1]+= ((tmp = inb(HID1)) >> 5)&0x7;     /* Second Character */
    name[2]= ’A’ -1 + (tmp & 0x1f);             /* Third Character */
    name[3]=0;
    tmp = inb(HID2);
    if ( strcmp ( name, HID_MFG ) || inb(HID2) != HID_PRD )
        return 0;   /* Not an Adaptec 174x */

/*  if ( inb(HID3) != HID_REV )
        printk("aha1740: Warning; board revision of %d; expected %d\n",
            inb(HID3),HID_REV); */

    if ( inb(EBCNTRL) != EBCNTRL_VALUE )
    {
        printk("aha1740: Board detected, but EBCNTRL = %x, "
               "so disabled it.\n", inb(EBCNTRL));
        return 0;
    }

    if ( inb(PORTADR) & PORTADDR_ENH )
        return 1;   /* Okay, we’re all set */

    printk("aha1740: Board detected, but not in enhanced mode, "
           "so disabled it.\n");
    return 0;
}

/* A "high" level interrupt handler */
void aha1740_intr_handle(int irq, void *dev_id, struct pt_regs * regs)
```

```
{
    void (*my_done)(Scsi_Cmnd *);
    int errstatus, adapstat;
    int number_serviced;
    struct ecb *ecbptr;
    Scsi_Cmnd *SCtmp;

    number_serviced = 0;

    while(inb(G2STAT) & G2STAT_INTPEND)
    {
        DEB(printk("aha1740_intr top of loop.\n"));
        adapstat = inb(G2INTST);
        ecbptr = (struct ecb *) bus_to_virt(inl(MBOXIN0));
        outb(G2CNTRL_IRST,G2CNTRL); /* interrupt reset */

        switch ( adapstat & G2INTST_MASK )
        {
        case    G2INTST_CCBRETRY:
        case    G2INTST_CCBERROR:
        case    G2INTST_CCBGOOD:
            /* Host Ready -> Mailbox in complete */
            outb(G2CNTRL_HRDY,G2CNTRL);
            if (!ecbptr)
            {
                printk("Aha1740 null ecbptr in interrupt (%x,%x,%x,%d)\n",
                       inb(G2STAT), adapstat, inb(G2INTST),
                       number_serviced++);
                continue;
            }
            SCtmp = ecbptr->SCpnt;
            if (!SCtmp)
            {
                printk("Aha1740 null SCtmp in interrupt (%x,%x,%x,%d)\n",
                       inb(G2STAT), adapstat, inb(G2INTST),
                       number_serviced++);
                continue;
            }
            if (SCtmp->host_scribble)
                scsi_free(SCtmp->host_scribble, 512);
          /*
           * Fetch the sense data, and tuck it away, in the required
           * slot.  The Adaptec automatically fetches it, and there is
           * no guarantee that we will still have it in the cdb when
           * we come back.
           */
            if ( (adapstat & G2INTST_MASK) == G2INTST_CCBERROR )
              {
                memcpy(SCtmp->sense_buffer, ecbptr->sense,
                       sizeof(SCtmp->sense_buffer));
                errstatus = aha1740_makecode(ecbptr->sense,ecbptr->status);
              }
            else
                errstatus = 0;
            DEB(if (errstatus) \
                printk("aha1740_intr_handle: returning %6x\n", errstatus));
            SCtmp->result = errstatus;
            my_done = ecbptr->done;
            memset(ecbptr,0,sizeof(struct ecb));
            if ( my_done )
```

```
                my_done(SCtmp);
            break;
        case    G2INTST_HARDFAIL:
            printk("aha1740 hardware failure!\n");
            panic("aha1740.c"); /* Goodbye */
        case    G2INTST_ASNEVENT:
            /* Say What? */
            printk("aha1740 asynchronous event: "
                   "%02x %02x %02x %02x %02x\n", adapstat,
                   inb(MBOXIN0),inb(MBOXIN1),inb(MBOXIN2),inb(MBOXIN3));
            /* Host Ready -> Mailbox in complete */
            outb(G2CNTRL_HRDY,G2CNTRL);
            break;
        case    G2INTST_CMDGOOD:
            /* set immediate command success flag here: */
            break;
        case    G2INTST_CMDERROR:
            /* Set immediate command failure flag here: */
            break;
        }
      number_serviced++;
    }
}

int aha1740_queuecommand(Scsi_Cmnd * SCpnt, void (*done)(Scsi_Cmnd *))
{
    unchar direction;
    unchar *cmd = (unchar *) SCpnt->cmnd;
    unchar target = SCpnt->target;
    unsigned long flags;
    void *buff = SCpnt->request_buffer;
    int bufflen = SCpnt->request_bufflen;
    int ecbno;
    DEB(int i);


    if(*cmd == REQUEST_SENSE)
    {
        if (bufflen != sizeof(SCpnt->sense_buffer))
        {
            printk("Wrong buffer length supplied for "
                   "request sense (%d)\n", bufflen);
        }
        SCpnt->result = 0;
        done(SCpnt);
        return 0;
    }

#ifdef DEBUG
    if (*cmd == READ_10 || *cmd == WRITE_10)
        i = xscsi2int(cmd+2);
    else if (*cmd == READ_6 || *cmd == WRITE_6)
        i = scsi2int(cmd+2);
    else
        i = -1;
    printk("aha1740_queuecommand: dev %d cmd %02x pos %d len %d ",
           target, *cmd, i, bufflen);
    printk("scsi cmd:");
    for (i = 0; i < SCpnt->cmd_len; i++) printk("%02x ", cmd[i]);
    printk("\n");
```

```
#endif

    /* locate an available ecb */

    save_flags(flags);
    cli();
    ecbno = aha1740_last_ecb_used + 1;          /* An optimization */
    if (ecbno >= AHA1740_ECBS) ecbno = 0;

    do{
      if( ! ecb[ecbno].cmdw )
        break;
      ecbno++;
      if (ecbno >= AHA1740_ECBS ) ecbno = 0;
    } while (ecbno != aha1740_last_ecb_used);

    if( ecb[ecbno].cmdw )
      panic("Unable to find empty ecb for aha1740.\n");

    /* SCSI Initiator Command doubles as reserved flag */
    ecb[ecbno].cmdw = AHA1740CMD_INIT;

    aha1740_last_ecb_used = ecbno;
    restore_flags(flags);

#ifdef DEBUG
    printk("Sending command (%d %x)...",ecbno, done);
#endif

    /* SCSI Command Descriptor Block Length */
    ecb[ecbno].cdblen = SCpnt->cmd_len;

    direction = 0;
    if (*cmd == READ_10 || *cmd == READ_6)
        direction = 1;
    else if (*cmd == WRITE_10 || *cmd == WRITE_6)
        direction = 0;

    memcpy(ecb[ecbno].cdb, cmd, ecb[ecbno].cdblen);

    if (SCpnt->use_sg)
    {
        struct scatterlist * sgpnt;
        struct aha1740_chain * cptr;
        int i;
#ifdef DEBUG
        unsigned char * ptr;
#endif
        /* SCSI Initiator Command  w/scatter-gather */
        ecb[ecbno].sg = 1;
        SCpnt->host_scribble = (unsigned char *) scsi_malloc(512);
        sgpnt = (struct scatterlist *) SCpnt->request_buffer;
        cptr = (struct aha1740_chain *) SCpnt->host_scribble;
        if (cptr == NULL)
                panic("aha1740.c: unable to allocate DMA memory\n");
        for(i=0; i<SCpnt->use_sg; i++)
        {
            cptr[i].dataptr = (long) sgpnt[i].address;
            cptr[i].datalen = sgpnt[i].length;
        }
```

```
        ecb[ecbno].datalen = SCpnt->use_sg * sizeof(struct aha1740_chain);
        ecb[ecbno].dataptr = (long) cptr;
#ifdef DEBUG
        printk("cptr %x: ",cptr);
        ptr = (unsigned char *) cptr;
        for(i=0;i<24;i++) printk("%02x ", ptr[i]);
#endif
    }
    else
    {
        SCpnt->host_scribble = NULL;
        ecb[ecbno].datalen = bufflen;
        ecb[ecbno].dataptr = (long) buff;
    }
    ecb[ecbno].lun = SCpnt->lun;
    ecb[ecbno].ses = 1; /* Suppress underrun errors */
    ecb[ecbno].dir= direction;
    ecb[ecbno].ars=1;  /* Yes, get the sense on an error */
    ecb[ecbno].senselen = 12;
    ecb[ecbno].senseptr = (long) ecb[ecbno].sense;
    ecb[ecbno].statusptr = (long) ecb[ecbno].status;
    ecb[ecbno].done = done;
    ecb[ecbno].SCpnt = SCpnt;
#ifdef DEBUG
    {
        int i;
        printk("aha1740_command: sending.. ");
        for (i = 0; i < sizeof(ecb[ecbno])-10; i++)
            printk("%02x ", ((unchar *)&ecb[ecbno])[i]);
    }
    printk("\n");
#endif
    if (done)
    { /*
       * You may question the code below, which contains potentially
       * non-terminating while loops with interrupts disabled.  So did
       * I when I wrote it, but the Adaptec Spec says the card is so
       * fast, that this problem virtually never occurs so I've kept
       * it.  We do printk a warning first, so that you'll know if it
       * happens.  In practice the only time we've seen this message
       * is when some- thing else is in the driver was broken, like
       * _makecode(), or when a scsi device hung the scsi bus.  Even
       * under these conditions, The loop actually only cycled < 3
       * times (we instrumented it).
       */

        DEB(printk("aha1740[%d] critical section\n",ecbno));
        save_flags(flags);
        cli();
        if ( ! (inb(G2STAT) & G2STAT_MBXOUT) )
        {
            printk("aha1740[%d]_mbxout wait!\n",ecbno);
            cli(); /* printk may have done a sti()! */
        }
        mb();
        while ( ! (inb(G2STAT) & G2STAT_MBXOUT) );      /* Oh Well. */
        outl(virt_to_bus(ecb+ecbno), MBOXOUT0);
        if ( inb(G2STAT) & G2STAT_BUSY )
        {
            printk("aha1740[%d]_attn wait!\n",ecbno);
```

```
            cli();
        }
        while ( inb(G2STAT) & G2STAT_BUSY );            /* And Again! */
        outb(ATTN_START | (target & 7), ATTN);  /* Start it up */
        restore_flags(flags);
        DEB(printk("aha1740[%d] request queued.\n",ecbno));
    }
    else
      printk("aha1740_queuecommand: done can’t be NULL\n");

    return 0;
}

static volatile int internal_done_flag = 0;
static volatile int internal_done_errcode = 0;

static void internal_done(Scsi_Cmnd * SCpnt)
{
    internal_done_errcode = SCpnt->result;
    ++internal_done_flag;
}

int aha1740_command(Scsi_Cmnd * SCpnt)
{
    aha1740_queuecommand(SCpnt, internal_done);

    while (!internal_done_flag);
    internal_done_flag = 0;
    return internal_done_errcode;
}

/* Query the board for its irq_level.  Nothing else matters
   in enhanced mode on an EISA bus. */

void aha1740_getconfig(void)
{
  static int intab[] = { 9,10,11,12,0,14,15,0 };

  irq_level = intab [ inb(INTDEF)&0x7 ];
  outb(inb(INTDEF) | 0x10, INTDEF);
}

int aha1740_detect(Scsi_Host_Template * tpnt)
{
    tpnt->proc_dir = &proc_scsi_aha1740;

    memset(&ecb, 0, sizeof(struct ecb));
    DEB(printk("aha1740_detect: \n"));

    for ( slot=MINEISA; slot <= MAXEISA; slot++ )
    {
        base = SLOTBASE(slot);
        /*
         * The ioports for eisa boards are generally beyond that used
         * in the check/allocate region code, but this may change at
         * some point, so we go through the motions.
         */
        if(check_region(base, 0x5c)) continue;  /* See if in use */
        if ( aha1740_test_port())  break;
    }
```

```
    if ( slot > MAXEISA )
        return 0;

    aha1740_getconfig();

    if ( (inb(G2STAT) & (G2STAT_MBXOUT | G2STAT_BUSY) ) != G2STAT_MBXOUT )
    {   /* If the card isn't ready, hard reset it */
        outb(G2CNTRL_HRST,G2CNTRL);
        outb(0,G2CNTRL);
    }

    printk("Configuring Adaptec at IO:%x, IRQ %d\n",base,
           irq_level);

    DEB(printk("aha1740_detect: enable interrupt channel %d\n", \
               irq_level));

    if (request_irq(irq_level,aha1740_intr_handle, 0, "aha1740", NULL))
    {
        printk("Unable to allocate IRQ for adaptec controller.\n");
        return 0;
    }
    /* Reserve the space that we need to use */
    request_region(base, 0x5c,"aha1740");
    return 1;
}

/*
 * Note: They following two functions do not apply very well to the
 * Adaptec, which basically manages its own affairs quite well without
 * our interference, so I haven't put anything into them.  I can
 * faintly imagine someone with a *very* badly behaved SCSI target
 * (perhaps an old tape?) wanting the abort(), but it hasn't happened
 * yet, and doing aborts brings the Adaptec to its knees.  I cannot
 * (at this moment in time) think of any reason to reset the card once
 * it's running.  So there.
 */
int aha1740_abort(Scsi_Cmnd * SCpnt)
{
    DEB(printk("aha1740_abort called\n"));
    return SCSI_ABORT_SNOOZE;
}

/*
 * We do not implement a reset function here, but the upper level code
 * assumes that it will get some kind of response for the command in
 * SCpnt.  We must oblige, or the command will hang the scsi system
 */
int aha1740_reset(Scsi_Cmnd * SCpnt)
{
    DEB(printk("aha1740_reset called\n"));
    return SCSI_RESET_PUNT;
}

int aha1740_biosparam(Disk * disk, kdev_t dev, int* ip)
{
  int size = disk->capacity;
DEB(printk("aha1740_biosparam\n"));
  ip[0] = 64;
  ip[1] = 32;
```

```
  ip[2] = size >> 11;
/*  if (ip[2] >= 1024) ip[2] = 1024; */
  return 0;
}

#ifdef MODULE
/* Eventually this will go into an include file, but this will be later */
Scsi_Host_Template driver_template = AHA1740;

#include "scsi_module.c"
#endif

/*
 * Okay, you made it all the way through.  As of this writing,
 * 3/31/93, I'm brad@saturn.gaylord.com or brad@bradpc.gaylord.com.
 * I'll try to help as time permits if you have any trouble with this
 * driver.  Happy Linuxing!
 */
```