```c
/*
 * Logitech Bus Mouse Driver for Linux
 * by James Banks
 *
 * Mods by Matthew Dillon
 *   calls verify_area()
 *   tracks better when X is busy or paging
 *
 * Heavily modified by David Giller
 *   changed from queue- to counter- driven
 *   hacked out a (probably incorrect) mouse_select
 *
 * Modified again by Nathan Laredo to interface with
 *   0.96c-pl1 IRQ handling changes (13JUL92)
 *   didn’t bother touching select code.
 *
 * Modified the select() code blindly to conform to the VFS
 *   requirements. 92.07.14 - Linus. Somebody should test it out.
 *
 * Modified by Johan Myreen to make room for other mice (9AUG92)
 *   removed assignment chr_fops[10] = &mouse_fops; see mouse.c
 *   renamed mouse_fops => bus_mouse_fops, made bus_mouse_fops public.
 *   renamed this file mouse.c => busmouse.c
 *
 * Minor addition by Cliff Matthews
 *   added fasync support
 *
 * Modularised 6-Sep-95 Philip Blundell <pjb27@cam.ac.uk>
 *
 * Replaced dumb busy loop with udelay()  16 Nov 95
 *   Nathan Laredo <laredo@gnu.ai.mit.edu>
 *
 * Track I/O ports with request_region().  12 Dec 95 Philip Blundell
 */

#include <linux/module.h>

#include <linux/kernel.h>
#include <linux/sched.h>
#include <linux/busmouse.h>
#include <linux/signal.h>
#include <linux/errno.h>
#include <linux/mm.h>
#include <linux/miscdevice.h>
#include <linux/random.h>
#include <linux/delay.h>
#include <linux/ioport.h>

#include <asm/io.h>
#include <asm/segment.h>
#include <asm/system.h>
#include <asm/irq.h>

static struct mouse_status mouse;
static int mouse_irq = MOUSE_IRQ;

void bmouse_setup(char *str, int *ints)
{
        if (ints[0] > 0)
                mouse_irq=ints[1];
```

```
}

static void mouse_interrupt(int irq, void *dev_id, struct pt_regs *regs)
{
        char dx, dy;
        unsigned char buttons;

        outb(MSE_READ_X_LOW, MSE_CONTROL_PORT);
        dx = (inb(MSE_DATA_PORT) & 0xf);
        outb(MSE_READ_X_HIGH, MSE_CONTROL_PORT);
        dx |= (inb(MSE_DATA_PORT) & 0xf) << 4;
        outb(MSE_READ_Y_LOW, MSE_CONTROL_PORT );
        dy = (inb(MSE_DATA_PORT) & 0xf);
        outb(MSE_READ_Y_HIGH, MSE_CONTROL_PORT);
        buttons = inb(MSE_DATA_PORT);
        dy |= (buttons & 0xf) << 4;
        buttons = ((buttons >> 5) & 0x07);
        if (dx != 0 || dy != 0 || buttons != mouse.buttons) {
          add_mouse_randomness((buttons << 16) + (dy << 8) + dx);
          mouse.buttons = buttons;
          mouse.dx += dx;
          mouse.dy -= dy;
          mouse.ready = 1;
          wake_up_interruptible(&mouse.wait);

          /*
           * keep dx/dy reasonable, but still able to track when X (or
           * whatever) must page or is busy (i.e. long waits between
           * reads)
           */
          if (mouse.dx < -2048)
              mouse.dx = -2048;
          if (mouse.dx >  2048)
              mouse.dx =  2048;

          if (mouse.dy < -2048)
              mouse.dy = -2048;
          if (mouse.dy >  2048)
              mouse.dy =  2048;

          if (mouse.fasyncptr)
              kill_fasync(mouse.fasyncptr, SIGIO);
        }
        MSE_INT_ON();
}

static int fasync_mouse(struct inode *inode, struct file *filp, int on)
{
        int retval;

        retval = fasync_helper(inode, filp, on, &mouse.fasyncptr);
        if (retval < 0)
                return retval;
        return 0;
}

/*
 * close access to the mouse
 */
```

```
static void close_mouse(struct inode * inode, struct file * file)
{
        fasync_mouse(inode, file, 0);
        if (--mouse.active)
                return;
        MSE_INT_OFF();
        free_irq(mouse_irq, NULL);
        MOD_DEC_USE_COUNT;
}

/*
 * open access to the mouse
 */

static int open_mouse(struct inode * inode, struct file * file)
{
        if (!mouse.present)
                return -EINVAL;
        if (mouse.active++)
                return 0;
        if (request_irq(mouse_irq, mouse_interrupt, 0, "busmouse", NULL)) {
                mouse.active--;
                return -EBUSY;
        }
        mouse.ready = 0;
        mouse.dx = 0;
        mouse.dy = 0;
        mouse.buttons = 0x87;
        MOD_INC_USE_COUNT;
        MSE_INT_ON();
        return 0;
}

/*
 * writes are disallowed
 */

static int write_mouse(struct inode * inode, struct file * file,
                       const char * buffer, int count)
{
        return -EINVAL;
}

/*
 * read mouse data.  Currently never blocks.
 */

static int read_mouse(struct inode * inode, struct file * file,
                      char * buffer, int count)
{
        int r;
        int dx;
        int dy;
        unsigned char buttons;
        /* long flags; */

        if (count < 3)
                return -EINVAL;
        if ((r = verify_area(VERIFY_WRITE, buffer, count)))
                return r;
```

```
        if (!mouse.ready)
                return -EAGAIN;

        /*
         * Obtain the current mouse parameters and limit as
         * appropriate for the return data format.  Interrupts are
         * only disabled while obtaining the parameters, NOT during
         * the puts_fs_byte() calls, so paging in put_user() does not
         * effect mouse tracking.
         */

        /* save_flags(flags); cli(); */
        disable_irq(mouse_irq);
        dx = mouse.dx;
        dy = mouse.dy;
        if (dx < -127)
            dx = -127;
        if (dx > 127)
            dx = 127;
        if (dy < -127)
            dy = -127;
        if (dy > 127)
            dy = 127;
        buttons = mouse.buttons;
        mouse.dx -= dx;
        mouse.dy -= dy;
        mouse.ready = 0;
        enable_irq(mouse_irq);
        /* restore_flags(flags); */

        put_user(buttons | 0x80, buffer);
        put_user((char)dx, buffer + 1);
        put_user((char)dy, buffer + 2);
        for (r = 3; r < count; r++)
            put_user(0x00, buffer + r);
        return r;
}

/*
 * select for mouse input
 */
static int mouse_select(struct inode *inode, struct file *file,
                        int sel_type, select_table * wait)
{
        if (sel_type == SEL_IN) {
                if (mouse.ready)
                        return 1;
                select_wait(&mouse.wait, wait);
        }
        return 0;
}

struct file_operations bus_mouse_fops = {
        NULL,           /* mouse_seek */
        read_mouse,
        write_mouse,
        NULL,           /* mouse_readdir */
        mouse_select,   /* mouse_select */
        NULL,           /* mouse_ioctl */
        NULL,           /* mouse_mmap */
```

```
        open_mouse,
        close_mouse,
        NULL,
        fasync_mouse,
};

static struct miscdevice bus_mouse = {
        LOGITECH_BUSMOUSE, "busmouse", &bus_mouse_fops
};

int bus_mouse_init(void)
{
        if (check_region(LOGIBM_BASE, LOGIBM_EXTENT)) {
          mouse.present = 0;
          return -EIO;
        }

        outb(MSE_CONFIG_BYTE, MSE_CONFIG_PORT);
        outb(MSE_SIGNATURE_BYTE, MSE_SIGNATURE_PORT);
        udelay(100L);     /* wait for reply from mouse */
        if (inb(MSE_SIGNATURE_PORT) != MSE_SIGNATURE_BYTE) {
                mouse.present = 0;
                return -EIO;
        }
        outb(MSE_DEFAULT_MODE, MSE_CONFIG_PORT);
        MSE_INT_OFF();

        request_region(LOGIBM_BASE, LOGIBM_EXTENT, "busmouse");

        mouse.present = 1;
        mouse.active = 0;
        mouse.ready = 0;
        mouse.buttons = 0x87;
        mouse.dx = 0;
        mouse.dy = 0;
        mouse.wait = NULL;
        printk(KERN_INFO "Logitech bus mouse detected, using IRQ %d.\n",
               mouse_irq);
        misc_register(&bus_mouse);
        return 0;
}

#ifdef MODULE

int init_module(void)
{
        return bus_mouse_init();
}

void cleanup_module(void)
{
        misc_deregister(&bus_mouse);
        release_region(LOGIBM_BASE, LOGIBM_EXTENT);
}
#endif
```