

Linux-Kernel: [patch] updated hdaps driver.

[patch] updated hdaps driver.

Source: <http://linux.derkeiler.com/Mailing-Lists/Kernel/2005-09/0077.html>

From: Robert Love (rml_at_novell.com)

Date: 09/01/05

To: linux-kernel@vger.kernel.org

Date: Wed, 31 Aug 2005 23:57:59 -0400

Below find an updated hdaps driver.

Various bug fixes, clean ups, additions to the DMI whitelist, and a new automatic inversion detector (some ThinkPads have the axes negated).

Andrew, since a new 2.6-mm has yet to come out, feel free to replace the original patch with this one.

Thanks,

Robert Love

Driver for the IBM Hard Drive Active Protection System (HDAPS), an accelerometer found in most modern ThinkPads.

Signed-off-by: Robert Love <rml@novell.com>

```
diff -urN linux-2.6.13/drivers/hwmon/hdaps.c linux/drivers/hwmon/hdaps.c
--- linux-2.6.13/drivers/hwmon/hdaps.c 1969-12-31 19:00:00.000000000 -0500
+++ linux/drivers/hwmon/hdaps.c 2005-08-31 23:50:36.000000000 -0400
@@ -0,0 +1,739 @@
+/*
+ * drivers/hwmon/hdaps.c - driver for IBM's Hard Drive Active Protection System
+ *
+ * Copyright (C) 2005 Robert Love <rml@novell.com>
+ * Copyright (C) 2005 Jesper Juhl <jesper.juhl@gmail.com>
+ *
+ * The HardDisk Active Protection System (hdaps) is present in the IBM ThinkPad
+ * T41, T42, T43, R51, and X40, at least. It provides a basic two-axis
+ * accelerometer and other data, such as the device's temperature.
+ *
+ * Based on the document by Mark A. Smith available at
+ * http://www.almaden.ibm.com/cs/people/marksmith/tpaps.html and a lot of trial
+ * and error.
+ *
+ * This program is free software; you can redistribute it and/or modify it
+ * under the terms of the GNU General Public License v2 as published by the
```

[patch] updated hdaps driver.

Linux–Kernel: [patch] updated hdaps driver.

```
+ * Free Software Foundation.
+ *
+ * This program is distributed in the hope that it will be useful, but WITHOUT
+ * ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or
+ * FITNESS FOR A PARTICULAR PURPOSE. See the GNU General Public License for
+ * more details.
+ *
+ * You should have received a copy of the GNU General Public License along with
+ * this program; if not, write to the Free Software Foundation, Inc.,
+ * 51 Franklin Street, Fifth Floor, Boston, MA 02110–1301, USA
+ */
+
+#include <linux/delay.h>
+#include <linux/device.h>
+#include <linux/input.h>
+#include <linux/kernel.h>
+#include <linux/module.h>
+#include <linux/timer.h>
+#include <linux/dmi.h>
+#include <asm/io.h>
+
+#define HDAPS_LOW_PORT 0x1600 /* first port used by hdaps */
+#define HDAPS_NR_PORTS 0x30 /* 0x1600 – 0x162f */
+
+#define STATE_FRESH 0x50 /* accelerometer data is fresh */
+
+#define REFRESH_ASYNC 0x00 /* do asynchronous refresh */
+#define REFRESH_SYNC 0x01 /* do synchronous refresh */
+
+#define HDAPS_PORT_STATE 0x1611 /* device state */
+#define HDAPS_PORT_YPOS 0x1612 /* y–axis position */
+#define HDAPS_PORT_XPOS 0x1614 /* x–axis position */
+#define HDAPS_PORT_TEMP1 0x1616 /* device temperature, in celcius */
+#define HDAPS_PORT_YVAR 0x1617 /* y–axis variance (what is this?) */
+#define HDAPS_PORT_XVAR 0x1619 /* x–axis variance (what is this?) */
+#define HDAPS_PORT_TEMP2 0x161b /* device temperature (again?) */
+#define HDAPS_PORT_UNKNOWN 0x161c /* what is this? */
+#define HDAPS_PORT_KMACT 0x161d /* keyboard or mouse activity */
+
+#define HDAPS_READ_MASK 0xff /* some reads have the low 8 bits set */
+
+#define KEYBD_MASK 0x20 /* set if keyboard activity */
+#define MOUSE_MASK 0x40 /* set if mouse activity */
+#define KEYBD_ISSET(n) (!! (n & KEYBD_MASK)) /* keyboard used? */
+#define MOUSE_ISSET(n) (!! (n & MOUSE_MASK)) /* mouse used? */
+
+#define INIT_TIMEOUT_MSECS 4000 /* wait up to 4s for device init ... */
+#define INIT_WAIT_MSECS 200 /* ... in 200ms increments */
+
+static struct platform_device *pdev;
+static struct input_dev hdaps_idev;
```

[patch] updated hdaps driver.

Linux–Kernel: [patch] updated hdaps driver.

```
+static struct timer_list hdaps_timer;
+static unsigned int hdaps_mousedev_threshold = 4;
+static unsigned long hdaps_poll_ms = 50;
+static unsigned int hdaps_mousedev;
+static unsigned int hdaps_invert;
+static u8 km_activity;
+static int rest_x;
+static int rest_y;
+
+static DECLARE_MUTEX(hdaps_sem);
+
+/*
+ * __get_latch – Get the value from a given port. Callers must hold hdaps_sem.
+ */
+static inline u8 __get_latch(u16 port)
+{
+ return inb(port) & HDAPS_READ_MASK;
+}
+
+/*
+ * __check_latch – Check a port latch for a given value. Callers must hold
+ * hdaps_sem. Returns zero if the port contains the given value.
+ */
+static inline unsigned int __check_latch(u16 port, u8 val)
+{
+ if (__get_latch(port) == val)
+ return 0;
+ return -EINVAL;
+}
+
+/*
+ * __wait_latch – Wait up to 100us for a port latch to get a certain value,
+ * returning zero if the value is obtained. Callers must hold hdaps_sem.
+ */
+static unsigned int __wait_latch(u16 port, u8 val)
+{
+ unsigned int i;
+
+ for (i = 0; i < 20; i++) {
+ if (!__check_latch(port, val))
+ return 0;
+ udelay(5);
+ }
+
+ return -EINVAL;
+}
+
+/*
+ * __device_refresh – Request a refresh from the accelerometer.
+ *
+ * If sync is REFRESH_SYNC, we perform a synchronous refresh and will wait.
```

[patch] updated hdaps driver.

Linux–Kernel: [patch] updated hdaps driver.

```
+ * Returns zero if successful and nonzero on error.
+ *
+ * If sync is REFRESH_ASYNC, we merely kick off a new refresh if the device is
+ * not up-to-date. Always returns zero.
+ *
+ * Callers must hold hdaps_sem.
+ */
+static int __device_refresh(unsigned int sync)
+{
+ u8 state;
+
+ udelay(100);
+
+ state = inb(0x1604);
+ if (state == STATE_FRESH)
+ return 0;
+
+ outb(0x11, 0x1610);
+ outb(0x01, 0x161f);
+ if (sync == REFRESH_ASYNC)
+ return 0;
+
+ return __wait_latch(0x1604, STATE_FRESH);
+}
+
+/*
+ * __device_complete – Indicate to the accelerometer that we are done reading
+ * data, and then initiate an async refresh. Callers must hold hdaps_sem.
+ */
+static inline void __device_complete(void)
+{
+ inb(0x161f);
+ inb(0x1604);
+ __device_refresh(REFRESH_ASYNC);
+}
+
+static int __hdaps_readb_one(unsigned int port, u8 *val)
+{
+ /* do a sync refresh -- we need to be sure that we read fresh data */
+ if (__device_refresh(REFRESH_SYNC))
+ return -EIO;
+
+ *val = inb(port);
+ __device_complete();
+
+ return 0;
+}
+
+/*
+ * hdaps_readb_one – reads a byte from a single I/O port, placing the value in
+ * the given pointer. Returns zero on success or a negative error on failure.
```

[patch] updated hdaps driver.

Linux–Kernel: [patch] updated hdaps driver.

```
+ * Can sleep.
+ */
+static int hdaps_readb_one(unsigned int port, u8 *val)
+{
+ int ret;
+
+ down(&hdaps_sem);
+ ret = __hdaps_readb_one(port, val);
+ up(&hdaps_sem);
+
+ return ret;
+}
+
+static int __hdaps_read_pair(unsigned int port1, unsigned int port2,
+ int *x, int *y)
+{
+ /* do a sync refresh --- we need to be sure that we read fresh data */
+ if (__device_refresh(REFRESH_SYNC))
+ return -EIO;
+
+ *y = inw(port2);
+ *x = inw(port1);
+ km_activity = inb(HDAPS_PORT_KMACT);
+ __device_complete();
+
+ /* if hdaps_invert is set, negate the two values */
+ if (hdaps_invert) {
+ *x = -*x;
+ *y = -*y;
+ }
+
+ return 0;
+}
+
+/*
+ * hdaps_read_pair – reads the values from a pair of ports, placing the values
+ * in the given pointers. Returns zero on success. Can sleep.
+ */
+static int hdaps_read_pair(unsigned int port1, unsigned int port2,
+ int *val1, int *val2)
+{
+ int ret;
+
+ down(&hdaps_sem);
+ ret = __hdaps_read_pair(port1, port2, val1, val2);
+ up(&hdaps_sem);
+
+ return ret;
+}
+
+/* initialize the accelerometer */
```

Linux–Kernel: [patch] updated hdaps driver.

```
+static int hdaps_device_init(void)
+{
+ unsigned int total_msecs = INIT_TIMEOUT_MSECS;
+ int ret = -ENXIO;
+
+ down(&hdaps_sem);
+
+ outb(0x13, 0x1610);
+ outb(0x01, 0x161f);
+ if (__wait_latch(0x161f, 0x00))
+ goto out;
+
+ /*
+ * The 0x03 value appears to only work on some thinkpads, such as the
+ * T42p. Others return 0x01.
+ *
+ * The 0x02 value occurs when the chip has been previously initialized.
+ */
+ if (__check_latch(0x1611, 0x03) &&
+ __check_latch(0x1611, 0x02) &&
+ __check_latch(0x1611, 0x01))
+ goto out;
+
+ printk(KERN_DEBUG "hdaps: initial latch check good (0x%02x).\n",
+ __get_latch(0x1611));
+
+ outb(0x17, 0x1610);
+ outb(0x81, 0x1611);
+ outb(0x01, 0x161f);
+ if (__wait_latch(0x161f, 0x00))
+ goto out;
+ if (__wait_latch(0x1611, 0x00))
+ goto out;
+ if (__wait_latch(0x1612, 0x60))
+ goto out;
+ if (__wait_latch(0x1613, 0x00))
+ goto out;
+ outb(0x14, 0x1610);
+ outb(0x01, 0x1611);
+ outb(0x01, 0x161f);
+ if (__wait_latch(0x161f, 0x00))
+ goto out;
+ outb(0x10, 0x1610);
+ outb(0xc8, 0x1611);
+ outb(0x00, 0x1612);
+ outb(0x02, 0x1613);
+ outb(0x01, 0x161f);
+ if (__wait_latch(0x161f, 0x00))
+ goto out;
+ if (__device_refresh(REFRESH_SYNC))
+ goto out;
```

[patch] updated hdaps driver.

Linux–Kernel: [patch] updated hdaps driver.

```
+ if (__wait_latch(0x1611, 0x00))
+ goto out;
+
+ /* we have done our dance, now let's wait for the applause */
+ while (total_msecs > 0) {
+ u8 ignored;
+
+ /* a read of the device helps push it into action */
+ __hdaps_readb_one(HDAPS_PORT_UNKNOWN, &ignored);
+ if (!__wait_latch(0x1611, 0x02)) {
+ ret = 0;
+ break;
+ }
+
+ msleep(INIT_WAIT_MSECS);
+ total_msecs -= INIT_WAIT_MSECS;
+ }
+
+out:
+ up(&hdaps_sem);
+ return ret;
+}
+
+
+/* Input class stuff */
+
+/*
+ * hdaps_calibrate – Zero out our "resting" values. Callers must hold hdaps_sem.
+ */
+static void hdaps_calibrate(void)
+{
+ int x, y;
+
+ if (__hdaps_read_pair(HDAPS_PORT_XPOS, HDAPS_PORT_YPOS, &x, &y))
+ return;
+
+ rest_x = x;
+ rest_y = y;
+}
+
+static void hdaps_mousedev_poll(unsigned long unused)
+{
+ int x, y;
+
+ /* Cannot sleep. Try nonblockingly. If we fail, try again later. */
+ if (down_trylock(&hdaps_sem)) {
+ mod_timer(&hdaps_timer, jiffies+msecs_to_jiffies(hdaps_poll_ms));
+ return;
+ }
+
+ if (__hdaps_read_pair(HDAPS_PORT_XPOS, HDAPS_PORT_YPOS, &x, &y))
```

[patch] updated hdaps driver.

Linux–Kernel: [patch] updated hdaps driver.

```
+ goto out;
+
+ x -= rest_x;
+ y -= rest_y;
+ if (abs(x) > hdaps_mousedev_threshold)
+ input_report_rel(&hdaps_idev, REL_X, x);
+ if (abs(y) > hdaps_mousedev_threshold)
+ input_report_rel(&hdaps_idev, REL_Y, y);
+ input_sync(&hdaps_idev);
+
+ mod_timer(&hdaps_timer, jiffies + msecs_to_jiffies(hdaps_poll_ms));
+
+out:
+ up(&hdaps_sem);
+}
+
+/*
+ * hdaps_mousedev_enable – enable the input class device. Can sleep.
+ */
+static void hdaps_mousedev_enable(void)
+{
+ down(&hdaps_sem);
+
+ /* calibrate the device before enabling */
+ hdaps_calibrate();
+
+ /* initialize the input class */
+ init_input_dev(&hdaps_idev);
+ hdaps_idev.dev = &pdev->dev;
+ hdaps_idev.evbit[0] = BIT(EV_KEY) | BIT(EV_REL);
+ hdaps_idev.relbit[0] = BIT(REL_X) | BIT(REL_Y);
+ hdaps_idev.keybit[LONG(BTN_LEFT)] = BIT(BTN_LEFT);
+ input_register_device(&hdaps_idev);
+
+ /* start up our timer */
+ init_timer(&hdaps_timer);
+ hdaps_timer.function = hdaps_mousedev_poll;
+ hdaps_timer.expires = jiffies + msecs_to_jiffies(hdaps_poll_ms);
+ add_timer(&hdaps_timer);
+
+ hdaps_mousedev = 1;
+
+ up(&hdaps_sem);
+
+ printk(KERN_INFO "hdaps: input device enabled.\n");
+}
+
+/*
+ * hdaps_mousedev_disable – disable the input class device. Caller must hold
+ * hdaps_sem.
+ */
```

[patch] updated hdaps driver.

Linux–Kernel: [patch] updated hdaps driver.

```
+static void hdaps_mousedev_disable(void)
+{
+ down(&hdaps_sem);
+ if (hdaps_mousedev) {
+ hdaps_mousedev = 0;
+ del_timer_sync(&hdaps_timer);
+ input_unregister_device(&hdaps_idev);
+ }
+ up(&hdaps_sem);
+}
+
+
+/* Device model stuff */
+
+static int hdaps_probe(struct device *dev)
+{
+ int ret;
+
+ ret = hdaps_device_init();
+ if (ret)
+ return ret;
+
+ printk(KERN_INFO "hdaps: device successfully initialized.\n");
+ return 0;
+}
+
+static int hdaps_resume(struct device *dev, u32 level)
+{
+ if (level == RESUME_ENABLE)
+ return hdaps_device_init();
+ return 0;
+}
+
+static struct device_driver hdaps_driver = {
+ .name = "hdaps",
+ .bus = &platform_bus_type,
+ .owner = THIS_MODULE,
+ .probe = hdaps_probe,
+ .resume = hdaps_resume
+};
+
+
+/* Sysfs Files */
+
+static ssize_t hdaps_position_show(struct device *dev,
+ struct device_attribute *attr, char *buf)
+{
+ int ret, x, y;
+
+ ret = hdaps_read_pair(HDAPS_PORT_XPOS, HDAPS_PORT_YPOS, &x, &y);
+ if (ret)
```

[patch] updated hdaps driver.

Linux–Kernel: [patch] updated hdaps driver.

```
+ return ret;
+
+ return sprintf(buf, "(%d,%d)\n", x, y);
+}
+
+static ssize_t hdaps_variance_show(struct device *dev,
+ struct device_attribute *attr, char *buf)
+{
+ int ret, x, y;
+
+ ret = hdaps_read_pair(HDAPS_PORT_XVAR, HDAPS_PORT_YVAR, &x, &y);
+ if (ret)
+ return ret;
+
+ return sprintf(buf, "(%d,%d)\n", x, y);
+}
+
+static ssize_t hdaps_temp1_show(struct device *dev,
+ struct device_attribute *attr, char *buf)
+{
+ u8 temp;
+ int ret;
+
+ ret = hdaps_readb_one(HDAPS_PORT_TEMP1, &temp);
+ if (ret < 0)
+ return ret;
+
+ return sprintf(buf, "%u\n", temp);
+}
+
+static ssize_t hdaps_temp2_show(struct device *dev,
+ struct device_attribute *attr, char *buf)
+{
+ u8 temp;
+ int ret;
+
+ ret = hdaps_readb_one(HDAPS_PORT_TEMP2, &temp);
+ if (ret < 0)
+ return ret;
+
+ return sprintf(buf, "%u\n", temp);
+}
+
+static ssize_t hdaps_keyboard_activity_show(struct device *dev,
+ struct device_attribute *attr,
+ char *buf)
+{
+ return sprintf(buf, "%u\n", KEYBD_ISSET(km_activity));
+}
+
+static ssize_t hdaps_mouse_activity_show(struct device *dev,
```

[patch] updated hdaps driver.

Linux–Kernel: [patch] updated hdaps driver.

```
+ struct device_attribute *attr,
+ char *buf)
+{
+ return sprintf(buf, "%u\n", MOUSE_ISSET(km_activity));
+}
+
+static ssize_t hdaps_calibrate_show(struct device *dev,
+ struct device_attribute *attr, char *buf)
+{
+ return sprintf(buf, "(%d,%d)\n", rest_x, rest_y);
+}
+
+static ssize_t hdaps_calibrate_store(struct device *dev,
+ struct device_attribute *attr,
+ const char *buf, size_t count)
+{
+ down(&hdaps_sem);
+ hdaps_calibrate();
+ up(&hdaps_sem);
+
+ return count;
+}
+
+static ssize_t hdaps_invert_show(struct device *dev,
+ struct device_attribute *attr, char *buf)
+{
+ return sprintf(buf, "%u\n", hdaps_invert);
+}
+
+static ssize_t hdaps_invert_store(struct device *dev,
+ struct device_attribute *attr,
+ const char *buf, size_t count)
+{
+ int invert;
+
+ if (sscanf(buf, "%d", &invert) != 1 || (invert != 1 && invert != 0))
+ return -EINVAL;
+
+ hdaps_invert = invert;
+ hdaps_calibrate();
+
+ return count;
+}
+
+static ssize_t hdaps_mousedev_show(struct device *dev,
+ struct device_attribute *attr, char *buf)
+{
+ return sprintf(buf, "%d\n", hdaps_mousedev);
+}
+
+static ssize_t hdaps_mousedev_store(struct device *dev,
```

[patch] updated hdaps driver.

Linux–Kernel: [patch] updated hdaps driver.

```
+ struct device_attribute *attr,  
+ const char *buf, size_t count)  
+ {  
+ int enable;  
+  
+ if (sscanf(buf, "%d", &enable) != 1)  
+ return -EINVAL;  
+  
+ if (enable == 1)  
+ hdaps_mousedev_enable();  
+ else if (enable == 0)  
+ hdaps_mousedev_disable();  
+ else  
+ return -EINVAL;  
+  
+ return count;  
+ }  
+  
+static ssize_t hdaps_poll_show(struct device *dev,  
+ struct device_attribute *attr, char *buf)  
+ {  
+ return sprintf(buf, "%lu\n", hdaps_poll_ms);  
+ }  
+  
+static ssize_t hdaps_poll_store(struct device *dev,  
+ struct device_attribute *attr,  
+ const char *buf, size_t count)  
+ {  
+ unsigned int poll;  
+  
+ if (sscanf(buf, "%u", &poll) != 1 || poll == 0)  
+ return -EINVAL;  
+ hdaps_poll_ms = poll;  
+  
+ return count;  
+ }  
+  
+static ssize_t hdaps_threshold_show(struct device *dev,  
+ struct device_attribute *attr, char *buf)  
+ {  
+ return sprintf(buf, "%u\n", hdaps_mousedev_threshold);  
+ }  
+  
+static ssize_t hdaps_threshold_store(struct device *dev,  
+ struct device_attribute *attr,  
+ const char *buf, size_t count)  
+ {  
+ unsigned int threshold;  
+  
+ if (sscanf(buf, "%u", &threshold) != 1 || threshold == 0)  
+ return -EINVAL;
```

[patch] updated hdaps driver.

Linux–Kernel: [patch] updated hdaps driver.

```
+ hdaps_mousedev_threshold = threshold;
+
+ return count;
+}
+
+static DEVICE_ATTR(position, 0444, hdaps_position_show, NULL);
+static DEVICE_ATTR(variance, 0444, hdaps_variance_show, NULL);
+static DEVICE_ATTR(temp1, 0444, hdaps_temp1_show, NULL);
+static DEVICE_ATTR(temp2, 0444, hdaps_temp2_show, NULL);
+static DEVICE_ATTR(keyboard_activity, 0444, hdaps_keyboard_activity_show, NULL);
+static DEVICE_ATTR(mouse_activity, 0444, hdaps_mouse_activity_show, NULL);
+static DEVICE_ATTR(calibrate, 0644, hdaps_calibrate_show,hdaps_calibrate_store);
+static DEVICE_ATTR(invert, 0644, hdaps_invert_show, hdaps_invert_store);
+static DEVICE_ATTR(mousedev, 0644, hdaps_mousedev_show, hdaps_mousedev_store);
+static DEVICE_ATTR(mousedev_poll_ms, 0644, hdaps_poll_show, hdaps_poll_store);
+static DEVICE_ATTR(mousedev_threshold, 0644, hdaps_threshold_show,
+ hdaps_threshold_store);
+
+static struct attribute *hdaps_attributes[] = {
+ &dev_attr_position.attr,
+ &dev_attr_variance.attr,
+ &dev_attr_temp1.attr,
+ &dev_attr_temp2.attr,
+ &dev_attr_keyboard_activity.attr,
+ &dev_attr_mouse_activity.attr,
+ &dev_attr_calibrate.attr,
+ &dev_attr_mousedev.attr,
+ &dev_attr_mousedev_threshold.attr,
+ &dev_attr_mousedev_poll_ms.attr,
+ &dev_attr_invert.attr,
+ NULL,
+};
+
+static struct attribute_group hdaps_attribute_group = {
+ .attrs = hdaps_attributes,
+};
+
+
+/* Module stuff */
+
+/*
+ * XXX: We should be able to return nonzero and halt the detection process.
+ * But there is a bug in dmi_check_system() where a nonzero return from the
+ * first match will result in a return of failure from dmi_check_system().
+ * I fixed this; the patch is in 2.6–mm. Once in Linus's tree we can make
+ * hdaps_dmi_match_invert() return hdaps_dmi_match(), which in turn returns 1.
+ */
+static int hdaps_dmi_match(struct dmi_system_id *id)
+{
+ printk(KERN_INFO "hdaps: %s detected.\n", id->ident);
+ return 0;
+}
```

[patch] updated hdaps driver.

Linux–Kernel: [patch] updated hdaps driver.

```
+}
+
+static int hdaps_dmi_match_invert(struct dmi_system_id *id)
+{
+  hdaps_invert = 1;
+  printk(KERN_INFO "hdaps: inverting axis readings.\n");
+  return 0;
+}
+
+#define HDAPS_DMI_MATCH_NORMAL(model) { \
+ .ident = "IBM " model, \
+ .callback = hdaps_dmi_match, \
+ .matches = { \
+ DMI_MATCH(DMI_BOARD_VENDOR, "IBM"), \
+ DMI_MATCH(DMI_PRODUCT_VERSION, model) \
+ } \
+}
+
+#define HDAPS_DMI_MATCH_INVERT(model) { \
+ .ident = "IBM " model, \
+ .callback = hdaps_dmi_match_invert, \
+ .matches = { \
+ DMI_MATCH(DMI_BOARD_VENDOR, "IBM"), \
+ DMI_MATCH(DMI_PRODUCT_VERSION, model) \
+ } \
+}
+
+static int __init hdaps_init(void)
+{
+  int ret;
+
+  /* Note that DMI_MATCH(...,"ThinkPad T42") will match "ThinkPad T42p" */
+  struct dmi_system_id hdaps_whitelist[] = {
+  HDAPS_DMI_MATCH_INVERT("ThinkPad R50p"),
+  HDAPS_DMI_MATCH_NORMAL("ThinkPad R50"),
+  HDAPS_DMI_MATCH_NORMAL("ThinkPad R51"),
+  HDAPS_DMI_MATCH_INVERT("ThinkPad T41p"),
+  HDAPS_DMI_MATCH_NORMAL("ThinkPad T41"),
+  HDAPS_DMI_MATCH_INVERT("ThinkPad T42p"),
+  HDAPS_DMI_MATCH_NORMAL("ThinkPad T42"),
+  HDAPS_DMI_MATCH_NORMAL("ThinkPad T43"),
+  HDAPS_DMI_MATCH_NORMAL("ThinkPad X40"),
+  { .ident = NULL }
+  };
+
+  if (!dmi_check_system(hdaps_whitelist)) {
+  printk(KERN_WARNING "hdaps: supported laptop not found!\n");
+  ret = -ENXIO;
+  goto out;
+  }
+}
```

[patch] updated hdaps driver.

Linux-Kernel: [patch] updated hdaps driver.

```
+ if (!request_region(HDAPS_LOW_PORT, HDAPS_NR_PORTS, "hdaps")) {
+ ret = -ENXIO;
+ goto out;
+ }
+
+ ret = driver_register(&hdaps_driver);
+ if (ret)
+ goto out_region;
+
+ pdev = platform_device_register_simple("hdaps", -1, NULL, 0);
+ if (IS_ERR(pdev)) {
+ ret = PTR_ERR(pdev);
+ goto out_driver;
+ }
+
+ ret = sysfs_create_group(&pdev->dev.kobj, &hdaps_attribute_group);
+ if (ret)
+ goto out_device;
+
+ if (hdaps_mousedev)
+ hdaps_mousedev_enable();
+
+ printk(KERN_INFO "hdaps: driver successfully loaded.\n");
+ return 0;
+
+out_device:
+ platform_device_unregister(pdev);
+out_driver:
+ driver_unregister(&hdaps_driver);
+out_region:
+ release_region(HDAPS_LOW_PORT, HDAPS_NR_PORTS);
+out:
+ printk(KERN_WARNING "hdaps: driver init failed (ret=%d)!\n", ret);
+ return ret;
+}
+
+static void __exit hdaps_exit(void)
+{
+ hdaps_mousedev_disable();
+
+ sysfs_remove_group(&pdev->dev.kobj, &hdaps_attribute_group);
+ platform_device_unregister(pdev);
+ driver_unregister(&hdaps_driver);
+ release_region(HDAPS_LOW_PORT, HDAPS_NR_PORTS);
+
+ printk(KERN_INFO "hdaps: driver unloaded.\n");
+}
+
+module_init(hdaps_init);
+module_exit(hdaps_exit);
+
```

[patch] updated hdaps driver.

Linux-Kernel: [patch] updated hdaps driver.

```
+module_param_named(mousedev, hdaps_mousedev, bool, 0);
+MODULE_PARM_DESC(mousedev, "enable the input class device");
+
+module_param_named(invert, hdaps_invert, bool, 0);
+MODULE_PARM_DESC(invert, "invert data along each axis");
+
+MODULE_AUTHOR("Robert Love");
+MODULE_DESCRIPTION("IBM Hard Drive Active Protection System (HDAPS) driver");
+MODULE_LICENSE("GPL v2");
diff -urN linux-2.6.13/drivers/hwmon/Kconfig linux/drivers/hwmon/Kconfig
--- linux-2.6.13/drivers/hwmon/Kconfig 2005-08-28 19:41:01.000000000 -0400
+++ linux/drivers/hwmon/Kconfig 2005-08-31 23:46:19.000000000 -0400
@@ -407,6 +407,23 @@
```

This driver can also be built as a module. If so, the module will be called w83627ehf.

```
+config SENSORS_HDAPS
+ tristate "IBM Hard Drive Active Protection System (hdaps)"
+ depends on HWMON && INPUT && X86
+ default n
+ help
+ This driver provides support for the IBM Hard Drive Active Protection
+ System (hdaps), which provides an accelerometer and other misc. data.
+ Supported laptops include the IBM ThinkPad T41, T42, T43, and R51.
+ The accelerometer data is readable via sysfs.
+
+ This driver also provides an input class device, allowing the
+ laptop to act as a pinball machine-esque mouse. This is off by
+ default but enabled via sysfs or the module parameter "mousedev".
+
+ Say Y here if you have an applicable laptop and want to experience
+ the awesome power of hdaps.
+
+config HWMON_DEBUG_CHIP
+ bool "Hardware Monitoring Chip debugging messages"
+ depends on HWMON
diff -urN linux-2.6.13/drivers/hwmon/Makefile linux/drivers/hwmon/Makefile
--- linux-2.6.13/drivers/hwmon/Makefile 2005-08-28 19:41:01.000000000 -0400
+++ linux/drivers/hwmon/Makefile 2005-08-31 23:46:19.000000000 -0400
@@ -18,6 +18,7 @@
obj-$(CONFIG_SENSORS_FSCPOS) += fscpos.o
obj-$(CONFIG_SENSORS_GL518SM) += gl518sm.o
obj-$(CONFIG_SENSORS_GL520SM) += gl520sm.o
+obj-$(CONFIG_SENSORS_HDAPS) += hdaps.o
obj-$(CONFIG_SENSORS_IT87) += it87.o
obj-$(CONFIG_SENSORS_LM63) += lm63.o
obj-$(CONFIG_SENSORS_LM75) += lm75.o
diff -urN linux-2.6.13/MAINTAINERS linux/MAINTAINERS
--- linux-2.6.13/MAINTAINERS 2005-08-28 19:41:01.000000000 -0400
+++ linux/MAINTAINERS 2005-08-31 23:51:09.000000000 -0400
@@ -933,6 +933,13 @@
```

[patch] updated hdaps driver.

Linux-Kernel: [patch] updated hdaps driver.

W: <http://www.kernel.org/pub/linux/utils/net/hdlc/>

S: Maintained

+HARD DRIVE ACTIVE PROTECTION SYSTEM (HDAPS) DRIVER

+P: Robert Love

+M: rlove@rlove.org

+M: linux-kernel@vger.kernel.org

+W: <http://www.kernel.org/pub/linux/kernel/people/rml/hdaps/>

+S: Maintained

+

HARMONY SOUND DRIVER

P: Kyle McMartin

M: kyle@parisc-linux.org

-

To unsubscribe from this list: send the line "unsubscribe linux-kernel" in the body of a message to majordomo@vger.kernel.org

More majordomo info at <http://vger.kernel.org/majordomo-info.html>

Please read the FAQ at <http://www.tux.org/lkml/>