

# [PATCH][ACPI] Panasonic Hotkey Driver

**Source:** <http://linux.derkeiler.com/Mailing-Lists/Kernel/2004-07/6392.html>

---

**From:** Hiroshi Miura ([miura\\_at\\_da-cha.org](mailto:miura_at_da-cha.org))

**Date:** 07/31/04

Date: Sat, 31 Jul 2004 23:17:46 +0900  
To: Len Brown <[len.brown@intel.com](mailto:len.brown@intel.com)>

Hi,

This is a hotkey driver for Panasonic Lets note series.  
I tested this driver on 2.6.7 kernel and Panasonic CF-R3 laptop.  
Using this driver, users can handle Fn+Fx key (except for Fn+F3) with acpid.

The event is for example 'HKEY HKEY 00000080 0000001' for Fn+F1.

sample scripts as follows

```
# /etc/acpi/events/hotkey
event=HKEY.*
action=/etc/acpi/hotkey.sh "%e"
```

hotkey.sh is

```
#!/bin/sh
set $*
# Take care about the way events are reported
key="$4";

case "$key" in
    0000000a)
        logger "acpid: received a suspend request"
        echo 1 > /proc/swsusp/activate
        hwclock --hctosys
        break
        ;;
    *)
        logger "acpid: action $key is not defined"
        ;;
esac
```

patch is follows

```
# This is a BitKeeper generated diff -Nru style patch.
#
```

## Linux-Kernel: [PATCH][ACPI] Panasonic Hotkey Driver

```
# ChangeSet
# 2004/07/26 11:50:39+09:00 miura@da-cha.org
# Panasonic Laptop Extra driver.
# This adds support of hotkey (Fn+Fx) on Panasonic Lets note Laptop series.
# Loding this driver, user can get event through acpid like "HKEY HKEY 0000080 000001"
#
# Signed-off-by: Hiroshi Miura <miura@da-cha.org>
#
# drivers/acpi/pcc_acpi.c
# 2004/07/26 11:50:23+09:00 miura@da-cha.org +518 -0
#
# drivers/acpi/pcc_acpi.c
# 2004/07/26 11:50:23+09:00 miura@da-cha.org +0 -0
# BitKeeper file /home/miura/kernel/linux-2.6.7-hm/drivers/acpi/pcc_acpi.c
#
# drivers/acpi/Makefile
# 2004/07/26 11:50:23+09:00 miura@da-cha.org +1 -0
# add Panasonic Laptop Extra driver entry
#
# drivers/acpi/Kconfig
# 2004/07/26 11:50:23+09:00 miura@da-cha.org +20 -0
# add Panasonic Laptop Extra driver entry
#
diff -Nru a/drivers/acpi/Kconfig b/drivers/acpi/Kconfig
--- a/drivers/acpi/Kconfig 2004-07-27 12:44:53 +09:00
+++ b/drivers/acpi/Kconfig 2004-07-27 12:44:53 +09:00
@@ -204,6 +204,26 @@
     If you have a legacy free Toshiba laptop (such as the Libretto L1
     series), say Y.

+config ACPI_PCC
+ tristate "Panasonic Laptop Extras"
+ depends on X86
+ depends on ACPI_INTERPRETER
+ default m
+ ---help---
+ This driver adds support for access to certain system settings
+ on panasonic Let's Note laptops.
+
+ On these machines, all hotkey is handled through the ACPI.
+ This driver is required for access to controls not covered
+ by the general ACPI drivers, such as LCD brightness, video output,
+ etc.
+
+ More information about this driver will be available at
+ <http://www.da-cha.org/letsnote/>
+
+ If you have a panasonic lets note laptop (such as the CF-T2, Y2,
+ R2, W2, R3), say Y.
+
config ACPI_DEBUG
```

Linux-Kernel: [PATCH][ACPI] Panasonic Hotkey Driver

```
bool "Debug Statements"
depends on ACPI_INTERPRETER
diff -Nru a/drivers/acpi/Makefile b/drivers/acpi/Makefile
--- a/drivers/acpi/Makefile 2004-07-27 12:44:53 +09:00
+++ b/drivers/acpi/Makefile 2004-07-27 12:44:53 +09:00
@@ -47,4 +47,5 @@
obj-$(CONFIG_ACPI_NUMA) += numa.o
obj-$(CONFIG_ACPI_ASUS) += asus_acpi.o
obj-$(CONFIG_ACPI_TOSHIBA) += toshiba_acpi.o
+obj-$(CONFIG_ACPI_PCC) += pcc_acpi.o
obj-$(CONFIG_ACPI_BUS) += scan.o
diff -Nru a/drivers/acpi/pcc_acpi.c b/drivers/acpi/pcc_acpi.c
--- /dev/null Wed Dec 31 16:00:00 196900
+++ b/drivers/acpi/pcc_acpi.c 2004-07-27 12:44:53 +09:00
@@ -0,0 +1,518 @@
+/*
+ * Panasonic HotKey control Extra driver
+ * (C) 2004 Hiroshi Miura <miura@da-cha.org>
+ * (C) 2004 NTT DATA Intellilink Co. http://www.intellilink.co.jp/
+ * All Rights Reserved
+ *
+ * This program is free software; you can redistribute it and/or modify
+ * it under the terms of the GNU General Public License version 2 as
+ * published by the 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., 59 Temple Place, Suite 330, Boston, MA 02111-1307 USA
+ *
+ * The devolpment page for this driver will be located at
+ * http://www.da-cha.org/
+ *
+ * -----
+ *
+ * ChangeLog:
+ * Jul.25, 2004 Hiroshi Miura <miura@da-cha.org>
+ * - v0.4 first post version
+ * - add debug function to retriive SIFR
+ *
+ * Jul.24, 2004 Hiroshi Miura <miura@da-cha.org>
+ * - v0.3 get proper status of hotkey
+ *
+ * Jul.22, 2004 Hiroshi Miura <miura@da-cha.org>
+ * - v0.2 add HotKey handler
+ *
+ * Jul.17, 2004 Hiroshi Miura <miura@da-cha.org>
+ * - v0.1 based on acpi video driver
```

## Linux-Kernel: [PATCH][ACPI] Panasonic Hotkey Driver

```
+ *
+ * TODO
+ * everything all
+ *
+ */
+
+#define ACPI_PCC_VERSION "0.4"
+#define PROC_INTERFACE_VERSION 2
+
+#include <linux/kernel.h>
+#include <linux/module.h>
+#include <linux/init.h>
+#include <linux/types.h>
+#include <linux/proc_fs.h>
+#include <asm/uaccess.h>
+
+#include <acpi/acpi_bus.h>
+#include <acpi/acpi_drivers.h>
+
+MODULE_AUTHOR("Hiroshi Miura");
+MODULE_DESCRIPTION("ACPI HotKey driver for lets note");
+MODULE_LICENSE("GPL");
+
+#define LOGPREFIX "acpi_pcc: "
+
+/******
+ * Define ACPI PATHs
+ *****/
+/* crt/lcd hot key definitions */
+#define DEVICE_NAME_VGA "\\_SB_.PCI0.GRFX"
+#define DEVICE_NAME_CRT "CRT1"
+#define DEVICE_NAME_LCD "LCD1"
+#define METHOD_CHGD "\\_SB_.CHGD"
+
+/* Lets note hotkeys definitions */
+#define DEVICE_NAME_HKEY "\\_SB_.HKEY"
+#define METHOD_HKEY_QUERY "HINF"
+
+/* ACPI BIOS inside use only? */
+#define METHOD_HKEY_RESET "HRES"
+#define METHOD_HKEY_SAVE "HSAV"
+#define METHOD_HKEY_HIND "HIND"
+
+/* event read/write functions */
+#define METHOD_HKEY_SQTY "SQTY"
+#define METHOD_HKEY_SINF "SINF"
+#define METHOD_HKEY_SSET "SSET"
+
+/* device(HKEY) definitions */
+#define HKEY_HID "MAT0019"
```

## Linux-Kernel: [PATCH][ACPI] Panasonic Hotkey Driver

```
+ #define HKEY_NOTIFY 0x80
+
+ /*****
+ *
+ * definitions for /proc/ interface
+ *
+ *****/
+ #define PROC_PCC "pcc"
+
+ #define ACPI_HOTKEY_DRIVER_NAME "PCC HotKey Driver"
+ #define ACPI_HOTKEY_DEVICE_NAME "HotKey"
+ #define ACPI_HOTKEY_CLASS "HKEY"
+
+ static int acpi_hotkey_add (struct acpi_device *device);
+ static int acpi_hotkey_remove (struct acpi_device *device, int type);
+
+ static struct acpi_driver acpi_hotkey_driver = {
+ .name = ACPI_HOTKEY_DRIVER_NAME,
+ .class = ACPI_HOTKEY_CLASS,
+ .ids = HKEY_HID,
+ .ops = {
+ .add = acpi_hotkey_add,
+ .remove = acpi_hotkey_remove,
+ },
+ };
+
+ struct acpi_hotkey {
+ acpi_handle handle;
+ struct acpi_device *device;
+ unsigned long status;
+ };
+
+
+ /*
+ * utility functions
+ */
+ static __inline__ void
+ _set_bit(u32* word, u32 mask, int value)
+ {
+ *word = (*word & ~mask) | (mask * value);
+ }
+
+ /* acpi interface wrappers
+ */
+ static int
+ is_valid_acpi_path(const char* methodName)
+ {
+ acpi_handle handle;
+ acpi_status status;
+
+ status = acpi_get_handle(0, (char*)methodName, &handle);
```

```

+ return !ACPI_FAILURE(status);
+}
+
+#if 0
+static int
+write_acpi_int(const char* methodName, int val)
+{
+ struct acpi_object_list params;
+ union acpi_object in_objs[1];
+ acpi_status status;
+
+ params.count = sizeof(in_objs)/sizeof(in_objs[0]);
+ params.pointer = in_objs;
+ in_objs[0].type = ACPI_TYPE_INTEGER;
+ in_objs[0].integer.value = val;
+
+ status = acpi_evaluate_object(0, (char*)methodName, &params, 0);
+ return (status == AE_OK);
+}
+#endif
+
+static int
+read_acpi_int(acpi_handle handle, const char* methodName, int* pVal)
+{
+ struct acpi_buffer results;
+ union acpi_object out_objs[1];
+ acpi_status status;
+
+ results.length = sizeof(out_objs);
+ results.pointer = out_objs;
+
+ status = acpi_evaluate_object(handle, (char*)methodName, 0, &results);
+ if (ACPI_FAILURE(status)) {
+ printk(KERN_INFO "acpi evaluate error on %s\n", methodName);
+ return (-EFAULT);
+ }
+
+ if (out_objs[0].type == ACPI_TYPE_INTEGER) {
+ *pVal = out_objs[0].integer.value;
+ } else {
+ printk(KERN_INFO "return value is not int\n");
+ status = AE_ERROR;
+ }
+
+ return (status == AE_OK);
+}
+
+static struct proc_dir_entry* acpi_pcc_dir;
+
+typedef struct _ProcItem
+{

```

```

+ const char* name;
+ char* (*read_func)(char*);
+ unsigned long (*write_func)(const char*, unsigned long);
+} ProcItem;
+
+
+/* register utils for proc handler */
+static int
+dispatch_read(char* page, char** start, off_t off, int count, int* eof,
+ ProcItem* item)
+{
+ char* p = page;
+ int len;
+
+ if (off == 0)
+ p = item->read_func(p);
+
+ /* ISSUE: I don't understand this code */
+ len = (p - page);
+ if (len <= off+count) *eof = 1;
+ *start = page + off;
+ len -= off;
+ if (len>count) len = count;
+ if (len<0) len = 0;
+ return len;
+}
+
+static int
+dispatch_write(struct file* file, __user const char* buffer,
+ unsigned long count, ProcItem* item)
+{
+ int result;
+ char* tmp_buffer;
+
+ /* Arg buffer points to userspace memory, which can't be accessed
+ * directly. Since we're making a copy, zero-terminate the
+ * destination so that sscanf can be used on it safely.
+ */
+ tmp_buffer = kmalloc(count + 1, GFP_KERNEL);
+ if (copy_from_user(tmp_buffer, buffer, count)) {
+ result = -EFAULT;
+ }
+ else {
+ tmp_buffer[count] = 0;
+ result = item->write_func(tmp_buffer, count);
+ }
+ kfree(tmp_buffer);
+ return result;
+}
+
+/*

```

```

+ * proc file handlers
+ */
+#ifdef DEBUG_PCC_VGA
+static unsigned long
+write_chgd(const char* buffer, unsigned long count)
+{
+ int value;
+ acpi_status status;
+
+ if (sscanf(buffer, "%i", &value) == 1 && value >= 0 && value < 2) {
+ if (value == 0)
+ /* do nothing */
+ status = AE_OK;
+ else {
+ status = acpi_evaluate_object(0, METHOD_CHGD, 0, 0);
+ }
+ if (ACPI_FAILURE(status)) {
+ printk(KERN_INFO LOGPREFIX "fail evaluate CHGD()\n");
+ return -EFAULT;
+ }
+ }
+ return count;
+
+ }
+
+static char*
+read_nothing(char* p)
+{
+ /* nothing to do*/
+ return p;
+ }
+#endif
+
+static char*
+read_hkey_status(char* p)
+{
+ acpi_status status;
+ struct acpi_buffer buffer = {ACPI_ALLOCATE_BUFFER, NULL};
+ union acpi_object *hkey = NULL;
+ int i, num_sifr;
+
+ if (!read_acpi_int(NULL, DEVICE_NAME_HKEY "." METHOD_HKEY_SQTY, &num_sifr)){
+ printk(KERN_INFO LOGPREFIX "evaluation error HKEY.SQTY\n");
+ return p;
+ }
+
+ status = acpi_evaluate_object(NULL, DEVICE_NAME_HKEY "." METHOD_HKEY_SINF, 0, &buffer);
+ if (ACPI_FAILURE(status)) {
+ printk(KERN_INFO LOGPREFIX "evaluation error HEKY.SINF\n");
+ return p;
+ }
+ }

```

## Linux-Kernel: [PATCH][ACPI] Panasonic Hotkey Driver

```

+ hkey = (union acpi_object *) buffer.pointer;
+ if (!hkey || (hkey->type != ACPI_TYPE_PACKAGE)) {
+ printk(KERN_INFO LOGPREFIX "Invalid HKEY.SINF\n");
+ goto end;
+ }
+
+ if (num_sifr != hkey->package.count) {
+ printk(KERN_INFO LOGPREFIX "SQTY is not equal to SINF length?\n");
+ goto end;
+ }
+
+ for (i = 0; i < hkey->package.count; i++) {
+ union acpi_object *element = &(hkey->package.elements[i]);
+ if (likely(element->type == ACPI_TYPE_INTEGER)) {
+ p += sprintf(p, "0x%02x,\n", (unsigned int)element->integer.value);
+ } else
+ printk(KERN_INFO LOGPREFIX "Invalid HKEY.SINF value\n");
+ }
+end:
+ acpi_os_free(buffer.pointer);
+ return p;
+}
+
+static char*
+read_version(char* p)
+{
+ p += sprintf(p, "%s version %s\n", ACPI_HOTKEY_DRIVER_NAME, ACPI_PCC_VERSION);
+ p += sprintf(p, "proc_interface version %d\n", PROC_INTERFACE_VERSION);
+ return p;
+}
+
+/* hotkey driver */
+static int
+acpi_hotkey_get_key(struct acpi_hotkey *hotkey)
+{
+ int result;
+ int status;
+
+ status = read_acpi_int(hotkey->handle, METHOD_HKEY_QUERY, &result);
+ if (!status) {
+ printk(KERN_INFO LOGPREFIX "error getting hotkey status\n");
+ } else
+ hotkey->status = result;
+
+ return (status);
+}
+
+void
+acpi_hotkey_notify(acpi_handle handle, u32 event, void *data)

```

```

+{
+ struct acpi_hotkey *hotkey = (struct acpi_hotkey *) data;
+
+ if (!hotkey || !hotkey->device)
+ return;
+
+ switch(event) {
+ case HKEY_NOTIFY:
+ if (acpi_hotkey_get_key(hotkey))
+ acpi_bus_generate_event(hotkey->device, event, hotkey->status);
+ break;
+ default:
+ /* nothing to do */
+ break;
+ }
+
+ return;
+}
+
+static int
+acpi_hotkey_add (struct acpi_device *device)
+{
+ int result = 0;
+ acpi_status status = AE_OK;
+ struct acpi_hotkey *hotkey = NULL;
+
+ if (!device)
+ return (-EINVAL);
+
+ hotkey = kmalloc(sizeof(struct acpi_hotkey), GFP_KERNEL);
+ if (!hotkey)
+ return (-ENOMEM);
+
+ memset(hotkey, 0, sizeof(struct acpi_hotkey));
+
+ hotkey->device = device;
+ hotkey->handle = device->handle;
+ acpi_driver_data(device) = hotkey;
+ strcpy(acpi_device_name(device), ACPI_HOTKEY_DEVICE_NAME);
+ strcpy(acpi_device_class(device), ACPI_HOTKEY_CLASS);
+
+ status = acpi_install_notify_handler (
+ hotkey->handle,
+ ACPI_DEVICE_NOTIFY,
+ acpi_hotkey_notify,
+ hotkey);
+ if (ACPI_FAILURE(status))
+ result = -ENODEV;
+
+ if (result)
+ kfree(hotkey);

```

```

+
+ return (result);
+}
+
+static int
+acpi_hotkey_remove(struct acpi_device *device, int type)
+{
+ acpi_status status = 0;
+ struct acpi_hotkey *hotkey = NULL;
+
+ if (!device || !acpi_driver_data(device))
+ return(-EINVAL);
+
+ hotkey = acpi_driver_data(device);
+ status = acpi_remove_notify_handler(hotkey->handle,
+ ACPI_DEVICE_NOTIFY, acpi_hotkey_notify);
+ if (ACPI_FAILURE(status))
+ printk(KERN_INFO LOGPREFIX "Error removing notify handler\n");
+
+ kfree(hotkey);
+
+ return(0);
+}
+
+/*
+ * proc and module init
+ */
+
+ ProcItem pcc_proc_items[] =
+ {
+ #ifdef DEBUG_PCC_VGA
+ { "chgd" , read_nothing , write_chgd},
+ #endif
+ { "hkey_status", read_hkey_status, NULL},
+ { "version", read_version , NULL},
+ { NULL , NULL , NULL},
+ };
+
+static acpi_status __init
+add_device(ProcItem *proc_items, struct proc_dir_entry* proc_entry)
+{
+ struct proc_dir_entry* proc;
+ ProcItem* item;
+
+ for (item = proc_items; item->name; ++item)
+ {
+ proc = create_proc_read_entry(item->name,
+ S_IFREG | S_IRUGO | S_IWUSR,
+ proc_entry, (read_proc_t*)dispatch_read, item);
+ if (proc)
+ proc->owner = THIS_MODULE;

```

```

+ if (proc && item->write_func)
+ proc->write_proc = (write_proc_t*)dispatch_write;
+ }
+
+ return(AE_OK);
+}
+
+
+static int __init
+pcc_proc_init(void)
+{
+ acpi_status status = AE_OK;
+
+ if (unlikely(!(acpi_pcc_dir = proc_mkdir(PROC_PCC, acpi_root_dir))))
+ return -ENODEV;
+
+ acpi_pcc_dir->owner = THIS_MODULE;
+ status = add_device(pcc_proc_items, acpi_pcc_dir);
+ if (ACPI_FAILURE(status)){
+ remove_proc_entry(PROC_PCC, acpi_root_dir);
+ return -ENODEV;
+ }
+
+ return (status == AE_OK);
+}
+
+static acpi_status __exit
+remove_device(ProcItem *proc_items, struct proc_dir_entry* proc_entry)
+{
+ ProcItem* item;
+
+ for (item = proc_items; item->name; ++item)
+ remove_proc_entry(item->name, proc_entry);
+ return(AE_OK);
+}
+
+
+
+/* init funcs. */
+static int __init
+acpi_pcc_init(void)
+{
+ acpi_status result = AE_OK;
+
+ if (acpi_disabled)
+ return -ENODEV;
+
+ /* simple device detection: look forI method */
+ if (!(is_valid_acpi_path(METHOD_CHGD)))
+ return -ENODEV;
+
+

```

## Linux-Kernel: [PATCH][ACPI] Panasonic Hotkey Driver

```
+ result = acpi_bus_register_driver(&acpi_hotkey_driver);
+ if (ACPI_FAILURE(result))
+ printk(KERN_INFO LOGPREFIX "Error registering hotkey driver\n");
+
+ printk(KERN_INFO LOGPREFIX "ACPI PCC HotKey driver version %s\n", ACPI_PCC_VERSION);
+
+ return (pcc_proc_init());
+
+}
+
+static void __exit
+acpi_pcc_exit(void)
+{
+ if (acpi_pcc_dir) {
+ remove_device(pcc_proc_items, acpi_pcc_dir);
+ remove_proc_entry(PROC_PCC, acpi_root_dir);
+ }
+ acpi_bus_unregister_driver(&acpi_hotkey_driver);
+ return;
+}
+
+module_init(acpi_pcc_init);
+module_exit(acpi_pcc_exit);
```

--

Hiroshi Miura --- <http://www.da-cha.org/> --- miura@da-cha.org  
NTTDATA Corp. OpenSource Software Center. --- miurahr@nttdata.co.jp  
NTTDATA Intellilink Corp. OpenSource Engineering Dev. -- miurahr@intellilink.co.jp  
Key fingerprint = 9117 9407 5684 FBF1 4063 15B4 401D D077 04AB 8617

-

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/>