

## [PATCH] 2.6.0-test4 – Watchdog patches

**Source:** <http://linux.derkeiler.com/Mailing-Lists/Kernel/2003-08/8350.html>

---

**From:** Wim Van Sebroeck ([wim\\_at\\_iguana.be](mailto:wim_at_iguana.be))

**Date:** 08/31/03

Date: Sun, 31 Aug 2003 22:52:36 +0200  
To: [torvalds@osdl.org](mailto:torvalds@osdl.org)

Hi Linus,

please do a

bk pull <http://linux-watchdog.bkbits.net/linux-2.5-watchdog>

This will update the following files:

```
drivers/char/watchdog/Kconfig | 12
drivers/char/watchdog/Makefile | 1
drivers/char/watchdog/acquirewdt.c | 230 ++++++++-----
drivers/char/watchdog/advantechwdt.c | 2
drivers/char/watchdog/alim1535_wdt.c | 465 ++++++++
drivers/char/watchdog/wafer5823wdt.c | 181 ++++++++
6 files changed, 754 insertions(+), 137 deletions(-)
```

through these ChangeSets:

<[wim@iguana.be](mailto:wim@iguana.be)> (03/08/30 1.1292)  
[WATCHDOG] advantechwdt.c – patch

small clean-up (add trivial comma)

<[wim@iguana.be](mailto:wim@iguana.be)> (03/08/30 1.1293)  
[WATCHDOG] wafer5823wdt.c – patch

general clean-up (comments, trailing spaces, ...)  
Added WATCHDOG\_NAME and PFX defines for easier printk's.  
clean-up printk's.

<[wim@iguana.be](mailto:wim@iguana.be)> (03/08/30 1.1294)  
[WATCHDOG] wafer5823wdt.c – patch2

fix possible wafwdt\_is\_open race  
make wdt\_stop and wdt\_start module params  
change wd\_margin to timeout and make it a module\_param  
make expect\_close the same system as in advantechwdt.c

Linux-Kernel: [PATCH] 2.6.0-test4 – Watchdog patches

clean-up ioctl handling  
added extra printk's to report what problem occurred  
add MODULE\_DESCRIPTION info

<wim@iguana.be> (03/08/31 1.1295)  
[WATCHDOG] wafer5823wdt.c – patch3

fix MODULE\_PARM\_DESC for timeout  
add WDIOC\_SETOPTIONS functionality

<wim@iguana.be> (03/08/31 1.1296)  
[WATCHDOG] acquirewdt.c – patch

clean-up of comments, trailing spaces, includes, ...  
removed unnecessary spinlocking  
added WATCHDOG\_NAME + PFX defines for easier printk's  
clean-up expect\_close / acq\_is\_open  
made wdt\_stop and wdt\_start a module\_param  
clean-up ioctl handling  
clean-up init and exit routines  
added MODULE\_AUTHOR + MODULE\_DESCRIPTION info

<wim@iguana.be> (03/08/31 1.1297)  
[WATCHDOG] alim1535\_wdt.c

Add "ALi M1535 PMU Watchdog Timer" driver

The ChangeSets can also be looked at on:

<http://linux-watchdog.bkbits.net:8080/linux-2.5-watchdog>

For completeness, I added the patches below.

Greetings,  
Wim.

```
=====
diff -Nru a/drivers/char/watchdog/advantechwdt.c b/drivers/char/watchdog/advantechwdt.c
--- a/drivers/char/watchdog/advantechwdt.c Sun Aug 31 22:46:08 2003
+++ b/drivers/char/watchdog/advantechwdt.c Sun Aug 31 22:46:08 2003
@@ -133,7 +133,7 @@
     static struct watchdog_info ident = {
         .options = WDIOF_KEEPALIVEPING | WDIOF_SETTIMEOUT | WDIOF_MAGICCLOSE,
         .firmware_version = 1,
-        .identity = "Advantech WDT"
+        .identity = "Advantech WDT",
     };

     switch (cmd) {
diff -Nru a/drivers/char/watchdog/wafer5823wdt.c b/drivers/char/watchdog/wafer5823wdt.c
--- a/drivers/char/watchdog/wafer5823wdt.c Sun Aug 31 22:46:28 2003
+++ b/drivers/char/watchdog/wafer5823wdt.c Sun Aug 31 22:46:28 2003
```

Linux-Kernel: [PATCH] 2.6.0-test4 – Watchdog patches

@@ -1,5 +1,5 @@

/\*

– \* ICP Wafer 5823 Single Board Computer WDT driver for Linux 2.4.x

+ \* ICP Wafer 5823 Single Board Computer WDT driver

\* [http://www.icpamerica.com/wafer\\_5823.php](http://www.icpamerica.com/wafer_5823.php)

\* May also work on other similar models

\*

@@ -17,10 +17,10 @@

\* modify it under the terms of the GNU General Public License

\* as published by the Free Software Foundation; either version

\* 2 of the License, or (at your option) any later version.

– \*

– \* Neither Alan Cox nor CymruNet Ltd. admit liability nor provide

– \* warranty for any of this software. This material is provided

– \* "AS-IS" and at no charge.

+ \*

+ \* Neither Alan Cox nor CymruNet Ltd. admit liability nor provide

+ \* warranty for any of this software. This material is provided

+ \* "AS-IS" and at no charge.

\*

\* (c) Copyright 1995 Alan Cox <alan@lxorguk.ukuu.org.uk>

\*

@@ -39,6 +39,10 @@

#include <asm/io.h>

#include <asm/uaccess.h>

+#define WATCHDOG\_NAME "Wafer 5823 WDT"

+#define PFX WATCHDOG\_NAME ": "

+#define WD\_TIMO 60 /\* 60 sec default timeout \*/

+

static unsigned long wafwdt\_is\_open;

static spinlock\_t wafwdt\_lock;

static int expect\_close = 0;

@@ -55,7 +59,6 @@

#define WDT\_START 0x443

#define WDT\_STOP 0x843

–#define WD\_TIMO 60 /\* 1 minute \*/

static int wd\_margin = WD\_TIMO;

#ifdef CONFIG\_WATCHDOG\_NOWAYOUT

@@ -124,7 +127,7 @@

static struct watchdog\_info ident = {

.options = WDIOF\_KEEPALIVEPING | WDIOF\_SETTIMEOUT | WDIOF\_MAGICCLOSE,

.firmware\_version = 1,

– .identity = "Wafer 5823 WDT"

+ .identity = "Wafer 5823 WDT",

};

int one=1;

@@ -174,10 +177,10 @@

Linux-Kernel: [PATCH] 2.6.0-test4 – Watchdog patches

```

wafwdt_close(struct inode *inode, struct file *file)
{
    clear_bit(0, &wafwdt_is_open);
- if (expect_close) {
+ if (expect_close) {
    wafwdt_stop();
    } else {
- printk(KERN_CRIT "WDT device closed unexpectedly. WDT will not stop!\n");
+ printk(KERN_CRIT PFX "WDT device closed unexpectedly. WDT will not stop!\n");
    }
    return 0;
}
@@ -201,6 +204,7 @@

static struct file_operations wafwdt_fops = {
    .owner = THIS_MODULE,
+ .llseek = no_llseek,
    .write = wafwdt_write,
    .ioctl = wafwdt_ioctl,
    .open = wafwdt_open,
@@ -210,23 +214,23 @@
static struct miscdevice wafwdt_miscdev = {
    .minor = WATCHDOG_MINOR,
    .name = "watchdog",
- .fops = &wafwdt_fops
+ .fops = &wafwdt_fops,
};

/*
 * The WDT needs to learn about soft shutdowns in order to
- * turn the timebomb registers off.
+ * turn the timebomb registers off.
 */

static struct notifier_block wafwdt_notifier = {
    .notifier_call = wafwdt_notify_sys,
    .next = NULL,
- .priority = 0
+ .priority = 0,
};

static int __init wafwdt_init(void)
{
- printk(KERN_INFO "WDT driver for Wafer 5823 single board computer initialising.\n");
+ printk(KERN_INFO PFX "WDT driver for Wafer 5823 single board computer initialising.\n");

    spin_lock_init(&wafwdt_lock);
    if(!request_region(WDT_STOP, 1, "Wafer 5823 WDT"))
diff -Nru a/drivers/char/watchdog/wafer5823wdt.c b/drivers/char/watchdog/wafer5823wdt.c
--- a/drivers/char/watchdog/wafer5823wdt.c Sun Aug 31 22:46:47 2003
+++ b/drivers/char/watchdog/wafer5823wdt.c Sun Aug 31 22:46:47 2003

```

## Linux-Kernel: [PATCH] 2.6.0-test4 – Watchdog patches

```
@@ -44,8 +44,8 @@
#define WD_TIMO 60 /* 60 sec default timeout */

static unsigned long wafwdt_is_open;
+static char expect_close;
static spinlock_t wafwdt_lock;
-static int expect_close = 0;

/*
 * You must set these – there is no sane way to probe for this board.
@@ -56,10 +56,17 @@
 * to restart it again.
 */

-#define WDT_START 0x443
-#define WDT_STOP 0x843
-
-static int wd_margin = WD_TIMO;
+static int wdt_stop = 0x843;
+module_param(wdt_stop, int, 0);
+MODULE_PARM_DESC(wdt_stop, "Wafer 5823 WDT 'stop' io port (default 0x843)");
+
+static int wdt_start = 0x443;
+module_param(wdt_start, int, 0);
+MODULE_PARM_DESC(wdt_start, "Wafer 5823 WDT 'start' io port (default 0x443)");
+
+static int timeout = WD_TIMO; /* in seconds */
+module_param(timeout, int, 0);
+MODULE_PARM_DESC(timeout, "Watchdog timeout in seconds. 1<= timeout <=255, default="
__MODULE_STRING(WATCHDOG_TIMEOUT) ".");

#ifdef CONFIG_WATCHDOG_NOWAYOUT
static int nowayout = 1;
@@ -73,24 +80,24 @@
static void wafwdt_ping(void)
{
    /* pat watchdog */
- spin_lock(&wafwdt_lock);
- inb_p(WDT_STOP);
- inb_p(WDT_START);
- spin_unlock(&wafwdt_lock);
+ spin_lock(&wafwdt_lock);
+ inb_p(wdt_stop);
+ inb_p(wdt_start);
+ spin_unlock(&wafwdt_lock);
}

static void wafwdt_start(void)
{
    /* start up watchdog */
- outb_p(wd_margin, WDT_START);
```

```

- inb_p(WDT_START);
+ outb_p(timeout, wdt_start);
+ inb_p(wdt_start);
}

static void
wafwdt_stop(void)
{
    /* stop watchdog */
- inb_p(WDT_STOP);
+ inb_p(wdt_stop);
}

static ssize_t wafwdt_write(struct file *file, const char *buf, size_t count, loff_t * ppos)
@@ -99,6 +106,7 @@
    if (ppos != &file->f_pos)
        return -ESPIPE;

+ /* See if we got the magic character 'V' and reload the timer */
    if (count) {
        if (!nowayout) {
            size_t i;
@@ -106,30 +114,30 @@
            /* In case it was set long ago */
            expect_close = 0;

+ /* scan to see whether or not we got the magic character */
            for (i = 0; i != count; i++) {
                char c;
                if (get_user(c, buf + i))
                    return -EFAULT;
                if (c == 'V')
- expect_close = 1;
+ expect_close = 42;
            }
        }
+ /* Well, anyhow someone wrote to us, we should return that favour */
        wafwdt_ping();
- return 1;
    }
- return 0;
+ return count;
}

static int wafwdt_ioctl(struct inode *inode, struct file *file, unsigned int cmd,
                      unsigned long arg)
{
- int new_margin;
+ int new_timeout;
    static struct watchdog_info ident = {
        .options = WDIOF_KEEPALIVEPING | WDIOF_SETTIMEOUT | WDIOF_MAGICCLOSE,

```

```

        .firmware_version = 1,
        .identity = "Wafer 5823 WDT",
    };
- int one=1;

    switch (cmd) {
    case WDIOG_GETSUPPORT:
@@ -139,25 +147,24 @@
        break;

    case WDIOG_GETSTATUS:
- if (copy_to_user((int *) arg, &one, sizeof (int)))
- return -EFAULT;
- break;
+ case WDIOG_GETBOOTSTATUS:
+ return put_user(0, (int *)arg);

    case WDIOG_KEEPLIVE:
        wafwdt_ping();
        break;

    case WDIOG_SETTIMEOUT:
- if (get_user(new_margin, (int *)arg))
+ if (get_user(new_timeout, (int *)arg))
        return -EFAULT;
- if ((new_margin < 1) || (new_margin > 255))
+ if ((new_timeout < 1) || (new_timeout > 255))
        return -EINVAL;
- wd_margin = new_margin;
+ timeout = new_timeout;
        wafwdt_stop();
        wafwdt_start();
        /* Fall */

    case WDIOG_GETTIMEOUT:
- return put_user(wd_margin, (int *)arg);
+ return put_user(timeout, (int *)arg);

    default:
        return -ENOTTY;
@@ -169,6 +176,10 @@
    {
        if (test_and_set_bit(0, &wafwdt_is_open))
            return -EBUSY;

+
+ /*
+ * Activate
+ */
        wafwdt_start();
        return 0;
    }
@@ -176,12 +187,14 @@

```

```

static int
wafwdt_close(struct inode *inode, struct file *file)
{
- clear_bit(0, &wafwdt_is_open);
- if (expect_close) {
+ if (expect_close == 42) {
    wafwdt_stop();
  } else {
    printk(KERN_CRIT PFX "WDT device closed unexpectedly. WDT will not stop!\n");
+ wafwdt_ping();
  }
+ clear_bit(0, &wafwdt_is_open);
+ expect_close = 0;
  return 0;
}

```

@@ -230,37 +243,77 @@

```

static int __init wafwdt_init(void)
{
- printk(KERN_INFO PFX "WDT driver for Wafer 5823 single board computer initialising.\n");
+ int ret;
+
+ printk(KERN_INFO "WDT driver for Wafer 5823 single board computer initialising.\n");

    spin_lock_init(&wafwdt_lock);
- if(!request_region(WDT_STOP, 1, "Wafer 5823 WDT"))
- goto error;
- if(!request_region(WDT_START, 1, "Wafer 5823 WDT"))
+
+ if (timeout < 1 || timeout > 63) {
+ timeout = WD_TIMO;
+ printk (KERN_INFO PFX "timeout value must be 1<=x<=255, using %d\n",
+ timeout);
+ }
+
+ if (wdt_stop != wdt_start) {
+ if(!request_region(wdt_stop, 1, "Wafer 5823 WDT")) {
+ printk (KERN_ERR PFX "I/O address 0x%04x already in use\n",
+ wdt_stop);
+ ret = -EIO;
+ goto error;
+ }
+ }
+
+ if(!request_region(wdt_start, 1, "Wafer 5823 WDT")) {
+ printk (KERN_ERR PFX "I/O address 0x%04x already in use\n",
+ wdt_start);
+ ret = -EIO;
+ goto error2;
- if(misc_register(&wafwdt_miscdev)<0)

```

```

+ }
+
+ ret = register_reboot_notifier(&wafwdt_notifier);
+ if (ret != 0) {
+ printk (KERN_ERR PFX "cannot register reboot notifier (err=%d)\n",
+ ret);
+     goto error3;
- register_reboot_notifier(&wafwdt_notifier);
- return 0;
+ }
+
+ ret = misc_register(&wafwdt_miscdev);
+ if (ret != 0) {
+ printk (KERN_ERR PFX "cannot register miscdev on minor=%d (err=%d)\n",
+ WATCHDOG_MINOR, ret);
+ goto error4;
+ }
+
+ printk (KERN_INFO PFX "initialized. timeout=%d sec (nowayout=%d)\n",
+ timeout, nowayout);
+
+ return ret;
+error4:
+ unregister_reboot_notifier(&wafwdt_notifier);
error3:
- release_region(WDT_START, 1);
+ release_region(wdt_start, 1);
error2:
- release_region(WDT_STOP, 1);
+ if (wdt_stop != wdt_start)
+ release_region(wdt_stop, 1);
error:
- return -ENODEV;
+ return ret;
}

static void __exit wafwdt_exit(void)
{
    misc_deregister(&wafwdt_miscdev);
    unregister_reboot_notifier(&wafwdt_notifier);
- release_region(WDT_STOP, 1);
- release_region(WDT_START, 1);
+ if(wdt_stop != wdt_start)
+ release_region(wdt_stop, 1);
+ release_region(wdt_start, 1);
}

module_init(wafwdt_init);
module_exit(wafwdt_exit);

MODULE_AUTHOR("Justin Cormack");

```

Linux-Kernel: [PATCH] 2.6.0-test4 – Watchdog patches

```
+MODULE_DESCRIPTION("ICP Wafer 5823 Single Board Computer WDT driver");
MODULE_LICENSE("GPL");

/* end of wafer5823wdt.c */
diff -Nru a/drivers/char/watchdog/wafer5823wdt.c b/drivers/char/watchdog/wafer5823wdt.c
--- a/drivers/char/watchdog/wafer5823wdt.c Sun Aug 31 22:47:06 2003
+++ b/drivers/char/watchdog/wafer5823wdt.c Sun Aug 31 22:47:06 2003
@@ -66,7 +66,7 @@

static int timeout = WD_TIMO; /* in seconds */
module_param(timeout, int, 0);
-MODULE_PARM_DESC(timeout, "Watchdog timeout in seconds. 1<= timeout <=255, default="
__MODULE_STRING(WATCHDOG_TIMEOUT) ".");
+MODULE_PARM_DESC(timeout, "Watchdog timeout in seconds. 1<= timeout <=255, default="
__MODULE_STRING(WD_TIMO) ".");

#ifdef CONFIG_WATCHDOG_NOWAYOUT
static int nowayout = 1;
@@ -165,6 +165,26 @@
    /* Fall */
    case WDIOC_GETTIMEOUT:
        return put_user(timeout, (int *)arg);
+
+ case WDIOC_SETOPTIONS:
+ {
+ int options, retval = -EINVAL;
+
+ if (get_user(options, (int *)arg))
+ return -EFAULT;
+
+ if (options & WDIOS_DISABLECARD) {
+ wafwdt_start();
+ retval = 0;
+ }
+
+ if (options & WDIOS_ENABLECARD) {
+ wafwdt_stop();
+ retval = 0;
+ }
+
+ return retval;
+ }

    default:
        return -ENOTTY;
diff -Nru a/drivers/char/watchdog/acquirewdt.c b/drivers/char/watchdog/acquirewdt.c
--- a/drivers/char/watchdog/acquirewdt.c Sun Aug 31 22:47:25 2003
+++ b/drivers/char/watchdog/acquirewdt.c Sun Aug 31 22:47:25 2003
@@ -1,5 +1,5 @@
/*
- * Acquire Single Board Computer Watchdog Timer driver for Linux 2.1.x
```

## Linux-Kernel: [PATCH] 2.6.0-test4 – Watchdog patches

```
+ * Acquire Single Board Computer Watchdog Timer driver
*
* Based on wdt.c. Original copyright messages:
*
@@ -10,10 +10,10 @@
* modify it under the terms of the GNU General Public License
* as published by the Free Software Foundation; either version
* 2 of the License, or (at your option) any later version.
- *
- * Neither Alan Cox nor CymruNet Ltd. admit liability nor provide
- * warranty for any of this software. This material is provided
- * "AS-IS" and at no charge.
+ *
+ * Neither Alan Cox nor CymruNet Ltd. admit liability nor provide
+ * warranty for any of this software. This material is provided
+ * "AS-IS" and at no charge.
*
* (c) Copyright 1995 Alan Cox <alan@redhat.com>
*
@@ -22,33 +22,39 @@
* Can't add timeout – driver doesn't allow changing value
*/

-#include <linux/config.h>
#include <linux/module.h>
#include <linux/moduleparam.h>
#include <linux/types.h>
#include <linux/miscdevice.h>
#include <linux/watchdog.h>
+#include <linux/fs.h>
#include <linux/ioport.h>
#include <linux/notifier.h>
-#include <linux/fs.h>
#include <linux/reboot.h>
#include <linux/init.h>
-#include <linux/spinlock.h>

#include <asm/io.h>
#include <asm/uaccess.h>
#include <asm/system.h>

-static int acq_is_open;
-static spinlock_t acq_lock;
-static int expect_close = 0;
+#define WATCHDOG_NAME "Acquire WDT"
+#define PFX WATCHDOG_NAME ": "
+#define WATCHDOG_TIMEOUT 0 /* ??? Is the timeout hardcoded to 1 minute ??? */
+
+static unsigned long acq_is_open;
+static char expect_close;
```

```

/*
 * You must set these – there is no sane way to probe for this board.
 */
-
-#define WDT_STOP 0x43
-#define WDT_START 0x443
+
+static int wdt_stop = 0x43;
+module_param(wdt_stop, int, 0);
+MODULE_PARM_DESC(wdt_stop, "Acquire WDT 'stop' io port (default 0x43)");
+
+static int wdt_start = 0x443;
+module_param(wdt_start, int, 0);
+MODULE_PARM_DESC(wdt_start, "Acquire WDT 'start' io port (default 0x443)");

#ifdef CONFIG_WATCHDOG_NOWAYOUT
static int nowayout = 1;
@@ -62,38 +68,52 @@
/*
 * Kernel methods.
 */
-
static void acq_ping(void)
{
    /* Write a watchdog value */
- inb_p(WDT_START);
+ inb_p(wdt_start);
}

+static void acq_stop(void)
+{
+ /* Turn the card off */
+ inb_p(wdt_stop);
+}
+
+ /* /dev/watchdog handling.
+ */
+
static ssize_t acq_write(struct file *file, const char *buf, size_t count, loff_t *ppos)
{
    /* Can't seek (pwrite) on this device */
    if (ppos != &file->f_pos)
        return -ESPIPE;

+ /* See if we got the magic character 'V' and reload the timer */
    if(count) {
        if (!nowayout) {
            size_t i;

```

## Linux-Kernel: [PATCH] 2.6.0-test4 – Watchdog patches

```

+ /* note: just in case someone wrote the magic character
+ * five months ago... */
        expect_close = 0;

+ /* scan to see wether or not we got the magic character */
        for (i = 0; i != count; i++) {
            char c;
            if (get_user(c, buf + i))
                return -EFAULT;
            if (c == 'V')
- expect_close = 1;
+ expect_close = 42;
        }
    }

+
+ /* Well, anyhow someone wrote to us, we should return that favour */
    acq_ping();
- return 1;
    }
- return 0;
+ return count;
}

static int acq_ioctl(struct inode *inode, struct file *file, unsigned int cmd,
@@ -103,65 +123,75 @@
    {
        .options = WDIOF_KEEPAALIVEPING | WDIOF_MAGICCLOSE,
        .firmware_version = 1,
- .identity = "Acquire WDT"
+ .identity = "Acquire WDT",
    };

-
+
    switch(cmd)
    {
        case WDIOCG_GETSUPPORT:
- if (copy_to_user((struct watchdog_info *)arg, &ident, sizeof(ident)))
- return -EFAULT;
- break;
-
+ return copy_to_user((struct watchdog_info *)arg, &ident, sizeof(ident)) ? -EFAULT : 0;
+
        case WDIOCG_GETSTATUS:
- if (copy_to_user((int *)arg, &acq_is_open, sizeof(int)))
- return -EFAULT;
- break;
+ case WDIOCG_GETBOOTSTATUS:
+ return put_user(0, (int *)arg);

        case WDIOCG_KEEPAALIVE:
            acq_ping();

```

```

- break;
+ return 0;
+
+ case WDIOCG_GETTIMEOUT:
+ return put_user(WATCHDOG_TIMEOUT, (int *)arg);
+
+ case WDIOCG_SETOPTIONS:
+ {
+ int options, retval = -EINVAL;
+
+ if (get_user(options, (int *)arg))
+ return -EFAULT;
+
+ if (options & WDIOCG_DISABLECARD)
+ {
+ acq_stop();
+ retval = 0;
+ }
+
+ if (options & WDIOCG_ENABLECARD)
+ {
+ acq_ping();
+ retval = 0;
+ }
+
+ return retval;
+ }

    default:
- return -ENOTTY;
+ return -ENOIOCTLCMD;
    }
- return 0;
}

static int acq_open(struct inode *inode, struct file *file)
{
- if ((minor(inode->i_rdev) == WATCHDOG_MINOR)) {
- spin_lock(&acq_lock);
- if(acq_is_open) {
- spin_unlock(&acq_lock);
- return -EBUSY;
- }
- if (nowayout)
- __module_get(THIS_MODULE);
+ if (test_and_set_bit(0, &acq_is_open))
+ return -EBUSY;

- /* Activate */
- acq_is_open=1;
- inb_p(WDT_START);

```

```

- spin_unlock(&acq_lock);
- return 0;
+ if (nowayout)
+ __module_get(THIS_MODULE);

- } else {
- return -ENODEV;
- }
+ /* Activate */
+ acq_ping();
+ return 0;
}

static int acq_close(struct inode *inode, struct file *file)
{
- if(minor(inode->i_rdev)==WATCHDOG_MINOR) {
- spin_lock(&acq_lock);
- if (expect_close)
- inb_p(WDT_STOP);
- else
- printk(KERN_CRIT "WDT closed unexpectedly. WDT will not stop!\n");
-
- acq_is_open=0;
- spin_unlock(&acq_lock);
+ if (expect_close == 42) {
+ acq_stop();
+ } else {
+ printk(KERN_CRIT PFX "Unexpected close, not stopping watchdog!\n");
+ acq_ping();
+ }
+ clear_bit(0, &acq_is_open);
+ expect_close = 0;
+ return 0;
}

@@ -172,20 +202,20 @@
static int acq_notify_sys(struct notifier_block *this, unsigned long code,
void *unused)
{
- if(code==SYS_DOWN || code==SYS_HALT)
- /* Turn the card off */
- inb_p(WDT_STOP);
-
+ if(code==SYS_DOWN || code==SYS_HALT) {
+ /* Turn the WDT off */
+ acq_stop();
+ }
+ return NOTIFY_DONE;
}
-
+

```

```

/*
 * Kernel Interfaces
 */
-
-
+
static struct file_operations acq_fops = {
    .owner = THIS_MODULE,
+ .llseek = no_llseek,
    .write = acq_write,
    .ioctl = acq_ioctl,
    .open = acq_open,
@@ -196,52 +226,84 @@
{
    .minor = WATCHDOG_MINOR,
    .name = "watchdog",
- .fops = &acq_fops
+ .fops = &acq_fops,
};

-
/*
 * The WDT card needs to learn about soft shutdowns in order to
- * turn the timebomb registers off.
+ * turn the timebomb registers off.
 */
-
+
static struct notifier_block acq_notifier =
{
    .notifier_call = acq_notify_sys,
    .next = NULL,
- .priority = 0
+ .priority = 0,
};

static int __init acq_init(void)
{
+ int ret;
+
    printk(KERN_INFO "WDT driver for Acquire single board computer initialising.\n");

- spin_lock_init(&acq_lock);
- if (misc_register(&acq_miscdev))
- return -ENODEV;
- if (!request_region(WDT_STOP, 1, "Acquire WDT")) {
- misc_deregister(&acq_miscdev);
- return -EIO;
- }
- if (!request_region(WDT_START, 1, "Acquire WDT")) {
- release_region(WDT_STOP, 1);

```

```

- misc_deregister(&acq_miscdev);
- return -EIO;
+ if (wdt_stop != wdt_start) {
+ if (!request_region(wdt_stop, 1, WATCHDOG_NAME)) {
+ printk (KERN_ERR PFX "I/O address 0x%04x already in use\n",
+ wdt_stop);
+ ret = -EIO;
+ goto out;
+ }
    }

- register_reboot_notifier(&acq_notifier);
- return 0;
+ if (!request_region(wdt_start, 1, WATCHDOG_NAME)) {
+ printk (KERN_ERR PFX "I/O address 0x%04x already in use\n",
+ wdt_start);
+ ret = -EIO;
+ goto unreg_stop;
+ }
+
+ ret = register_reboot_notifier(&acq_notifier);
+ if (ret != 0) {
+ printk (KERN_ERR PFX "cannot register reboot notifier (err=%d)\n",
+ ret);
+ goto unreg_regions;
+ }
+
+ ret = misc_register(&acq_miscdev);
+ if (ret != 0) {
+ printk (KERN_ERR PFX "cannot register miscdev on minor=%d (err=%d)\n",
+ WATCHDOG_MINOR, ret);
+ goto unreg_reboot;
+ }
+
+ printk (KERN_INFO PFX "initialized. (nowayout=%d)\n",
+ nowayout);
+
+out:
+ return ret;
+unreg_reboot:
+ unregister_reboot_notifier(&acq_notifier);
+unreg_regions:
+ release_region(wdt_start, 1);
+unreg_stop:
+ if (wdt_stop != wdt_start)
+ release_region(wdt_stop, 1);
+ goto out;
    }
-
+
static void __exit acq_exit(void)

```

```

{
    misc_deregister(&acq_miscdev);
    unregister_reboot_notifier(&acq_notifier);
- release_region(WDT_STOP,1);
- release_region(WDT_START,1);
+ if(wdt_stop != wdt_start)
+ release_region(wdt_stop,1);
+ release_region(wdt_start,1);
}

module_init(acq_init);
module_exit(acq_exit);

MODULE_LICENSE("GPL");
+MODULE_AUTHOR("Unkown");
+MODULE_DESCRIPTION("Acquire Single Board Computer Watchdog Timer driver");
diff -Nru a/drivers/char/watchdog/Kconfig b/drivers/char/watchdog/Kconfig
--- a/drivers/char/watchdog/Kconfig Sun Aug 31 22:47:44 2003
+++ b/drivers/char/watchdog/Kconfig Sun Aug 31 22:47:44 2003
@@ -346,6 +346,18 @@
     module, say M here and read <file:Documentation/modules.txt>. Most
     people will say N.

+config ALIM1535_WDT
+ tristate "ALi M1535 PMU Watchdog Timer"
+ depends on WATCHDOG
+ ---help---
+ This is the driver for the hardware watchdog on the ALi M1535 PMU.
+
+ This driver is also available as a module (= code which can be
+ inserted in and removed from the running kernel whenever you want).
+ The module is called alim1535_wdt. If you want to compile it as a
+ module, say M here and read <file:Documentation/modules.txt>. Most
+ people will say N.
+
config SC1200_WDT
    tristate "National Semiconductor PC87307/PC97307 (ala SC1200) Watchdog"
    depends on WATCHDOG
diff -Nru a/drivers/char/watchdog/Makefile b/drivers/char/watchdog/Makefile
--- a/drivers/char/watchdog/Makefile Sun Aug 31 22:47:44 2003
+++ b/drivers/char/watchdog/Makefile Sun Aug 31 22:47:44 2003
@@ -27,6 +27,7 @@
obj-$(CONFIG_W83877F_WDT) += w83877f_wdt.o
obj-$(CONFIG_SC520_WDT) += sc520_wdt.o
obj-$(CONFIG_ALIM7101_WDT) += alim7101_wdt.o
+obj-$(CONFIG_ALIM1535_WDT) += alim1535_wdt.o
obj-$(CONFIG_SC1200_WDT) += sc1200wdt.o
obj-$(CONFIG_WAFER_WDT) += wafer5823wdt.o
obj-$(CONFIG_CPU5_WDT) += cpu5wdt.o
diff -Nru a/drivers/char/watchdog/alim1535_wdt.c b/drivers/char/watchdog/alim1535_wdt.c
--- /dev/null Wed Dec 31 16:00:00 1969

```

## Linux-Kernel: [PATCH] 2.6.0-test4 – Watchdog patches

```
+++ b/drivers/char/watchdog/alim1535_wdt.c Sun Aug 31 22:47:44 2003
@@ -0,0 +1,465 @@
+/*
+ * Watchdog for the 7101 PMU version found in the ALi M1535 chipsets
+ *
+ * This program is free software; you can redistribute it and/or
+ * modify it under the terms of the GNU General Public License
+ * as published by the Free Software Foundation; either version
+ * 2 of the License, or (at your option) any later version.
+ */
+
+#include <linux/module.h>
+#include <linux/moduleparam.h>
+#include <linux/types.h>
+#include <linux/miscdevice.h>
+#include <linux/watchdog.h>
+#include <linux/ioport.h>
+#include <linux/notifier.h>
+#include <linux/reboot.h>
+#include <linux/init.h>
+#include <linux/pci.h>
+
+#include <asm/uaccess.h>
+#include <asm/io.h>
+
+#define WATCHDOG_NAME "ALi_M1535"
+#define PFX WATCHDOG_NAME ": "
+#define WATCHDOG_TIMEOUT 60 /* 60 sec default timeout */
+
+/* internal variables */
+static unsigned long ali_is_open;
+static char ali_expect_release;
+static struct pci_dev *ali_pci;
+static u32 ali_timeout_bits; /* stores the computed timeout */
+static spinlock_t ali_lock; /* Guards the hardware */
+
+/* module parameters */
+static int timeout = WATCHDOG_TIMEOUT;
+module_param(timeout, int, 0);
+MODULE_PARM_DESC(timeout, "Watchdog timeout in seconds. (0<timeout<18000, default="
+__MODULE_STRING(WATCHDOG_TIMEOUT) ")");
+
+#ifdef CONFIG_WATCHDOG_NOWAYOUT
+static int nowayout = 1;
+#else
+static int nowayout = 0;
+#endif
+
+module_param(nowayout, int, 0);
+MODULE_PARM_DESC(nowayout, "Watchdog cannot be stopped once started
+(default=CONFIG_WATCHDOG_NOWAYOUT)");
```

```

+
+/*
+ * ali_start – start watchdog countdown
+ *
+ * Starts the timer running providing the timer has a counter
+ * configuration set.
+ */
+
+static void ali_start(void)
+{
+ u32 val;
+
+ spin_lock(&ali_lock);
+
+ pci_read_config_dword(ali_pci, 0xCC, &val);
+ val &= ~0x3F; /* Mask count */
+ val |= (1<<25) | ali_timeout_bits;
+ pci_write_config_dword(ali_pci, 0xCC, val);
+
+ spin_unlock(&ali_lock);
+}
+
+/*
+ * ali_stop – stop the timer countdown
+ *
+ * Stop the ALi watchdog countdown
+ */
+
+static void ali_stop(void)
+{
+ u32 val;
+
+ spin_lock(&ali_lock);
+
+ pci_read_config_dword(ali_pci, 0xCC, &val);
+ val &= ~0x3F; /* Mask count to zero (disabled) */
+ val &= ~(1<<25); /* and for safety mask the reset enable */
+ pci_write_config_dword(ali_pci, 0xCC, val);
+
+ spin_unlock(&ali_lock);
+}
+
+/*
+ * ali_keepalive – send a keepalive to the watchdog
+ *
+ * Send a keepalive to the timer (actually we restart the timer).
+ */
+
+static void ali_keepalive(void)
+{
+ ali_start();

```

```

+}
+
+/*
+ * ali_settimer – compute the timer reload value
+ * @t: time in seconds
+ *
+ * Computes the timeout values needed
+ */
+
+static int ali_settimer(int t)
+{
+ if(t < 0)
+ return -EINVAL;
+ else if(t < 60)
+ ali_timeout_bits = t|(1<<6);
+ else if(t < 3600)
+ ali_timeout_bits = (t/60)|(1<<7);
+ else if(t < 18000)
+ ali_timeout_bits = (t/300)|(1<<6)|(1<<7);
+ else return -EINVAL;
+
+ timeout = t;
+ return 0;
+}
+
+/*
+ * /dev/watchdog handling
+ */
+
+/*
+ * ali_write – writes to ALi watchdog
+ * @file: file from VFS
+ * @data: user address of data
+ * @len: length of data
+ * @ppos: pointer to the file offset
+ *
+ * Handle a write to the ALi watchdog. Writing to the file pings
+ * the watchdog and resets it. Writing the magic 'V' sequence allows
+ * the next close to turn off the watchdog.
+ */
+
+static ssize_t ali_write(struct file *file, const char *data,
+ size_t len, loff_t * ppos)
+{
+ /* Can't seek (pwrite) on this device */
+ if (ppos != &file->f_pos)
+ return -ESPIPE;
+
+ /* See if we got the magic character 'V' and reload the timer */
+ if (len) {
+ if (!nowayout) {

```

```

+ size_t i;
+
+ /* note: just in case someone wrote the magic character
+ * five months ago... */
+ ali_expect_release = 0;
+
+ /* scan to see wether or not we got the magic character */
+ for (i = 0; i != len; i++) {
+ char c;
+ if(get_user(c, data+i))
+ return -EFAULT;
+ if (c == 'V')
+ ali_expect_release = 42;
+ }
+ }
+
+ /* someone wrote to us, we should reload the timer */
+ ali_start();
+ }
+ return len;
+}
+
+/*
+ * ali_ioctl – handle watchdog ioctls
+ * @inode: VFS inode
+ * @file: VFS file pointer
+ * @cmd: ioctl number
+ * @arg: arguments to the ioctl
+ *
+ * Handle the watchdog ioctls supported by the ALi driver. Really
+ * we want an extension to enable irq ack monitoring and the like
+ */
+
+static int ali_ioctl(struct inode *inode, struct file *file,
+ unsigned int cmd, unsigned long arg)
+{
+ static struct watchdog_info ident = {
+ .options = WDIOF_KEEPAALIVEPING |
+ WDIOF_SETTIMEOUT |
+ WDIOF_MAGICCLOSE,
+ .firmware_version = 0,
+ .identity = "ALi M1535 WatchDog Timer",
+ };
+
+ switch (cmd) {
+ case WDIOG_GETSUPPORT:
+ return copy_to_user((struct watchdog_info *) arg, &ident,
+ sizeof (ident)) ? -EFAULT : 0;
+
+ case WDIOG_GETSTATUS:
+ case WDIOG_GETBOOTSTATUS:

```

```

+ return put_user(0, (int *) arg);
+
+ case WDIOC_KEEPALIVE:
+ ali_keepalive();
+ return 0;
+
+ case WDIOC_SETOPTIONS:
+ {
+ int new_options, retval = -EINVAL;
+
+ if (get_user (new_options, (int *) arg))
+ return -EFAULT;
+
+ if (new_options & WDIOS_DISABLECARD) {
+ ali_stop();
+ retval = 0;
+ }
+
+ if (new_options & WDIOS_ENABLECARD) {
+ ali_start();
+ retval = 0;
+ }
+
+ return retval;
+ }
+
+ case WDIOC_SETTIMEOUT:
+ {
+ int new_timeout;
+
+ if (get_user(new_timeout, (int *) arg))
+ return -EFAULT;
+
+ if (ali_settimer(new_timeout))
+ return -EINVAL;
+
+ ali_keepalive();
+ /* Fall */
+ }
+
+ case WDIOC_GETTIMEOUT:
+ return put_user(timeout, (int *)arg);
+
+ default:
+ return -ENOIOCTLCMD;
+ }
+}
+
+/*
+ * ali_open – handle open of ali watchdog
+ * @inode: inode from VFS

```

```

+ * @file: file from VFS
+ *
+ * Open the ALi watchdog device. Ensure only one person opens it
+ * at a time. Also start the watchdog running.
+ */
+
+static int ali_open(struct inode *inode, struct file *file)
+{
+ /* /dev/watchdog can only be opened once */
+ if (test_and_set_bit(0, &ali_is_open))
+ return -EBUSY;
+
+ /* Activate */
+ ali_start();
+ return 0;
+}
+
+/*
+ * ali_release – close an ALi watchdog
+ * @inode: inode from VFS
+ * @file: file from VFS
+ *
+ * Close the ALi watchdog device. Actual shutdown of the timer
+ * only occurs if the magic sequence has been set.
+ */
+
+static int ali_release(struct inode *inode, struct file *file)
+{
+ /*
+ * Shut off the timer.
+ */
+ if (ali_expect_release == 42) {
+ ali_stop();
+ } else {
+ printk(KERN_CRIT PFX "Unexpected close, not stopping watchdog!\n");
+ ali_keepalive();
+ }
+ clear_bit(0, &ali_is_open);
+ ali_expect_release = 0;
+ return 0;
+}
+
+/*
+ * ali_notify_sys – System down notifier
+ *
+ * Notifier for system down
+ */
+
+static int ali_notify_sys(struct notifier_block *this, unsigned long code, void *unused)
+{

```

```

+ if (code==SYS_DOWN || code==SYS_HALT) {
+ /* Turn the WDT off */
+ ali_stop();
+ }
+
+ return NOTIFY_DONE;
+}
+
+/*
+ * Data for PCI driver interface
+ *
+ * This data only exists for exporting the supported
+ * PCI ids via MODULE_DEVICE_TABLE. We do not actually
+ * register a pci_driver, because someone else might one day
+ * want to register another driver on the same PCI id.
+ */
+
+static struct pci_device_id ali_pci_tbl[] __initdata = {
+ { PCI_VENDOR_ID_AL, 1535, PCI_ANY_ID, PCI_ANY_ID,},
+ { 0, },
+};
+MODULE_DEVICE_TABLE(pci, ali_pci_tbl);
+
+/*
+ * ali_find_watchdog – find a 1535 and 7101
+ *
+ * Scans the PCI hardware for a 1535 series bridge and matching 7101
+ * watchdog device. This may be overtight but it is better to be safe
+ */
+
+static int __init ali_find_watchdog(void)
+{
+ struct pci_dev *pdev;
+ u32 wdog;
+
+ /* Check for a 1535 series bridge */
+ pdev = pci_find_device(PCI_VENDOR_ID_AL, 0x1535, NULL);
+ if(pdev == NULL)
+ return -ENODEV;
+
+ /* Check for the a 7101 PMU */
+ pdev = pci_find_device(PCI_VENDOR_ID_AL, 0x7101, NULL);
+ if(pdev == NULL)
+ return -ENODEV;
+
+ if(pci_enable_device(pdev))
+ return -EIO;
+
+ ali_pci = pdev;
+
+ /*

```

```

+ * Initialize the timer bits
+ */
+ pci_read_config_dword(pdev, 0xCC, &wdog);
+
+ wdog &= ~0x3F; /* Timer bits */
+ wdog &= ~((1<<27)|(1<<26)|(1<<25)|(1<<24)); /* Issued events */
+ wdog &= ~((1<<16)|(1<<13)|(1<<12)|(1<<11)|(1<<10)|(1<<9)); /* No monitor bits */
+
+ pci_write_config_dword(pdev, 0xCC, wdog);
+
+ return 0;
+}
+
+/*
+ * Kernel Interfaces
+ */
+
+static struct file_operations ali_fops = {
+ .owner = THIS_MODULE,
+ .llseek = no_llseek,
+ .write = ali_write,
+ .ioctl = ali_ioctl,
+ .open = ali_open,
+ .release = ali_release,
+};
+
+static struct miscdevice ali_miscdev = {
+ .minor = WATCHDOG_MINOR,
+ .name = "watchdog",
+ .fops = &ali_fops,
+};
+
+static struct notifier_block ali_notifier = {
+ .notifier_call = ali_notify_sys,
+ .next = NULL,
+ .priority = 0,
+};
+
+/*
+ * watchdog_init – module initialiser
+ *
+ * Scan for a suitable watchdog and if so initialize it. Return an error
+ * if we cannot, the error causes the module to unload
+ */
+
+static int __init watchdog_init(void)
+{
+ int ret;
+
+ spin_lock_init(&ali_lock);
+

```

Linux-Kernel: [PATCH] 2.6.0-test4 – Watchdog patches

```

+ /* Check whether or not the hardware watchdog is there */
+ if (ali_find_watchdog() != 0) {
+ return -ENODEV;
+ }
+
+ /* Check that the timeout value is within its range ; if not reset to the default */
+ if (timeout < 1 || timeout >= 18000) {
+ timeout = WATCHDOG_TIMEOUT;
+ printk(KERN_INFO PFX "timeout value must be 0<timeout<18000, using %d\n",
+ timeout);
+ }
+
+ /* Calculate the watchdog's timeout */
+ ali_settimer(timeout);
+
+ ret = misc_register(&ali_miscdev);
+ if (ret != 0) {
+ printk(KERN_ERR PFX "cannot register miscdev on minor=%d (err=%d)\n",
+ WATCHDOG_MINOR, ret);
+ goto out;
+ }
+
+ ret = register_reboot_notifier(&ali_notifier);
+ if (ret != 0) {
+ printk(KERN_ERR PFX "cannot register reboot notifier (err=%d)\n",
+ ret);
+ goto unreg_miscdev;
+ }
+
+ printk(KERN_INFO PFX "initialized. timeout=%d sec (nowayout=%d)\n",
+ timeout, nowayout);
+
+out:
+ return ret;
+unreg_miscdev:
+ misc_deregister(&ali_miscdev);
+ goto out;
+}
+
+/*
+ * watchdog_exit – module de-initialiser
+ *
+ * Called while unloading a successfully installed watchdog module.
+ */
+
+static void __exit watchdog_exit(void)
+{
+ /* Stop the timer before we leave */
+ ali_stop();
+
+ /* Deregister */

```

Linux-Kernel: [PATCH] 2.6.0-test4 – Watchdog patches

```
+ unregister_reboot_notifier(&ali_notifier);
+ misc_deregister(&ali_miscdev);
+}
+
+module_init(watchdog_init);
+module_exit(watchdog_exit);
+
+MODULE_AUTHOR("Alan Cox");
+MODULE_DESCRIPTION("ALi M1535 PMU Watchdog Timer driver");
+MODULE_LICENSE("GPL");
```

–

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/>