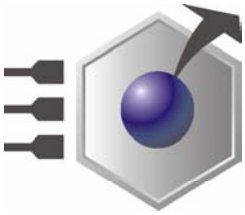
EmbeddedDNA®



EuroTecH

An0052

CPU-1420; Watchdog Linux Driver for CPU-1420

Rev. 1.0

May 2005

ABOUT THIS MANUAL

This application note contains information about the use of the Watchdog functionality in the CPU-1420 under the Linux operating System.

Via J. Linussio 1
33020 AMARO (UD)
ITALY
Phone: +39 0433 485 411
Fax: +39 0433 485 499

Web: http://www.eurotech.it
e-mail: mailto:sales@eurotech.it

NOTICE

Although all the information contained herein has been carefully verified, Eurotech S.p.A. assumes no responsibility for errors that might appear in this document, or for damage to property or persons resulting from an improper use of this manual and of the related software. Eurotech S.p.A. reserves the right to change the contents and form of this document, as well as the features and specifications of its products at any time, without notice.

Conventions

The following table lists conventions used throughout this guide.

Icon	Notice Type	Description
	**Information note**	Important features or instructions
	**Warning**	Information to alert you to potential damage to a program, system or device or potential personal injury

Mode of the register:
R/W: Read and write register.
RO: Read only register.
W: Meaning of the register when written.
R: Meaning of the register when read.

Name ranges:
A name followed by a range in brackets, for example Name[0:2], represent a range of logically related entities.

Hex Number:
Hexadecimal numbers are represented with an “h” suffix. (For example 11Ch)

(This page is intentionally left blank.)

Contents

Conventions 3

Contents 5

Chapter 1 Using the Watchdog drivers' functionalities with the CPU-1420 under Linux O.S. 7

Where you can find the drivers: 7

Installing the CPU-1420 Watchdog Linux driver 8

Testing the CPU-1420 watchdog driver functionality 8

Source code and Documentation 9

Eurotech watchdog Linux test source code 9

Eurotech watchdog Linux source code 10

Related Documents 19

Where to find us 19

(This page is intentionally left blank.)

Chapter 1 Using the Watchdog drivers' functionalities with the CPU-1420 under Linux O.S.

This brief application note describes the Linux Kernel support for the watchdog management with the CPU-1420.

The watch dog is a part of the on–board SUPER I/O device *SMSC FDC 37B782*

The Super I/0 watchdog allows the management of time-outs in seconds or minutes (depending on the programming of the Super I/O).

This application note doesn't report information on how to write program code to control the watchdog, for this information refer to the CPU manuals or Linux documentation.

Where you can find the drivers:

The driver is released with all official kernels starting from release 2.4.18. Please refer to the following Internet link to download the updated kernel release: www.kernel.org

Installing the CPU-1420 Watchdog Linux driver

After obtaining the kernel sources you have to compile the Kernel including the Eurotech CPU-1420 Watchdog timer support. For further information on how to compile the Kernel please refer to "The Linux Kernel HOWTO" www.tldp.org

Starting a shell:

```
bash# cd /usr/src/linux-2.4.18/
bash# make menuconfig
```

After starting the graphical configuration interface, from the Main Menu you have to enter into the Character devices menu and select the Watchdog Cards option; a list of cards will be displayed, you have to select the Eurotech CPU-1220/1410 Watchdog Timer (NEW) setting the flag as you prefer:

```
<M>     as module
<*>     compiled into the kernel architecture
```

Save the Configuration to an Alternative File and exit from the menu configuration program.
Then you have to proceed with the compilation of the kernel and the modules, for this purpose you can refer to the "The Linux Kernel HOWTO" documentation.

In this application not we assume that you have made your kernel compiling the driver as a module.

Testing the CPU-1420 watchdog driver functionality

If you compile the driver as a module you can load it with the following command:

```
bash# modprobe eurotechwdt
```

Listing the modules loaded with the following command you will find:

```
bash# lsmod

Module Size Used by Not tainted
eurotechwdt 3065 0 (unused)
usbcore 59308 1
8139too 14376 2
mii 2272 0 [8139too]
crc32 2880 0 [8139too]
ide-scsi 9328 0
```

Please be careful during the loading of the module that the Watchdog is initialized with proper values to prevent erroneous reset situations. Looking at the kernel sources of the driver you will find the proper value used to initialize the watchdog is by default 60 seconds.

Source code and Documentation

This section provides additional information about the source code of the Linux Eurotech watchdog driver and a brief test program to verify the functionality of the driver.

Eurotech watchdog Linux test source code

The following example watchdog source code is derived from the /usr/src/linux/Documentation/watchdog.txt file included with the kernel sources:

test_wdt.c

```
#include <stdio.h>
#include <unistd.h>
#include <fcntl.h>

int main(int argc, const char *argv[]) {

        int fd=open("/dev/watchdog",O_WRONLY);
        if (fd==1) {
            perror("watchdog");
            exit(1);
        }
        while (1) {

        write(fd, "\0", 1);
        sleep(10);
        }
}
```

You can compile the previous program as described below:

```
bash# gcc test_wdt.c –o test_wdt
```

You need to launch the program verifying that the eurotechwdt module is loaded.

```
bash# ./test_wdt
```

The program reloads the watchdog register contents continuously if all is working fine. To test the watchdog capability you can stop the program pressing CTRL + C finding the following message:

```
Eurwdt: unexpected Close, not stopping watchdog!
```

After the watchdog timeout (by default this is set to 60 seconds) the system will reset.

Eurotech watchdog Linux source code

Browsing the kernel source Linux directory (/usr/src/linux-2.4.18/) you can find the driver sources.
Below is the Eurotech Watchdog Linux driver:

Eurotechwdt.c

```
/*
 *              Eurotech CPU-1220/1410 on board WDT driver for Linux 2.4.x
 *
 *              (c) Copyright 2001 Ascensit <support@ascensit.com>
 *              (c) Copyright 2001 Rodolfo Giometti <giometti@ascensit.com>
 *              (c) Copyright 2002 Rob Radez <rob@osinvestor.com>
 *
 *              Based on wdt.c.
 *              Original copyright messages:
 *
 *      (c) Copyright 1996-1997 Alan Cox <alan@redhat.com>, All Rights Reserved.
 *                              http://www.redhat.com
 *
 *      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.
 *
 *      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>*
 */
```

```
/* Changelog:
 *
 * 2002/04/25 - Rob Radez
 *              clean up #includes
 *              clean up locking
 *              make __setup param unique
 *              proper options in watchdog_info
 *              add WDIOC_GETSTATUS and WDIOC_SETOPTIONS ioctls
 *              add expect_close support
 *
 * 2001 - Rodolfo Giometti
 *              Initial release
 *
 * 2002.05.30 - Joel Becker <joel.becker@oracle.com>
 *              Added Matt Domsch's nowayout module option.
 */

#include <linux/config.h>
#include <linux/module.h>
#include <linux/types.h>
#include <linux/errno.h>
#include <linux/kernel.h>
#include <linux/miscdevice.h>
#include <linux/watchdog.h>
#include <linux/ioport.h>
#include <linux/fcntl.h>
#include <asm/io.h>
#include <asm/uaccess.h>
#include <asm/system.h>
#include <linux/notifier.h>
#include <linux/reboot.h>
#include <linux/init.h>

static unsigned long eurwdt_is_open;
static int eurwdt_timeout;
static char eur_expect_close;

/*
 *      You must set these - there is no sane way to probe for this board.
 *      You can use wdt=x,y to set these now.
 */

static int io = 0x3f0;
static int irq = 10;
static char *ev = "int";

#define WDT_TIMEOUT                     60              /* 1 minute */

#ifdef CONFIG_WATCHDOG_NOWAYOUT
static int nowayout = 1;
#else
static int nowayout = 0;
#endif

MODULE_PARM(nowayout,"i");
MODULE_PARM_DESC(nowayout, "Watchdog cannot be stopped once started
                (default=CONFIG_WATCHDOG_NOWAYOUT)");

/*
```

```
 * Some symbolic names
 */

#define WDT_CTRL_REG                    0x30
#define WDT_OUTPIN_CFG      0xe2
#define WDT_EVENT_INT       0x00
#define WDT_EVENT_REBOOT            0x08
#define WDT_UNIT_SEL                    0xf1
#define WDT_UNIT_SECS       0x80
#define WDT_TIMEOUT_VAL     0xf2
#define WDT_TIMER_CFG       0xf3


#ifndef MODULE

/**
 *      eurwdt_setup:
 *      @str: command line string
 *
 *      Setup options. The board isn't really probe-able so we have to
 *      get the user to tell us the configuration. Sane people build it
 *      modular but the others come here.
 */

static int __init eurwdt_setup(char *str)
{
   int ints[4];

   str = get_options (str, ARRAY_SIZE(ints), ints);

   if (ints[0] > 0) {
      io = ints[1];
      if (ints[0] > 1)
         irq = ints[2];
   }

   return 1;
}

__setup("eurwdt=", eurwdt_setup);

#endif /* !MODULE */

MODULE_PARM(io, "i");
MODULE_PARM_DESC(io, "Eurotech WDT io port (default=0x3f0)");
MODULE_PARM(irq, "i");
MODULE_PARM_DESC(irq, "Eurotech WDT irq (default=10)");
MODULE_PARM(ev, "s");
MODULE_PARM_DESC(ev, "Eurotech WDT event type (default is `int')");


/*
 *      Programming support
 */

static inline void eurwdt_write_reg(u8 index, u8 data)
{
                outb(index, io);
                outb(data, io+1);
```

```
}

static inline void eurwdt_lock_chip(void)
{
        outb(0xaa, io);
}

static inline void eurwdt_unlock_chip(void)
{
        outb(0x55, io);
        eurwdt_write_reg(0x07, 0x08);   /* set the logical device */
}

static inline void eurwdt_set_timeout(int timeout)
{
        eurwdt_write_reg(WDT_TIMEOUT_VAL, (u8) timeout);
}

static inline void eurwdt_disable_timer(void)
{
        eurwdt_set_timeout(0);
}

static void eurwdt_activate_timer(void)
{
        eurwdt_disable_timer();
        eurwdt_write_reg(WDT_CTRL_REG, 0x01);      /* activate the WDT */
        eurwdt_write_reg(WDT_OUTPIN_CFG, !strcmp("int", ev) ?
                WDT_EVENT_INT : WDT_EVENT_REBOOT);
        /* Setting interrupt line */
        if (irq == 2 || irq > 15 || irq < 0) {
            printk(KERN_ERR ": invalid irq number\n");
            irq = 0;   /* if invalid we disable interrupt */
        }
        if (irq == 0)
            printk(KERN_INFO ": interrupt disabled\n");
        eurwdt_write_reg(WDT_TIMER_CFG, irq<<4);

        eurwdt_write_reg(WDT_UNIT_SEL, WDT_UNIT_SECS);   /* we use seconds */
        eurwdt_set_timeout(0);                          /* the default timeout */
}


/*
 *      Kernel methods.
 */

void eurwdt_interrupt(int irq, void *dev_id, struct pt_regs *regs)
{
        printk(KERN_CRIT "timeout WDT timeout\n");

#ifdef ONLY_TESTING
        printk(KERN_CRIT "Would Reboot.\n");
#else
        printk(KERN_CRIT "Initiating system reboot.\n");
        machine_restart(NULL);
#endif
}
```

```
/**
 *      eurwdt_ping:
 *
 *      Reload counter one with the watchdog timeout.
 */

static void eurwdt_ping(void)
{
                /* Write the watchdog default value */
                eurwdt_set_timeout(eurwdt_timeout);
}

/**
 *      eurwdt_write:
 *      @file: file handle to the watchdog
 *      @buf: buffer to write (unused as data does not matter here
 *      @count: count of bytes
 *      @ppos: pointer to the position to write. No seeks allowed
 *
 *      A write to a watchdog device is defined as a keepalive signal. Any
 *      write of data will do, as we we don't define content meaning.
 */

static ssize_t eurwdt_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;

                if (count) {
                    if (!nowayout) {
                                size_t i;

                                eur_expect_close = 0;

                                for (i = 0; i != count; i++) {
                                        char c;
                                        if(get_user(c, buf+i))
                                                return -EFAULT;
                                        if (c == 'V')
                                                eur_expect_close = 42;
                                }
                    }
                    eurwdt_ping();   /* the default timeout */
                }
                return count;
}

/**
 *      eurwdt_ioctl:
 *      @inode: inode of the device
 *      @file: file handle to the device
 *      @cmd: watchdog command
 *      @arg: argument pointer
 *
 *      The watchdog API defines a common set of functions for all watchdogs
 *      according to their available features.
 */
```

```
static int eurwdt_ioctl(struct inode *inode, struct file *file,
        unsigned int cmd, unsigned long arg)
{
                static struct watchdog_info ident = {
                    options             : WDIOF_KEEPALIVEPING | WDIOF_SETTIMEOUT |
                WDIOF_MAGICCLOSE,
                    firmware_version        : 0,
                    identity            : "WDT Eurotech CPU-1220/1410"
                };
                int time;

                switch(cmd) {
                    default:
                            return -ENOTTY;
                    case WDIOC_GETSUPPORT:
                            return copy_to_user((struct watchdog_info *)arg, &ident, sizeof(ident)) ? -EFAULT :
                0;
                    case WDIOC_GETSTATUS:
                    case WDIOC_GETBOOTSTATUS:
                            return put_user(0, (int *) arg);
                    case WDIOC_KEEPALIVE:
                            eurwdt_ping();
                            return 0;
                    case WDIOC_SETTIMEOUT:
                            if (copy_from_user(&time, (int *) arg, sizeof(int)))
                                    return -EFAULT;
                            /* Sanity check */
                            if (time < 0 || time > 255)
                                    return -EINVAL;
                            eurwdt_timeout = time;
                            eurwdt_set_timeout(time);
                            /* Fall */
                    case WDIOC_GETTIMEOUT:
                            return put_user(eurwdt_timeout, (int *)arg);
                    case WDIOC_SETOPTIONS:
                    {
                            int options, retval = -EINVAL;

                            if (get_user(options, (int *)arg))
                                    return -EFAULT;
                            if (options & WDIOS_DISABLECARD) {
                                    eurwdt_disable_timer();
                                    retval = 0;
                            }

                            if (options & WDIOS_ENABLECARD) {
                                    eurwdt_activate_timer();
                                    eurwdt_ping();
                                    retval = 0;
                            }
                            return retval;
                    }
                }
}

/**
 *      eurwdt_open:
 *      @inode: inode of device
```

```
 *      @file: file handle to device
 *
 *      The misc device has been opened. The watchdog device is single
 *      open and on opening we load the counter.
 */

static int eurwdt_open(struct inode *inode, struct file *file)
{
                if (test_and_set_bit(0, &eurwdt_is_open))
                    return -EBUSY;

                eurwdt_timeout = WDT_TIMEOUT;   /* initial timeout */

                /* Activate the WDT */
                eurwdt_activate_timer();

                return 0;
}

/**
 *      eurwdt_release:
 *      @inode: inode to board
 *      @file: file handle to board
 *
 *      The watchdog has a configurable API. There is a religious dispute
 *      between people who want their watchdog to be able to shut down and
 *      those who want to be sure if the watchdog manager dies the machine
 *      reboots. In the former case we disable the counters, in the latter
 *      case you have to open it again very soon.
 */

static int eurwdt_release(struct inode *inode, struct file *file)
{
                if (eur_expect_close == 42) {
                    eurwdt_disable_timer();
                } else {
                    printk(KERN_CRIT "eurwdt: Unexpected close, not stopping watchdog!\n");
                    eurwdt_ping();
                }
                clear_bit(0, &eurwdt_is_open);
                eur_expect_close = 0;
                return 0;
}

/**
 *      eurwdt_notify_sys:
 *      @this: our notifier block
 *      @code: the event being reported
 *      @unused: unused
 *
 *      Our notifier is called on system shutdowns. We want to turn the card
 *      off at reboot otherwise the machine will reboot again during memory
 *      test or worse yet during the following fsck. This would suck, in fact
 *      trust me - if it happens it does suck.
 */

static int eurwdt_notify_sys(struct notifier_block *this, unsigned long code,
        void *unused)
{
```

```
                if (code == SYS_DOWN || code == SYS_HALT) {
                    /* Turn the card off */
                    eurwdt_disable_timer();
                }
                return NOTIFY_DONE;
}

/*
 *      Kernel Interfaces
 */


static struct file_operations eurwdt_fops = {
        owner:          THIS_MODULE,
        llseek:         no_llseek,
        write:          eurwdt_write,
        ioctl:          eurwdt_ioctl,
        open:           eurwdt_open,
        release:        eurwdt_release,
};

static struct miscdevice eurwdt_miscdev =
{
        minor:      WATCHDOG_MINOR,
        name:       "watchdog",
        fops:       &eurwdt_fops,
};

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

static struct notifier_block eurwdt_notifier =
{
        eurwdt_notify_sys,
        NULL,
        0
};

/**
 *      cleanup_module:
 *
 *      Unload the watchdog. You cannot do this with any file handles open.
 *      If your watchdog is set to continue ticking on close and you unload
 *      it, well it keeps ticking. We won't get the interrupt but the board
 *      will not touch PC memory so all is fine. You just have to load a new
 *      module in 60 seconds or reboot.
 */

static void __exit eurwdt_exit(void)
{
                eurwdt_lock_chip();
                misc_deregister(&eurwdt_miscdev);
                unregister_reboot_notifier(&eurwdt_notifier);
                release_region(io, 2);
                free_irq(irq, NULL);
}
```

```
/**
 *      eurwdt_init:
 *
 *      Set up the WDT watchdog board. After grabbing the resources
 *      we require we need also to unlock the device.
 *      The open() function will actually kick the board off.
 */

static int __init eurwdt_init(void)
{
                int ret;

                ret = misc_register(&eurwdt_miscdev);
                if (ret) {
                    printk(KERN_ERR "eurwdt: can't misc_register on minor=%d\n", WATCHDOG_MINOR);
                    goto out;
                }
                ret = request_irq(irq, eurwdt_interrupt, SA_INTERRUPT, "eurwdt", NULL);
                if(ret) {
                    printk(KERN_ERR "eurwdt: IRQ %d is not free.\n", irq);
                    goto outmisc;
                }

                if (!request_region(io, 2, "eurwdt")) {
                    printk(KERN_ERR "eurwdt: IO %X is not free.\n", io);
                    ret = -EBUSY;
                    goto outirq;
                }

                ret = register_reboot_notifier(&eurwdt_notifier);
                if (ret) {
                    printk(KERN_ERR "eurwdt: can't register reboot notifier (err=%d)\n", ret);
                    goto outreg;
                }
                eurwdt_unlock_chip();

                ret = 0;
                printk(KERN_INFO "Eurotech WDT driver 0.01 at %X (Interrupt %d)"
                " - timeout event: %s\n",  io, irq, (!strcmp("int", ev) ? "int" : "reboot"));
                return ret;
outreg:
                release_region(io, 2);
outirq:
                free_irq(irq, NULL);
outmisc:
                misc_deregister(&eurwdt_miscdev);
out:
                return ret;
}

module_init(eurwdt_init);
module_exit(eurwdt_exit);

MODULE_AUTHOR("Rodolfo Giometti");
MODULE_DESCRIPTION("Driver for Eurotech CPU-1220/1410 on board watchdog");
MODULE_LICENSE("GPL");
EXPORT_NO_SYMBOLS;
```

Related Documents

For more information please refer to the specific CPU user manual. http://www.eurotech.it
SMSC FDC37B782 SuperIO data sheet www.smsc.com
Linux kernel source Documentation www.kernel.org

Where to find us

Eurotech S.p.A.
Via Jacopo Linussio, 1 - 33020 Amaro (UD) ITALY
Tel. +39 0433 486258 - Fax +39 0433 486263

welcome@eurotech.it

http://www.eurotech.it
ftp://ftp.eurotech.it

Technical & Sales Assistance

If you have a technical question, please contact the Eurotech Customer Support Service

techsupp@eurotech.it

Old and new versions of manuals, application notes, patches, drivers and BIOS can be found at:

ftp://ftp.eurotech.it/

If you have a sales question, please contact your local Eurotech Sales Representative or the Regional Sales Office for your area.

Additional and latest information is available at Eurotech website, located at:

http://www.eurotech.it