WDT Device Driver Package
2.6.38 Kernel-Based Linux

1 Introduction

1.0 The WDT Device Driver Package consists of a Linux Device Driver, an application programming interface library, and example application programs

1.1 The driver was built and tested on a Linux version 2.6.32-34 generic based Ubuntu 10.04 Lucid Lynx distribution.

1.2 The driver is for use with WinSystems, Inc. Single Board Computers (SBC) that utilize our exclusive watchdog timer. The timer is accessed through two hard-coded I/O ports, at address 0x565 and 0x566. Port 0x566 is used to enable the watchdog timer, program the desired time-out count value, and reset the counter. Port 0x565 is used to program the counter for seconds or minutes.

The following WinSystems, Inc. Products incorporate the watchdog timer functional block:

SBC: EPX-C380-S, EPX-C380-D, EPX-855, EBC-C384-S, EBC-C384-D, LBC-LX800-G, EBC-855. PPM-LX800-G, PCM-VDX-1-256, PCM-VDX-2-512

1.3 This driver is provided on an 'as-is' basis and no warranty as to usability or fitness of purpose is inferred or claimed.

1.4 WinSystems, Inc. does not provide support for the modification of this driver. Customer application specific queries can be sent to: support@winsystems.com.

1.5 This work is provided under the terms of the GNU General Public License (GPL).

2 Installations and Build

2.0 The device driver and the sample applications are provided in source code form as a compressed zipped folder.

2.1 It will be necessary to become the root user to build and install the driver and device nodes.

2.2 The MAJOR number for this device is allocated dynamically. A static number can be assigned by editing the *wdt_init_major* variable at the beginning of *wdt.c*.

2.3 To create the device driver Loadable Kernel Module and the sample applications in a command shell, execute: ***make all***. The device driver Loadable Kernel Module ***wdt.ko*** is created and moved to the appropriate kernel driver directory. The file access permissions are set to allow access by all users and groups; they may be changed manually as desired. The sample program ***timer*** is also built.

make install will install the kernel driver to a kernel directory and create dependencies.
make uninstall will remove the kernel driver from the kernel directory.
make timer will create the timer sample program.
make clean will remove objects created by the build.
make spotless will forcibly remove all artifacts of the build.

2.4 The device driver can be loaded with the provided initialization script ***wdt_load*** or manually. In either case modprobe is used to install the driver. Executing:

modprobe wdt.ko

The *wdt_load* script can be added to the /etc/rc.local file to load the driver automatically on boot.

3 Driver Usage

3.0 The WDT configuration space is accessed utilizing byte-wide I/O access instructions. The device driver model that is most similar is the "Character" model. Block oriented, File I/O, and random access operations on these devices (read, write, and seek) are very inefficient, and may not always give the desired results. The driver was designed to use ioctl as its principal programming interface.

3.1 The file *wdtio.c* implements the ioctl interface and presents to the application a set of standard C functions that may be called directly from the application without any further need for dealing with, or understanding how to access the driver through ioctl. An application must include *wdt.h* and link to *wdt.o*.

3.2 Application Programming Interface

An object file containing the Application Programming Interface utilized by user level programs to access the Kernal Loadable Module device driver driven devices is created as part of the build procedure.

3.2.1 int read_wdt(void)

This function returns the current value of port 0x566 or -1 if the watchdog timer is inaccessible.

3.2.2 int write_wdt(int value)

This function writes the desired value to port 0x566. It returns the value 0 for a successful write or -1 if the watchdog timer is inaccessible. Writing a 0 to this address will disable the watchdog timer. Any other value enables the counter and starts the count-down timer. Subsequent writes reset the timer.

3.2.3 int set_wdt_sec(void)

This function writes the value 0x80 to port 0x565 which selects seconds as the unit for the count-down timer. It returns the value 0 for a successful write or -1 if the watchdog timer is inaccessible.

3.2.4 int set_wdt_min(void)

This function writes the value 0 to port 0x565 which selects minutes as the unit for the count-down timer. It returns the value 0 for a successful write or -1 if the watchdog timer is inaccessible.

4 Sample Programs

4.1 Timer

The *timer* sample application is a simple program that illustrates how to configure and control the watchdog timer. The syntax for the command line is:

./timer <time> <min/sec>

The *time* input selects the desired time count and the *min/sec* input selects the desired units.

Once you start the program, the count-down value is continuously displayed as it counts down. If any key is hit, it will reset the count-down value. If the 'Q' key is hit, the timer is disabled and the program is exited.

The Flash executable is built as part of the driver build, but may be built separately by: ***make timer***.

NOTE: In this sample program the -static switch is used with gcc. This causes all library functions used, to be contained within the executable. This creates a very large executable file. We use this technique because our ELS Linux distribution has a limited number of dynamic libraries present on the Disk-On-Chip and it's possible that the required library, especially in the case of pthread might not be present on the target system. It's certainly possible, and even recommended, that if a variety of programs are going to be run on an embedded system, to copy over all of the dynamic libraries necessary for their operation in which case static linking would not be required.