Watch Dog Timer (WDT) Driver Specification

Revision 1.0

February 2006

Table of Contents

1. Introduction

The Intel® 3100 Chipset Watch Dog Timer (WDT) is a kernel mode driver designed to run on Microsoft* Windows* OS (XP*, Server 2003*, Vista*, and XP* Embedded) platforms. When enabled, the WDT can generate an interrupt or reboot the system in the event of a system lockup.

This WDT supports a number of IOCTL commands. These commands enable the applications to access and control all the aspects of the WDT.

The WDT is located on the LPC bus. As such, it is not detected by the Plug-n-Play (PnP) manager. To install the driver, the user must manually go through the “Add Device” wizard in the Control Panel. Installation details are covered later in this document.

1.1. Scope

This document describes the methods to install and communicate with the WDT. A working knowledge with Microsoft* Windows* device drivers is assumed.

1.2. Reference Documents

- *Windows* XP 3790 DDK*
- *wdt.h*

2. Installing the WDT Driver

The WDT driver does not announce itself to the device manager.

To install the WDT driver:

Note: These instructions assume that you are in the default Microsoft XP control panel view.*

1. Open Control Panel and click “Printers and Other Hardware”.
2. In the “See Also” box at the top left hand corner, click “Add Hardware”.
3. Click “Next” to look for hardware.
4. Click “Yes, I have already connected the hardware” and then click “Next”.
5. Scroll to the bottom of the “Installed hardware” list and select “Add a new hardware device”.
6. Click “Install the hardware that I manually select from a list” and then Click “Next”.
7. Click “Show All Devices” and then Click “Next”.
8. Click “Have Disk…”, point to the location where the WDT driver is stored and click “OK”.
9. Click on “Intel® 3100 Chipset Watch Dog Timer” and install.

3. Interface with the Driver

The only supported driver communication method is through IOCTLs. The IOCTLs defined in this section allow a user-mode application to access all the capabilities of the WDT.

3.1. Symbolic Link

The driver creates a named device object that can be accessed through the symbolic link "**\\\\.\\SAWD1**". As such, the application can communicate with the driver by obtaining a handle from *CreateFile()* with the path, "**\\\\.\\SAWD1**".

```
/* Example:  Getting a handle to the WDT driver */

HANDLE handle_to_wdt;

handle_to_wdt = CreateFile(
                    "\\\\.\\SAWD1",
                    GENERIC_WRITE,
                    FILE_SHARE_WRITE,
                    NULL,
                    OPEN_EXISTING,
                    0,
                    NULL);
```

Figure 1: Getting a handle to the WDT driver

Once a handle is obtained successfully, the application uses the *DeviceIoControl()* function to communicate with the driver.

```
/* Example:  Sending an IOCTL to the driver.  Here we assume
 *           handle_to_wdt was obtained successfully
 */

BOOL     status;
int      return_length;

status = DeviceIoControl(
                handle_to_wdt,   /* handle to wdt    */
                IOCTL_ENABLE_WDT,/* IOCTL to send     */
                NULL,            /* no input buffer   */
                0,               /* input length = 0  */
                NULL,            /* no output buffer  */
                0,               /* output length = 0 */
                &return_length,  /* should be 0       */
                NULL );          /* wait until I/O completes */
```

Figure 2: Sending an IOCTL command to the driver

3.2. Supported IOCTLs

IOCTL_GET_CAPABILITIES

```
#define IOCTL_GET_CAPABILITIES                          \
            CTL_CODE( FILE_DEVICE_UNKNOWN, 0x802,       \
            METHOD_BUFFERED, FILE_ANY_ACCESS )
```

This IOCTL accepts a buffer of type PSAWD_CAPABILITY_OUT_BUFF, and fills in the `min` and `max` field, indicating the minimum and maximum downcounter value. The `Version` field will be set to 1.

```
typedef struct _SAWD_CAPABILITY_OUT_BUFF {

      unsigned long     Version;    /* version of interface used */
      unsigned long         min;    /* minimum value in msecs */
      unsigned long         max;    /* maximum value in msecs */

} SAWD_CAPABILITY_OUT_BUFF, *PSAWD_CAPABILITY_OUT_BUFF;
```

IOCTL_ENABLE_WDT

```
#define IOCTL_ENABLE_WDT                                \
            CTL_CODE( FILE_DEVICE_UNKNOWN, 0x0D00,      \
            METHOD_BUFFERED, FILE_ANY_ACCESS )
```

This IOCTL enables the Watch Dog Timer.

IOCTL_DISABLE_WDT

```
#define IOCTL_DISABLE_WDT                               \
            CTL_CODE( FILE_DEVICE_UNKNOWN, 0x0D01,      \
            METHOD_BUFFERED, FILE_ANY_ACCESS )
```

This IOCTL disables the Watch Dog Timer.

IOCTL_LOAD_COUNTER

```
#define IOCTL_LOAD_COUNTER                              \
            CTL_CODE( FILE_DEVICE_UNKNOWN, 0x0D02,      \
            METHOD_BUFFERED, FILE_ANY_ACCESS )
```

This IOCTL accepts a buffer of type PSAWD_CTRL, and loads the preload counter 1 and 2 using the data in the `Preload1` and `Preload2` fields.

```
typedef struct _SAWD_CTRL {

      HANDLE          UserEvent;      /* Handle to a Ring 3 user event */
      unsigned long   Version;        /* version of interface used     */
      unsigned long   Flags;          /* flags defined below */
      unsigned long   Preload1;       /* Preload register # 1 */
      unsigned long   Preload2;       /* Preload Register # 2 */
      unsigned long   DownCount;      /* Holds current down count */
```

```
        unsigned long  ConfigReg;      /* 16 bit Configuation register */
        unsigned short DeviceStatus;   /* store device status bits */
        unsigned short ReloadReg;      /* 16 bit reload register */
        unsigned short LockReg;        /* 8 bit Lock register */
        unsigned long  InterruptCount; /* # of times stage 1 INT occured */
        unsigned short IRQ;
        short          BusType;
        unsigned short BusNumber;
        unsigned short IrqOffset;

} SAWD_CTRL, *PSAWD_CTRL;
```

IOCTL_PING_THE_WDT

```
#define IOCTL_PING_THE_WDT                              \
            CTL_CODE( FILE_DEVICE_UNKNOWN, 0x0D02,      \
            METHOD_BUFFERED, FILE_ANY_ACCESS )
```

This IOCTL reloads the Watch Dog Timer's down-counter with the preload value to prevent a timeout. If the WDT is in stage 1 of the countdown, then preload value 1 will be loaded into the main down counter. If the WDT is in stage 2 of the countdown, then preload value 2 will be loaded into the main down counter.

IOCTL_SET_PRESCALAR

```
 #define  IOCTL_SET_PRESCALAR                           \
            CTL_CODE( FILE_DEVICE_UNKNOWN, 0x0D04,      \
            METHOD_BUFFERED, FILE_ANY_ACCESS )
```

This IOCTL sets how the 20-bit preload value will be loaded into the 35-bit down counter. The IOCTL accepts an unsigned long integer set to 0 or 1 in the input buffer. If the input is 0, then the 20-bit preload value will be counted down at 1KHz. If the input is 1, then the 20-bit preload value will be counted down at 1MHz.

IOCTL_READ_DOWN_COUNTER

```
#define IOCTL_READ_DOWN_COUNTER                         \
            CTL_CODE( FILE_DEVICE_UNKNOWN, 0x0D05,      \
            METHOD_BUFFERED, FILE_ANY_ACCESS )
```

This IOCTL returns the current value of the down counter (in an unsigned long integer).

IOCTL_SET_EXTERNAL_OUT

```
#define IOCTL_SET_EXTERNAL_OUT                          \
            CTL_CODE( FILE_DEVICE_UNKNOWN, 0x0D07,      \
            METHOD_BUFFERED, FILE_ANY_ACCESS )
```

This IOCTL enables or disables the external out pin when stage 2 time out occurs. This function accepts an unsigned long integer set to 0 (enable) or 1 (disable).

IOCTL_LOCK_DEVICE

```
#define IOCTL_LOCK_DEVICE                                   \
            CTL_CODE( FILE_DEVICE_UNKNOWN, 0x0D08,     \
            METHOD_BUFFERED, FILE_ANY_ACCESS )
```

This IOCTL locks the WDT device. Once locked, the state (WDT mode vs. free running mode, and Enabled vs. Disabled) of the Watch Dog Timer cannot be changed until the system resets.

IOCTL_ROUTE_INTERRUPT

```
#define IOCTL_ROUTE_INTERRUPT                               \
            CTL_CODE( FILE_DEVICE_UNKNOWN, 0x0D09,     \
            METHOD_BUFFERED, FILE_ANY_ACCESS )
```

This IOCTL accepts an unsigned long integer, sets to 0-3, and uses it to set the type of interrupt to trigger when stage 1 timer expires.

0	sets interrupt type to PCI
1	sets interrupt type to NMI
2	sets interrupt type to SMI
3	disables stage 1 interrupt

IOCTL_USER_HANDLE

```
#define IOCTL_USER_HANDLE                                   \
            CTL_CODE( FILE_DEVICE_UNKNOWN, 0x0D0D,     \
            METHOD_BUFFERED, FILE_ANY_ACCESS )
```

This IOCTL accepts a buffer of type `PSAWD_CTRL`, and creates an internal reference to the function specified by the `UserEvent` field. When a stage 2 timeout occurs, the driver will call this function.

IOCTL_TEST_CALLBACK

```
#define IOCTL_TEST_CALLBACK                                 \
            CTL_CODE( FILE_DEVICE_UNKNOWN, 0x0D0F,     \
            METHOD_BUFFERED, FILE_ANY_ACCESS )
```

This IOCTL is used for testing the event handling code and to verify that the application thread is released when the shared UserEvent is signaled by the WDT driver. This IOCTL exists mainly for application developers to debug their application.

IOCTL_GET_STATUS

```
#define IOCTL_GET_STATUS                                    \
            CTL_CODE( FILE_DEVICE_UNKNOWN, 0x0D13,     \
            METHOD_BUFFERED, FILE_ANY_ACCESS )
```

This IOCTL accepts a buffer of type `PSAWD_CTRL`, and fills the `ConfigReg` and `LockReg` fields with the current values of those two registers.

IOCTL_GET_STATUS

```
#define IOCTL_SET_MODE                                  \
      CTL_CODE( FILE_DEVICE_UNKNOWN, 0x0D15,            \
            METHOD_BUFFERED, FILE_ANY_ACCESS )
```

This IOCTL accepts an unsigned long integer, sets to 0 or 1, and uses it to set the WDT mode. If the input is 0, then the WDT is set to Watch Dog Timer mode. If the input is 1, then the WDT is set to free running mode.

IOCTL_GET_TIMEOUT_STATUS

```
#define IOCTL_GET_TIMEOUT_STATUS                        \
            CTL_CODE( FILE_DEVICE_UNKNOWN, 0x0D16,      \
            METHOD_BUFFERED, FILE_ANY_ACCESS )
```

This IOCTL accepts an unsigned long integer and uses it as both input and output.

As an input parameter, if the integer is set to 1, then the time out bit on the WDT device will be reset.

As an output parameter, the integer is set to 1 if both stage 1 and stage 2 timers have expired.

IOCTL_GET_STAGE_1_TIMEOUT_STATUS

```
#define IOCTL_GET_STAGE_1_TIMEOUT_STATUS                \
            CTL_CODE( FILE_DEVICE_UNKNOWN, 0x0D17,      \
            METHOD_BUFFERED, FILE_ANY_ACCESS )
```

This IOCTL accepts an unsigned long integer and uses it as both input and output.

As an input parameter, if the integer is set to 1, then the time out bit on the WDT device will be reset.

As an output parameter, the integer is set to 1 if the stage 1 timer has expired.