Watchdog Timer
Windows Device Driver Package

1 Introduction

1.1 Many of WinSystems full-featured, high-performance single board computers offer an integrated watchdog timer, which can be used to guard against software lockups. The watchdog timer can be configured from the CMOS setup utility or directly from software. The selection in the CMOS setting serves as the default timeout value as the processor boots. The provided driver allows user control of the watchdog timer under the Windows 7, 8, and 10 operating systems.

1.2 The WDT driver package is designed for and has been verified with 32-bit and 64-bit versions of Microsoft Windows 7, 8, and 10.

1.3 The driver is for use with the following WinSystems, Inc. Single Board Computers (SBC) that provide the integrated watchdog timer.

SBC: EPX-C380, EBC-C384, PXM-C388, PPM-C393, PPM-C407, EBC-C413, EPX-C414, PPM-LX800-G, PCM-VDX

2 Installation

2.1 Before loading the Windows operating system, verify that the Watchdog Timer settings are correct using the BIOS setup utility. This will only be applicable for SBC products with the integrated watchdog timer.

2.2 The driver, support files, and application are supplied in a zip file along with this document. The following files are included:

a. wdt.sys – Windows device driver
b. wdt.inf – Windows installation file
c. wdt.cat – Windows catalog file
d. WdfCoinstaller01011.dll – Windows co-installer
e. wdtDLL.dll – Windows DLL
f. wdtDLL.lib – Windows library file
g. wdtDLL.h – driver include file
h. WDTPetApp.cpp –Windows application source
i. WDTPetApp.exe –Windows application
j. vcredist_x86 or vcredist_x64 - Microsoft Visual C++ Redistributable

2.3 Installation is accomplished via the 'Add legacy hardware' selection found in the Action menu of the Windows Device Manager. Navigate to the drive and folder containing the driver files and select *wdt.inf*. The Windows installer will copy the *wdt.sys* and *wdtDLL.dll* files to the appropriate directory in the Windows installation.

2.4 In Device Manager, the wdt Device will appear under the Wdt Class. The only allowed hardware configuration for installation is I/O range 564-567H. The driver I/O range cannot be changed. A reboot may be required after resource selection is complete.

2.5 The watchdog timer is accessed through two I/O ports, at addresses 565H and 566H. Port 565H is used to program the counter for seconds or minutes. Port 566H is used to enable the watchdog timer, program the desired time-out count value, and reset the counter.

2.6 The included example program, *WdtPetApp.exe*, can be used to verify driver installation and functionality. Usage of this program is described later in this document.

3 Driver Overview and Architecture

3.1 The file *wdt.sys* is a Windows Driver Foundation kernel-mode (KMDF) driver which facilitates access to the underlying hardware.

3.2 The file *wdtDLL.dll* is a Windows Dynamic-Link Library which provides more user-friendly functions to access the device.

3.1 The driver utilizes the I/O Control (IOCTL) Request Framework to control the register set of the watchdog timer. Data is passed to and from the driver utilizing input and output buffers.

4 Driver Usage

4.1 The *wdtDLL.h* file included in the driver distribution contains the supported functions to be used by an application to communicate with the wdt Driver. This file is included in this document in Appendix A: wdtDLL.h.

4.2 An application calls the function *InitializeSession* to open the driver. This is required before any of the other functions can be called. The following example opens the WinSystems wdt driver. If a zero is returned, then the driver has been successfully initialized. Any other returned value indicates that an error has occurred and the device is unusable.

```
if (InitializeSession())
      printf("Error opening device.\n");
```

4.3 Once the driver is initialized, the other functions can be used to control the watchdog timer. Following is a description and sample code for each function. For all functions, if a zero is

returned, the function was successful. Otherwise there was an error and function did not complete.

4.3.1 int ReadTimerValue(unsigned int *readValue)

Reads the current 8-bit value of the watchdog timer register.

```
unsigned int read_data;

if (ReadTimerValue(&read_data))
       printf("Error reading timer.\n");
```

4.3.2 int WriteTimerValue(unsigned int writeValue)

Changes the current value of the watchdog timer with the parameter writeValue.

```
unsigned int write_data = 0xA5;

if (WriteTimerValue(write_data))
       printf("Error writing timer.\n");
```

4.3.3 int EnableTimer (unsigned int timeoutValue, int min_sec)

This function is used to start the watchdog timer. It loads the timer with the parameter timeoutValue and configures the watchdog timer units for either minutes or seconds. An enum value is provided in the *wdtDLL.h* file which defines valid values for min_sec (SECONDS = 0 and MINUTES = 1).

```
unsigned int timeoutValue = 0x80;

if (EnableTimer(timeoutValue, SECONDS))
       printf("Error starting timer.\n");
```

4.3.4 int DisableTimer ()

This function is used to disable the watchdog timer. It loads the timer with a time-out value of zero and configures the watchdog timer units for seconds.

```
unsigned int timeoutValue = 0x80;

if (EnableTimer(timeoutValue, SECONDS))
       printf("Error starting timer.\n");
```

4.3.5 int CloseSession()

This function is used to disable the watchdog timer and close the driver when complete. If a zero is returned, the timer is disabled and the driver is closed. Otherwise there was an error and the driver is still open.

```
if (CloseSession())
       printf("Error closing driver.\n");
```

4.4 Every function returns an integer value indicating success or failure. If a zero is returned, the function has completed successfully. If a failure occurs, the specific value returned provides more clarity as to the failure mechanism.

4.4.1 DRIVER_ERROR (1)

This error indicates that some function within the driver has failed. This error indicates that one of the IOCTL calls within the driver itself has not completed successfully. Using Windows Device Manager, verify that the driver is loaded and has no resource conflicts.

4.4.2 INVALID_HANDLE (2)

This error indicates that the driver has not initialized or closed. The driver attempts to obtain a handle to the WDT device, and this error indicates that the handle was not obtained. Verify that the driver has loaded successfully.

4.4.3 INVALID_PARAMETER (3)

This error indicates that one of the parameters in the function is out of bounds. Watchdog timer values must be between 0 and 255. A NULL pointer check is used for pointer parameters.

5 Sample Program

5.1 There is one sample program called *WDTPetApp.exe*. This application will enable the timer with a 30 second shut down time, and can be pet and reset to 30 seconds by pressing any key except ‘q’. A separate thread will read the timer and display the remaining time until shutdown. The application will close the WDT and exit when ‘q’ is pressed. The functionality of the timer reset can be verified by letting the timer run out. The computer should reset. The source code for this application is provided in Appendix B: WDTPetApp.cpp.

Appendix A: wdtDLL.h

```
//*****************************************************************************
//
//     Copyright 2017 by WinSystems Inc.
//
//*****************************************************************************
//
//     Name    : wdtDLL.h
//
//     Project : WDT Windows DLL
//
//*****************************************************************************
//
//       Date        Rev                    Description
//     --------    -------     -----------------------------------------
//     04/14/17      1.0          Original Release of DLL
//
//*****************************************************************************

#ifndef _WDT_DLL_H_
#define _WDT_DLL_H_
#include <iostream>

#if defined DLL_EXPORT
#define DECLDIR __declspec(dllexport)
#else
#define DECLDIR __declspec(dllimport)
#endif

extern "C"
{
    DECLDIR int InitializeSession();
    DECLDIR int ReadTimerValue(unsigned int *readValue);
    DECLDIR int WriteTimerValue(unsigned int writeValue);
    DECLDIR int EnableTimer(unsigned int timeoutValue, int min_sec);
    DECLDIR int CloseSession();
}

enum {
    SUCCESS = 0,
    DRIVER_ERROR,
    INVALID_HANDLE,
    INVALID_PARAMETER
};

enum {
    SECONDS = 0,
    MINUTES
};

#endif
```

Appendix B: WDTPetApp.cpp

```
// WDTPetApp.cpp : Defines the entry point for the console application.
//

#include "stdafx.h"
#include "wdtDLL.h"
#include <Windows.h>
#include <conio.h>
#include <process.h>

//reads time every second and prints timeout remaining
void countThread(void *)
{
	unsigned int reading;
	while (1)
	{
		if (ReadTimerValue(&reading))
			std::cout<<"Failed to read\n";

		std::cout << "Current timeout: "<<reading<<"\n";
		Sleep(1000); //1 second sleep
	}
}

int _tmain(int argc, _TCHAR* argv[])
{
	int errCode = 0;
	unsigned int timeout = 30;
	int timeunit = SECONDS;

	//Pet app --- Require key press to reset timer to 30 seconds.
	//If key press is q then it will close the driver and quit.
	try
	{
		//Get access to driver, and check status
		if (errCode = InitializeSession())
		{
			std::cout<<"Failed to open device\n";
			throw errCode;
		}

		if (errCode = EnableTimer(5, timeunit))
		{
			std::cout << "Failed to start WDT with the following timeout and
min_sec args " << timeout << " " << timeunit << "\n";
			throw errCode;
		}
		else
		{
			std::cout << "Set timer to " << timeout << " seconds\n";
		}

		Sleep(2000);//two second sleep

		DisableTimer();
```

```
		//enable WDT in seconds mode with a timeout of 30, and check success
		if (errCode = EnableTimer(timeout, timeunit))
			{
				std::cout << "Failed to start WDT with the following timeout
and min_sec args " << timeout << " " << timeunit << "\n";
				throw errCode;
			}
		else
			{
				std::cout<<"Set timer to "<<timeout<<" seconds\n";
			}

		_beginthread(countThread, 0, NULL);

		//Loops writing 30 second timeout and reading timeout.
		//Waits for keypress to write 30 second timeout
		//q closes the driver and breaks loop
		while (1)
		{
			if (errCode = WriteTimerValue(timeout))
			{
				std::cout<<"Failed to write time to driver\n";
				throw errCode;
			}

			std::cout<<"Press key to reset timer to "<<timeout<<" seconds. Press
q to close WDT and quit application.\n";

			_kbhit();
			if (_getch() == 'q')
			{
				if (errCode = DisableTimer())
				{
					std::cout << "Failed to disable timer.\n";
					throw errCode;
				}

				if (errCode = CloseSession())
				{
					std::cout<<"Failed to close Driver.\n";
					throw errCode;
				}
				std::cout<<"WDT closed. Exiting\n";
				break;
			}
		}
	}
	catch (int err)
	{
		switch (err)
		{
			case DRIVER_ERROR:
			{
				std::cout << "Failed to communicate with driver\n";
				break;
			}
			case INVALID_HANDLE:
```

```
                        {
                                std::cout << "Failed to obtain handle to driver\n";
                                break;
                        }
                        case INVALID_PARAMETER:
                        {
                                std::cout << "Bad parameter\n";
                                break;
                        }
                        default :
                                std::cout << "Unkown error!\n";
                }
                system("pause");
        }
        return 0;
}
```