# PCI
# PC Watchdog™ Programmer's Guide

**BERKSHIRE PRODUCTS, INC.**

Phone: 770-271-0088

http://www.berkprod.com/

Rev: 1.02

# Table of Contents

# 1. Notes

This manuals covers the interface to the functions provided in the PCI_WDog.dll file provided on the CD. This DLL is on the CD in the New_DLL subdirectory.

There is a new Console Ap with source code with examples of interfacing with the new DLL. Berkshire will no longer distribute a separate static library (.LIB file) for the PCI board with these new functions.

The old DLL and static library files are still on the CD in the subdirectory called OLD_DLL_LIB_Files. The old style functions from the old DLL are included in the new DLL as well. It should be fine to intermix them if you a transitioning your code to the new board. However we recommend that you use the WD_OPEN() and WD_Close() functions described in this manual.

A lot of these functions store customized parameters in the non-volatile memory on the watchdog board. The memory is a type of Flash (EEPROM) memory that takes time to program. Allow 15-20 milli-seconds of time for each parameter written.

The latest versions of all manuals and sample code can be found on our site at:

http://www.berkprod.com/

If you have any questions, corrections, or feedback about this manual please contact us at:

man1130feedback@berkprod.com

## 2. Functions

All the functions will return a status of type:  WD_STATUS, defined in the header file: PCI_WDog.dll.

Some functions will internally write additional error or information strings into buffers within the DLL. There is a function in the DLL that can be called to get these strings for further information.

All the functions in the DLL use 32 bit integers and pointers which is the native data size for the PC and Pentium CPUs. The microprocessors on the watchdogs use 8 bit bytes and 16 bit integers. The following function definitions will let you know what the actual data size is within the 32 bit integers.

There are three additional files for use with C or C++. Two are the header files, PCI_WDog.h and old_PCI_WDog.h, with the defines and function definitions. The third is a library file, PCI_WDog.lib, that allows link time binding to the DLL instead of using a Load command in the source.

Section 3 covers the various flags that can be sent or returned by the watchdog unless the flags are very specific to a particular function. They will be covered in the function description.

NOTE: The Watchdog has two operational states – **POD** & Armed/Active:

1. The **P**ower-**O**n-**D**elay state where it waits for 2.5 minutes (or a user time) to allow the PC to complete a re-boot. If the POD DIP Switch is on then the board will remain in the POD state after the 2.5 minute delay until it is "tickled" the first time via the function call to get the temp. The external hardware tickle on the DB-25 connector can also force POD to end.
2. The Armed and Active state is where it start counting down the timer until it gets "tickled". When it gets "tickled" it will reload the countdown timer and start over.

## 2.1 WD_Open

Open the Watchdog board and return a Handle that must used for all other function calls. This function should always be called first.

**WD_STATUS WD_Open(WD_HANDLE *pwdHandle )**

Parameters:

pwdHandle – pointer to a variable of type WD_HANDLE.

Return Value:

WD_OK if successful or a WD error code.

Example:

```
WD_HANDLE wdHandle ;
WD_STATUS wdStatus ;

wdStatus = WD_Open(&wdHandle) ;
if(wdStatus != WD_OK)
{
     // Error Handler
}
```

## 2.2 WD_Close

Closes the Watchdog board This function should always be called last.

**WD_STATUS WD_Close(WD_HANDLE wdHandle )**

Parameters:

wdHandle – Handle of the device from WD_Open().

Return Value:

WD_OK if successful or a WD error code.

Example:

```
WD_HANDLE wdHandle ;
WD_STATUS wdStatus ;

wdStatus = WD_Close(wdHandle) ;
if(wdStatus != WD_OK)
{
      // Error Handler
}
```

## 2.3 WD_GetDllVersion

This function returns a 32 bit encoding of the DLL version. The most significant byte will be set to 0x01 to identify the DLL as the PCI type. The middle high byte contains the major number, the middle low byte is the minor and the lower byte is the sub-minor. For example PCI version 1.09.03 would be returned as: 0x01010903. This call does not require a handle so it can be called before a device open.

I all cases a difference in the sub-minor version can be ignored by your application. This level is reserved for trivial fixes like a spelling fix in an error message.

All version change info is documented in the header file PCI_WDog.h.

**WD_STATUS WD_GetDllVersion(UINT32 *pDllVersion )**

Parameters:

pDllVersion – pointer to a variable to save the version.

Return Value:

Always WD_OK.

Example:

```
UINT32 iVersion

WD_GetDllVersion(&iVersion) ;
printf("DLL Version: %02X.%02X.%02X\n", (iVersion >> 16),
      ((iVersion & 0xff00) >> 8),(iVersion & 0x00ff)),
```

## 2.4 WD_GetErrorInfoMsg

This function can be called to get additional information messages after each function call if any has been written to the internal DLL buffers. If there is no message to return the function will get zero length strings. This function does not require a handle.

**WD_STATUS WD_GetErrorInfoMsg(char *ErrorMsg, char *InfoMsg )**

Parameters:

ErrorMsg – string location for error message if present.
InfoMsg – string location for information message if present.

Return Value:

Always returns WD_OK.

Example:

```
char ErrBuff[INFO_ERR_BUFF_MAXSIZE] ;
char InfBuff[INFO_ERR_BUFF_MAXSIZE] ;
WD_STATUS wdStatus ;

wdStatus = WD_GetErrorInfoMsg(ErrBuff, InfBuff) ;
if(ErrBuff[0] != '\0')
{
      printf("%s\n", ErrBuff) ;
}
if(InfBuff[0] != '\0')
{
      printf("%s\n", InfBuff) ;
}
```

## 2.5 WD_ GetDeviceInfo

Gets information about the PC Watchdog.

**WD_STATUS WD_GetDeviceInfo(WD_HANDLE wdHandle, UINT32* pStat,
UINT32* pDipSw, UINT32* pVer, UINT32* pTick,
UINT32* pDiag )**

Parameters:

wdHandle – Handle of the device from WD_Open().
pStat – pointer to hold status flags from the board.
pDipSw – pointer for current Dip Switch setting – lower 8 bits will match Dip Switch
pVer – pointer to firmware version info returned in two lower bytes.
(Ex: 0x0000012c = version 01.44)
pTick – pointer to 16 bit count of number of times the board has been tickled by reading the temperature. This is not from the external hardware trigger on the DB-25 connector. The tickle count rolls over back to zero after 0xFFFF.
pDiag – pointer to hold diagnostic status flags from the board.

Return Value:

WD_OK if successful or a WD error code.

See Section 3 for a description of the Status and Diagnostic flags that can be returned.

Example:

```
WD_HANDLE wdHandle ;
WD_STATUS wdStatus ;
UINT32 iStat=0, iDsw=0, iVer=0, iTick=0, iDiag=0 ;

wdStatus = WD_GetDeviceInfo(wdHandle, &stat, &dsw, &ver,
                  &tck, &iDiag) ;
if(wdStatus == WD_OK)
{
  printf("Board Status Bits = 0x%04X\n", iStat) ;
  printf("Firmware Version: %02d.%02d\n", (iVer>>8),(iVer&0x0ff));
  printf("Current Dip Switch = 0x%02X\n", iDsw) ;
  printf("Tickle count = %d\n", iTick) ;
  printf("Board Diagnostic Bits = 0x%04X\n", iDiag) ;
}
```

## 2.6 WD_ GetTempTickle

This will be the most used function for the board. It reads the temperature and it “tickles” the board to make it re-load the count down timer. It also increments the tickle count by 1 before it returns the value. The board must be in the armed state to increment the count.

**WD_STATUS WD_GetTempTickle( WD_HANDLE wdHandle, INT32* pTempw,
UINT32* pTempf, UINT32* pTick)**

Parameters:

wdHandle – Handle of the device from WD_Open().
pTempw – pointer to whole number part of temp in ºC. Note this is an INT and can be negative!
pTempf – pointer to flag for fractional portion. If non-zero then add 0.5ºC.
pTick – pointer to 16 bit count of number of times the board has been tickled. 0x0000-0xFFFF

Return Value:

WD_OK if successful or a WD error code.

Example:

```
WD_HANDLE wdHandle ;
WD_STATUS wdStatus ;
UINT32 iTmpf=0, iTick=0 ;
INT32 iTmpw=0 ;

wdStatus = WD_GetTempTickle(wdHandle, &iTmpw, &iTmpf, &iTick) ;
if(wdStatus == WD_OK)
{
  if(iTmpf)                   // flag set for 0.5C?
    iTmpf = 5 ;               // for printf
  printf("Temp = %d.%0d C\n", iTmpw, iTmpf) ;
  printf("Tickle count = %d\n", iTick) ;
}
```

The tickle count rolls over back to zero after 0xFFFF.

** NOTE ** The DLL code reads the temp first and then inserts a short (software loop) to allow the board to update the tickle count. It is possible for the DLL to outrun the processor on the board and show the tickle count being 1 less than you expect.

## 2.7 WD_ SetPowerOnDlyTimes

The standard mode of the Watchdog is to wait 2.5 minutes (150 seconds) after a **P**ower-**O**n-**D**elay (or reboot) of the PC before it arms itself with the watchdog countdown time. This allows time for the PC to complete the reboot sequence. If your PC or application needs more time to reboot, this function provides two alternatives. First you can write a new longer (or shorter) delay time that will only be active for the current restart of the PC. Or you can write a non-zero value to the on-board non-volatile memory that will be used at every reboot in future. If you write a zero (0x0000) to the non-volatile memory then it is disabled and the the board reverts to the 2.5 minute delay at the next reboot.

**WD_STATUS WD_SetPowerOnDlyTimes( WD_HANDLE wdHandle, UINT32 iPod,
UINT32 iNvPod, UINT32 iSetFlag, UINT32* pResFlag)**

Parameters:

wdHandle – Handle of the device from WD_Open().
iPod – a 16 bit value for the new POD time in seconds – 0x0000 to 0xFFFF.
iNvPod – a 16 bit value for the new non-volatile **POD** time in seconds.
iSetFlag – flags for Set operation.
pResFlag – pointer for result flag from function call.

Return Value:

WD_OK if successful or a WD error code.

Flags for Set operations:

WD_POD_SETPOD – must be set to enable the 16 bit value in **iPod** to be written. If clear, then **iPod** value is ignored.
WD_POD_SETNVPOD – must be set to enable the 16 bit value in **iNvPod** to be written to the non-volatile memory. If clear, then **iNvPod** value is ignored.

Result flags:

WD_POD_SET_IGNORE – returned if the Watchdog has already finished the Power-On-Delay and is armed and you sent the WD_POD_SETPOD flag. It is too late to change **POD** at this point.

Example:

```
WD_HANDLE wdHandle ;
WD_STATUS wdStatus ;
UINT32 iPod, iNvPod, iSetFlag, pResFlag ;

iPod = iNvPod = 600 ;       // 600 seconds - 10 minutes
iSetFlag = WD_POD_SETNVPOD | WD_POD_SETPOD ;
pResFlag = 0;
wdStatus = WD_SetPowerOnDlyTimes(wdHandle, iPod, iNvPod, iSetFlag,
                &pResFlag);
if(wdStatus == WD_OK)
{
    if((pResFlag & WD_POD_SET_IGNORE) == 0)   // is WDog armed
        printf("POD Successfully Set\n") ;
    else
        printf("POD Set Fail - WDog already armed\n") ;
}
```

## 2.8 WD_ GetPowerOnDlyTimes

This functions gets the current **POD** time if applicable and the non-volatile time. If the current **POD** time returned is 0x0000, the watchdog is no longer in **POD** and could be armed. If the status does not show armed and the DIP option was selected for an Extended Power-On-Delay then the board is waiting for the first “tickle” If the non-volatile value is not zero then this value is being used for **POD** time at every reboot instead of the fixed 2.5 minute delay.

**WD_STATUS WD_GetPowerOnDlyTimes( WD_HANDLE wdHandle, UINT32* pPod,
UINT32* pNvPod)**

Parameters:

wdHandle – Handle of the device from WD_Open().
pPod – pointer for the current POD time in seconds – 0x0000 to 0xFFFF.
pNvPod – pointer for the current non-volatile **POD** time in seconds – 0x0000 to 0xFFFF.

Return Value:

WD_OK if successful or a WD error code.

Example:

```
WD_HANDLE wdHandle ;
WD_STATUS wdStatus ;
UINT32 iPod, iNvPod ;

iPod = iNvPod = 0;
wdStatus = WD_GetPowerOnDlyTimes(wdHandle, &iPod, &iNvPod);
if(wdStatus == WD_OK)
{
    printf("POD Time = %d\n", iPod) ;
    printf("NV POD Time = %d\n", iNvPod) ;
}
```

## 2.9 WD_ SetWdogTimes

The standard mode of the Watchdog as shipped is to read the last three dip switches for one of eight possible countdown (timeout) delay values. These values range from 5 seconds to 1 hour. This function allows you to change the countdown timer value from 1 to 65535 (0x0001 to 0xFFFF) seconds.

The first option is to send a replacement for the watchdog time that will stay in force as long as the board is powered on (even after resets of the PC). This value is placed in a holding register on the board and will be loaded into the countdown timer the *next time you "tickle" the board*. If this value is set to zero then the board will first check to see if there is a non-volatile value to put in the holding register, otherwise the holding register is cleared. If the holding register is clear the watchdog will revert to the dip switch setting at the next "tickle"

The second option allows you to store your replacement value in non-volatile memory to be used forever. This value you send will also be placed in the holding register. If you send 0x0000 the non-volatile will be cleared, the holding register will clear and the board will revert to the dip switch option at the next "tickle"

**WD_STATUS WD_SetWdogTimes(WD_HANDLE wdHandle, UINT32 iWdTime,**
**UINT32 iNvWdTime, UINT32 iFlag)**

Parameters:

wdHandle – Handle of the device from WD_Open().
iWdTime – a 16 bit value for the new watchdog countdown time in seconds.
iNvWdTime – a 16 bit value for the new non-volatile watchdog time in seconds.
iFlag – flag bits for Set operations

Return Value:

WD_OK if successful or a WD error code.

Flags for Set operations:

WD_SET_TIMEOUT – must be set to enable the 16 bit value in **iWdTime** to be written to the internal holding register.
WD_SETNV_TIMEOUT – must be set to enable the 16 bit value in **iNvWdTime** to be written to the non-volatile memory and the internal holding register.

** NOTE **  If both flags are set, the non-volatile will be processed first and the iWdTime value second. The holding register will end up getting the iWdTime value. The non-volatile value will take effect after the next power cycle.

Example:

```
WD_HANDLE wdHandle ;
WD_STATUS wdStatus ;
UINT32 iWdgTm, iNvWdgTm, iWdgtFl ;

iWdgTm = iNvWdgTm = iWdgtFl = 0;
iWdgTm = 600 ;         // 600s = 10 minutes for new Wdog time
iNvWdgTm = 600 ;       // 10 minutes for nv memory for new Wdog time
iWdgtFl = WD_SET_TIMEOUT | WD_SETNV_TIMEOUT ;
wdStatus = WD_SetWdogTimes(wdHandle, iWdgTm, iNvWdgTm, iWdgtFl);
if(wdStatus == WD_OK)
{
    printf("\t-- WDG TIME -- New Countdown Timers Write OK\n") ;
}
```

## 2.10 WD_ GetWdogTimes

This function returns four 16 bit values. The first is the current watchdog countdown time remaining if the board is armed. If the watchdog is still in **POD** time then the value returned will be 0xFFFF. The second value is the stored non-volatile time that will be used at power up. If this value is 0x0000 then it is disabled. The third value is what is currently in the holding register. If it is non-zero then this value will be reloaded into the countdown timer each time the watchdog is "tickled". The last value returned is the time selected by the dip switches that will be used if the holding register is clear.

**WD_STATUS WD_GetWdogTimes( WD_HANDLE wdHandle, UINT32* pWdTime,**
**UINT32* pNvWdTime, UINT32* pHoldRegTime, UINT32* pDipSwTime)**

Parameters:

wdHandle – Handle of the device from WD_Open().
pWdTime – pointer to 16 bit value for the current watchdog time remaining or 0xFFFF if not armed.
pNvWdTime – pointer to 16 bit value for the non-volatile watchdog time in seconds.
pHoldRegTime - pointer to 16 value for the time in the holding register.
pDipSwTime – pointer to 16 bit value for time selected on dip switches that will be used if the holding register is clear.

Return Value:

WD_OK if successful or a WD error code.

** NOTE ** PCI PC Watchdog firmware prior to Version 1.72 can not read back the holding register value. The pHoldRegTime value will always be set to 0x0000.

Example:

```
WD_HANDLE wdHandle ;
WD_STATUS wdStatus ;
UINT32 iWdgTm, iNvWdgTm, iHoldReg, iDipSwTime ;

iWdgTm = iNvWdgTm = iHoldReg = iDipSwTime = 0 ;
wdStatus = WD_GetWdogTimes(wdHandle,  &iWdgTm, &iNvWdgTm, &iHoldReg,
                           &iDipSwTime);
if(wdStatus == WD_OK)
{
    printf("Current Countdown Time = %d\n", iWdgTm) ;
    printf("Current NV Countdown Time = %d\n", iNvWdgTm) ;
    printf("Current Holding Register Time = %d\n", iHoldReg) ;
    printf("DipSwitch Time = %d\n", iDipSwTime) ;
}
```

## 2.11 WD_ GetPciAnalogIn

This function returns the reading from the analog input on the PCI DB-25 (DSub) connector. The analog value is 8 bits and the 256 values range from 0.0V to 5.0V. Do not exceed these levels on this input.

Note that the analog input on the PCI ranges from 0.0 to 5.0V instead of the upper limit of 3.3V as on the Ether USB PC Watchdog.

**WD_STATUS WD_GetPciAnaloglIn(WD_HANDLE wdHandle, UINT32* pAi)**

Parameters:

wdHandle – Handle of the device from WD_Open().
pAi – pointer to 8 bit value for the analog input value.

Return Value:

WD_OK if successful or a WD error code.

Example:

```
WD_HANDLE wdHandle ;
WD_STATUS wdStatus ;
UINT32 iAnIn;

wdStatus = WD_GetPciAnalogIn(wdHandle, &iAnIn) ;
if(wdStatus == WD_OK)
{
    iAnIn = ((iAnIn * 500)/255) ;             // 500 = 5.0V * 100
    printf("Analog In = %d.%02dV\n", (iAnIn/100), (iAnIn % 100));
}
```

## 2.12 WD_ GetSetPciDigitalInOut

This function allows you to read the 4 general purpose digital inputs and set the four general purpose digital outputs on the PCI DB-25 (DSub) connector. The digital output data is in the lowest four bits of the iDigOut variable. The data read back in pDigIn is 8 bits. The lower 4 bits reflect what was on the output port **before** your new data output write. The upper four bits of iDigIn reflect the current input values on the digital inputs. The digital input pins DI3-0 map to D7-4 of pDigIn.

**WD_STATUS WD_GetSetPciDigitalInOut(WD_HANDLE wdHandle, UINT32 iDigOut, UINT32* pDigIn, UINT32 iFlag)**

Parameters:

wdHandle – Handle of the device from WD_Open().
iDigOut – 4 bit value to set the digital output pins.
pDigIn – lower 4 bits (D3-0) are the current digital output values **before** the write. The next 4 bits (D7-4) are the digital inputs.
iFlag – flag to enable writing output port data.

Return Value:

WD_OK if successful or a WD error code.

Flag:

WD_PCI_DIG_OUT_EN – this flag must be set to enable writing the digital outputs.

** NOTE ** The IO pins on the PCI board have electrical inversion. The output pins are driven by open-collector drivers with pull-up resistors. The reset state of the board sets the output bits to zero (0) which gets inverted by the drivers so the outputs are pulled high. Writing a one (1) to an output pin cause the driver to turn on and pull to ground potential.

The input pins have their inputs pulled up with resistors to +5.0V. Inversion causes the input bits to read as zero. If you pull an input pin low to ground (ex: dry contact switch) then the input bit will read as a one (1).

Example:

```
WD_HANDLE wdHandle ;
WD_STATUS wdStatus ;
UINT32 iDigOut, pDigIn, iFlag ;

iDigOut = 0x06 ;           // set two of the bits
iFlag = WD_PCI_DIG_OUT_EN ;
wdStatus = WD_GetSetPciDigitalInOut(wdHandle, iDigOut, &pDigIn,
                iFlag);
if(wdStatus == WD_OK)
{
     printf("\t-- DIG OUT -- Prior Digital Out = 0x%02X\n",
                (pDigIn & 0x0f)) ;
      printf("\t-- DIG IN  -- Current Digital In = 0x%02X\n",
                ((pDigIn & 0xf0) >> 4)) ;
}
iFlag =0 ;             // now do read only
wdStatus = WD_GetSetPciDigitalInOut(wdHandle, iDigOut, &pDigIn,
                iFlag);
if(wdStatus == WD_OK)
{
     printf("\t-- DIG OUT -- Current Digital Out = 0x%02X\n",
                (pDigIn & 0x0f)) ;
}
```

## 2.13 WD_ GetSetPciRelays

This function gives you some control of the two relays on the board. The operation of the two relays is also governed by the setting of the DIP Switch options as well. Turning the relays on and off is done by firmware on the board. The returned status is the state of the relays **before** any changes are applied.

Relay #1 has the option of being inverted. When it is inverted it will do the opposite of the DIP Switch options at the next watchdog reboot. There is also a non-volatile memory option to force immediate inversion every time the board powers up. Note that inversion does not apply to the ON/OFF operation in this function. If you send a relay #1 ON commend then the relay actually turns on. If you know that you have inversion in effect make appropriate changes in your call to this function.

The board also has a hardware option to allow you to get exclusive control of relay 2 completely locking out the processor on the board. This function also provide the hardware options which always override the software.

**WD_STATUS WD_GetSetPciRelays(WD_HANDLE wdHandle, UINT32 iRelaySet,
UINT32* pRelayGet)**

Parameters:

wdHandle – Handle of the device from WD_Open().
iRelaySet – flags sent for control.
pRelayGet – pointer to flags returned from the board

Return Value:

WD_OK if successful or a WD error code.

Flags for Set operations:

WD_PCI_INVRT_EN – set this flag for relay #1 inversion enable. Note that the DIP Switch options will be inverted at the next watchdog reboot and the board will power up with Relay #1 on. If you need inversion immediately you should use the flags to set the relay #1 on as well.
WD_PCI_HDW2__EN – this flag must be set to force exclusive use of Relay #2. It will be used to set or clear the hardware Port #1 options on the board.
WD_PCI_RLY2__EN – this flag must be set to turn the relay #2 on or off. This Relay #2 flag is the software option done by firmware on the board.
WD_PCI_RLY1__EN – this flag must be set to turn the relay #1 on or off. Relay #1 is always a software option done by firmware on the board.

Flags for Set and Get operations:

WD_PCI_NV_INVRT – set this flag (along with WD_PCI_INVRT_EN) if you want the inversion option you select with WD_PCI_INVRT_ON to be saved in non-volatile memory also.
WD_PCI_INVRT_ON - set this flag (along with WD_PCI_INVRT_EN) if you want relay #1 operation to become inverted at the next watchdog reboot.. **pRelayGet** will contain this flag if the option is enabled.
WD_PCI_HDW_EXCL_ON - set or clear this flag (along with WD_PCI_HDW2_EN) if you want to gain exclusive control of relay #2. **pRelayGet** will contain this flag if relay #2 exclusive control is currently on. This is the hardware option to gain control of Relay #2 and uses the **R2DS** bit in hardware Port #2.
WD_PCI_HDW_RLY2_ON - set or clear this flag (along with WD_PCI_HDW2_EN) if you want to turn relay #2 on. If it is clear the relay will be turned off. **pRelayGet** will contain this flag if relay #2 is currently on. This is the hardware option to turn on Relay #2 and uses the **RLY2** bit in hardware Port #2.
WD_PCI_RLY2_ON - set or clear this flag (along with WD_PCI_RLY2_EN) if you want to turn relay #2 on or off. If it is clear the relay will be turned off. **pRelayGet** will contain this flag if relay #2 is currently on. This is the firmware option to turn on Relay #2.
WD_PCI_RLY1_ON - set or clear this flag (along with WD_PCI_RLY1_EN) if you want to turn relay #1 on or off. If it is clear the relay will be turned off. **pRelayGet** will contain this flag if relay #1 is currently on.

Example:

```
WD_HANDLE wdHandle ;
WD_STATUS wdStatus ;
UINT32 iRlySet, iRlyGet, i ;

// Firmware turn On relay #1
iRlySet = (WD_PCI_RLY1_EN | WD_PCI_RLY1_ON) ;
// Hardware turn on relay #2 & exclusive
iRlySet |= (WD_PCI_HDW2_EN | WD_PCI_HDW_RLY2_ON
           | WD_PCI_HDW_EXCL_ON) ;
// Ask for relay #1 inversion and save in nv memory
iRlySet |= (WD_PCI_INVRT_EN | WD_PCI_INVRT_ON | WD_PCI_NV_INVRT) ;
wdStatus = WD_GetSetPciRelays(wdHandle, iRlySet, &iRlyGet);
if(wdStatus == WD_OK)
{
     for(i=0; i<0x00ffffff; i++)   // short loop to hear relays click
          ;
     printf("\t-- PCIRELAY -- Relay Status was = 0x%04X\n",
                iRlyGet) ;
}
// just read new status now
iRlySet = 0 ;
wdStatus = WD_GetSetPciRelays(wdHandle, iRlySet, &iRlyGet);
if(wdStatus == WD_OK)
{
     printf("\t-- PCIRELAY -- New Relay Status = 0x%04X\n",
                iRlyGet) ;
}
// finally undo everything
// Firmware turn Off relay #1
iRlySet = WD_PCI_RLY1_EN  ;
// Hardware turn off relay #2 & exclusive
iRlySet |= WD_PCI_HDW2_EN ;
// turn relay #1 inversion and clear nv memory
iRlySet |= (WD_PCI_INVRT_EN  | WD_PCI_NV_INVRT) ;
wdStatus = WD_GetSetPciRelays(wdHandle, iRlySet, &iRlyGet);
```

## 2.14 WD_ EnableDisable

This functions allows you to Enable or Disable the watchdog. When the watchdog is disabled and it is armed  it will act like it it is being tickled and continuously reload the countdown timer. When it is removed from the disabled state and it is armed it will start counting down again.

If you disable the watchdog during Power-On-Delay (**POD**), the board will finish the **POD** timer and entered the armed state, but be disabled from counting down.

**WD_STATUS WD_EnableDisable(WD_HANDLE wdHandle, UINT32 iFlagSet)**

Parameters:

wdHandle –  Handle of the device from WD_Open().
iFlagSet – flags for enable or disable

Return Value:

WD_OK if successful or a WD error code.

Flags Set Operations:

WD_WDOG_ENABLE – set this flag to enable the watchdog.
WD_WDOG_DISABLE – set this flag to disable the watchdog.

** NOTES **

If both flags are set the watchdog will be disabled.

On the PCI board the disable is handled by one of the hardware ports on the board. If you check status after sending this command the WD_STAT_PCI_WDIS flag should be set immediately. The WD_STAT_CMD_DISABLED flag might take up to 250 micro-seconds for the on board firmware to recognize the hardware and update its status accordingly. The example on the next page shows using both bits to test for a disable.

Example:

```
WD_HANDLE wdHandle ;
WD_STATUS wdStatus ;
UINT32 iStat=0, iDsw=0, iVer=0, iTick=0, iDiag=0  ;
UINT32 iEnDisFlag ;

iEnDisFlag = WD_WDOG_DISABLE ;
wdStatus = WD_EnableDisable(wdHandle, iEnDisFlag);
wdStatus |= WD_GetDeviceInfo(wdHandle, &iStat, &iDsw, &iVer,
           &iTick, &iDiag);
if(wdStatus == WD_OK)
{
    if(iStat & (WD_STAT_CMD_DISABLED | WD_STAT_PCI_WDIS))
        printf("Watchdog is DISabled\n") ;
    else
        printf("Watchdog is ENabled\n") ;
}
```

## 2.15 WD_ GetResetCount

Each time the watchdog resets the PC it will increment a counter that runs from 0 to 255 (0x00-0xFF). The count will not rollover, it stops at 255. This command allows you to get the reset count and optionally clear it after the current value has been retrieved. If you clear the reset count, the watchdog will also clear the bottom LED as well.

**WD_STATUS WD_GetResetCount(WD_HANDLE wdHandle, UINT32 iFlag,
UINT32* pRstCnt)**

Parameters:

wdHandle – Handle of the device from WD_Open().
iFlag – flag to clear the reset counter after retrieving.
pRstCnt – pointer to store the current reset count.

Return Value:

WD_OK if successful or a WD error code.

Flag:

WD_CLEAR_RST_CNT – set this flag to clear the reset count after it has been retrieved. It will also clear the bottom LED at the back of the board. Leave flag cleared if you just want the count.

Example:

```
WD_HANDLE wdHandle ;
WD_STATUS wdStatus ;
UINT32 iFlag, iRstCnt ;

iFlag = WD_CLEAR_RST_CNT ;          // clear count & bottom LED
wdStatus = WD_GetResetCount(wdHandle, iFlag, &iRstCnt);
if(wdStatus == WD_OK)
{
    printf("Reset Count = %d\n", iRstCnt) ;
}
```

## 2.16 WD_ GetSetNvTempOffset

The watchdog board monitors the temperature and looks for trip points to start the buzzer and optionally reset the PC. The process is described in the Hardware Manual for Dip Switch #4. This function allows you to increase the trip points in 1 degree centigrade increments up to 31 degrees (0x1F). This value is always stored in the non-volatile memory and becomes effective immediately.

**WD_STATUS WD_GetSetNvTempOffset(WD_HANDLE wdHandle, UINT32 iFlag, UINT32 iNvOffset, UINT32* pCurOffset)**

Parameters:

wdHandle – Handle of the device from WD_Open().
iFlag – flag to enable writing the non-volatile memory
iNvOffset – new offset value for non-volatile memory
pCurOffset – pointer to current non-volatile value. In case of a write, it will equal what you just wrote.

Return Value:

WD_OK if successful or a WD error code.

Flag:

WD_TEMP_OFF_WREN – this flag must be set to store the new offset value. If it is clear then the function just returns the current stored value in **pCurOffset**.

Example:

```
WD_HANDLE wdHandle ;
WD_STATUS wdStatus ;
UINT32 iFlag, iNvOffset, iCurOffset ;

iNvOffset = 11 ;              // set 11C offset
iFlag = WD_TEMP_OFF_WREN ;    // enable nv write
wdStatus = WD_GetSetNvTempOffset(wdHandle, iFlag, iNvOffset,
                &iCurOffset);
if(wdStatus == WD_OK)
{
    printf("NV Temp Offset = %d\n", iCurOffset) ;
}
```

## 2.17 WD_ SetBuzzer

The buzzer on the board defaults to a 0.6 second (600 milli-second) beep at power up and after each re-boot. The power up buzzer is fixed, but this function allows you to change the buzzer time for subsequent watchdog timeout reboots. This function also allows you to turn the buzzer on or off.

The values sent for the PCI board are different than the Ether-USB board since the PCI board has limited buzzer control. If you set the iBuzzTime variable to a non-zero value it will turn on the buzzer and leave it on until you send a value of zero.

The value sent for the non-volatile memory value allows you three options for the reboot buzzer. If the value equals 255 (0xFF) the reboot buzzer will be on continuously and you will have to use this function to turn it off. If the value equals 250 (0xFA) then the reboot buzzer time will be set to 2.5 seconds. Any other value for the non-volatile memory value causes the board reverts back to the 0.6 second buzzer at each re-boot.

The PCI DLL does support a complete buzzer disable flag. One flag for a disable as long as the board is power up and another to store the disable permanently in non-volatile memory.

**WD_STATUS WD_SetBuzzer(WD_HANDLE wdHandle, UINT32 iBuzzTime,
UINT32 iNvBuzzTime, UINT32 iFlags)**

Parameters:

wdHandle – Handle of the device from WD_Open().
iBuzzTime – non-zero value to turn buzzer on. Send 0x00 to turn the buzzer off.
iNvBuzzTime – reboot buzzer time value for non-volatile memory.
iFlags – flags to enable buzzer on and write to non-volatile memory.

Return Value:

WD_OK if successful or a WD error code.

Flags:

WD_BUZZ_ON_EN – this flag must be set to turn on/off the buzzer with iBuzzTime.
WD_BUZZ_NV_WREN – this flag must be set to write non-volatile memory.

WD_PCI_BUZZ_DIS_EN – this flag must be set to do any disable operations.
WD_PCI_BUZZ_DISBL – If this flag is set (along with WD_PCI_BUZZ_DIS_EN) then the buzzer is disabled as long as the board is powered on. If clear the buzzer is re-enabled.
WD_PCI_NVBUZZ_DISBL – if this flag is set (along with WD_PCI_BUZZ_DIS_EN) the non-volatile memory will be updated with status of WD_PCI_BUZZ_DISBL.

Example:

```
WD_HANDLE wdHandle ;
WD_STATUS wdStatus ;
UINT32 iBuzzTime, iNvBuzzTime, iFlags, i ;

iBuzzTime = 0x01;     // buzzer on
iNvBuzzTime = 250;    // 0xfa - reboot buzzer = 2.5 seconds
iFlags = WD_BUZZ_ON_EN |WD_BUZZ_NV_WREN ;
wdStatus = WD_SetBuzzer(wdHandle, iBuzzTime, iNvBuzzTime, iFlags);
if(wdStatus == WD_OK)
{
    printf("\t-- BUZZER -- Buzzer should be ON\n") ;
}
for(i=0; i<0x0fffffff; i++)
      ;
iBuzzTime = 0x0;           // buzzer off
iFlags = WD_BUZZ_ON_EN ;
wdStatus = WD_SetBuzzer(wdHandle, iBuzzTime, iNvBuzzTime, iFlags);
```

## 2.18 WD_ GetBuzzer

This function returns the buzzer times set with the prior command. The variable pBuzzTime returns three values on the PCI board. If the buzzer is off it returns 0x00, if it is on it returns 0x01, and if the buzzer is disabled it returns 0x80. The pNvBuzzTime returns three values also on the PCI board. If the normal 0.6 reboot buzzer is active it returns 0x00, if the 2.5 second buzzer is enabled it returns 0xFA (250), and it will return 0xFF if the continuous reboot buzzer is active.

**WD_STATUS WD_GetBuzzer(WD_HANDLE wdHandle, UINT32* pBuzzTime, UINT32* pNvBuzzTime**

Parameters:

wdHandle – Handle of the device from WD_Open().
pBuzzTime – pointer to 8 bit buzzer time returned. If non-zero then the buzzer is on.
pNvBuzzTime – pointer to 8 bit reboot buzzer time value from non-volatile memory.

Return Value:

WD_OK if successful or a WD error code.

Example:

```
WD_HANDLE wdHandle ;
WD_STATUS wdStatus ;
UINT32 iBuzzTime, iNvBuzzTime ;

iBuzzTime = iNvBuzzTime = 0;
wdStatus = WD_GetBuzzer(wdHandle, &iBuzzTime, &iNvBuzzTime);
if(wdStatus == WD_OK)
{
    printf("Buzz Time = %d\n", iBuzzTime) ;
    printf("NV Re-Boot Buzzer Time = %d\n", iNvBuzzTime) ;
}
```

## 2.19 WD_ GetSetNvRelayPulse

Each time the watchdog times out and re-boots the PC, it defaults to activating the reset relay for 2.5 seconds. In a few very rare instances, some older Dell machines (pre 2000 with ISA slots) have gone into setup mode if reset is held this long. This function permits you to increase or decrease the reset relay active time. The 8 bit time is always written to non-volatile memory and the time is in increments of 50 milli-second (0.05 second) tics. Write a value of zero to clear this option and return to the default 2.5 second activation. The max value is 255 or 12.75 seconds.

**WD_STATUS WD_GetSetNvRelayPulse( WD_HANDLE wdHandle, UINT32 iFlag, UINT32 iNvRelayPulse, UINT32* pCurRelayPulse)**

Parameters:

wdHandle – Handle of the device from WD_Open().
iFlag – flag to enable writing the non-volatile memory
iNvRelayPulse – new 8 bit relay pulse value for non-volatile memory (0 - 255)
pCurRelayPulse – pointer to current non-volatile value. In case of a write, it will equal what you just wrote.

Return Value:

WD_OK if successful or a WD error code.

Flag:

WD_RLY_PLS_WREN – this flag must be set to store the new relay value. If it is clear then the function just returns the current stored value in **pCurRelayPulse**.

Example:

```
WD_HANDLE wdHandle ;
WD_STATUS wdStatus ;
UINT32 iFlag, iNvRelayPulse, iCurRelayPulse ;

iNvRelayPulse = 25 ;           // pulse = 25 * 0.05 = 1.25 seconds
iFlag = WD_RLY_PLS_WREN ;      // enable nv write
wdStatus = WD_GetSetNvRelayPulse(wdHandle, iFlag, iNvRelayPulse,
           &iCurRelayPulse);
if(wdStatus == WD_OK)
{
    printf("Pulse time = %d - 50mS Tics\n", iCurRelayPulse) ;
}
```

## 2.20 WD_ GetSetNvUserCode

This command allows you to store your own unique 8 byte user code or information in the non-volatile memory on the watchdog.

**WD_STATUS WD_GetSetNvUserCode(WD_HANDLE wdHandle, UINT32 iFlag, UCHAR* pNvUserCode, UCHAR* pCurNvUserCode)**

Parameters:

wdHandle – Handle of the device from WD_Open().
iFlag – flag to enable writing the non-volatile memory
pNvUserCode – pointer to an 8 byte (char) array to be written if the flag is set.
pCurNvUserCode – pointer to an 8 byte (char) array retrieve the current non-volatile code. In case of a write, it will equal what you just wrote.

Return Value:

WD_OK if successful or a WD error code.

Flag:

WD_USE_CODE_WREN – this flag must be set to store the new code value. If it is clear then the function just returns the current stored value in non-volatile memory.

Example:

```
WD_HANDLE wdHandle ;
WD_STATUS wdStatus ;
UINT32 iFlag, i;
UCHAR bNvUserCode[8], bCurNvUserCode[8] ;

for(i=0; i<8; i++)
    bNvUserCode[i] = i+1;
iFlag = WD_USE_CODE_WREN ;    // enable nv write
wdStatus = WD_GetSetNvUsbUserCode(wdHandle, iFlag, bNvUserCode,
                    bCurNvUserCode);
if(wdStatus == WD_OK)
{
    printf("Code = ");
    for(i=0; i<8; i++)
        printf(" 0x%02X ", (int)bCurNvUserCode[i]) ;
    printf("\n") ;
}
```

## 2.21 WD_ EnableDisablePcReset

This command allows you to force an immediate PC reset. There is also a 8 bit delay time that is passed as the number of seconds (0x00-0xFF) before the reset occurs. Values larger than 255 (0xFF) will be truncated to 255. Set this time to zero for an immediate reset.

Once this command has been sent the PCI PC Watchdog will no longer respond to any more commands and it will ignore all "tickles" from the temperature read and the External Trip input on the DB-25.

**WD_STATUS WD_EnableDisablePcReset(WD_HANDLE wdHandle, UINT32 iFlag, UINT32 iResetTime, UINT32* pGetTime)**

Parameters:

wdHandle – Handle of the device from WD_Open().
iFlag – flag to enable or disable the reset
iResetTime – 8 bit delay value in seconds before reset occurrs.
pGetTime – on PCI this will always return a 0x00.

Return Value:

WD_OK if successful or a WD error code.

Note: if you send a time of zero then you will not get the return – the PC should reset almost immediately.

Flag:

WD_PC_RESET_EN – this flag must be set to enable the reset to start.
WD_PC_RESET_DIS – this flag is ignored in PCI PC Watchdog DLL.

Example:

```
WD_HANDLE wdHandle ;
WD_STATUS wdStatus ;
UINT32 iFlag, iResetTime, iCurTime ;

iResetTime = 2 ;            // 2 seconds till PC reset
iFlag = WD_PC_RESET_EN ;    // enable reset
wdStatus = WD_EnableDisablePcReset(wdHandle, iFlag, iResetTime,
                &iCurTime);
if(wdStatus == WD_OK)
{
    printf("\t-- Reset PC - Command Accepted\n");
}
```

## 3. System Status / Information Flags

These flags and their actual bits are also listed in the header (.h) file.

### 3.1 Status Flags

**WD_STAT_ACTIVE_ARMED** - the watchdog is armed and done with POD time.

**WD_STAT_POD_ACTIVE** - the watchdog is still in 2.5 minute (or user time) delay.

**WD_STAT_POD_DSW_DELAY** – indicates that the board has finished the POD time and is waiting for the first "tickle".

**WD_STAT_CMD_DISABLED** - the watchdog has acknowledged the WDIS bit in Port #2 and is disabled.

**WD_STAT_PCI_ENTP** – shows the status of the ENTP bit in Port #2 of the PCI board. If this bit is set and the DIP Switch option is selected then the PCI board can hold the PC in reset after the upper temp limit is tripped.

**WD_STAT_PCI_WDIS** – shows the status of the WDIS bit in Port #2 of the PCI board. If this bit is set then the PCI board has been disabled.

**WD_STAT_PCI_TTRP** – shows the status of the TTRP bit in Port #1 of the PCI board. If this bit is set then the PCI board has signaled that the first temperature trip point has been exceeded.

**WD_STAT_PCI_WTRP** – shows the status of the WTRP bit in Port #1 of the PCI board. If this bit is set then the PCI board has reset the PC one or more times.

### 3.2 Diagnostic Flags

**WD_DIAG_TEMP_OK** – This bit should always be set. If it is clear the temperature sensor IC on the board has failed.

**WD_DIAG_NVMEM_OK** – This bit should always be set. If it is clear the non-volatile memory IC on the board has failed or a checksum error was found resulting in the memory being cleared. If this bit stays set even after cycling power on the board then the memory has failed and should be returned for repair.