Ethernet - USB PC Watchdog™ USB Interface 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 EUWDog-Int.dll file provided on the CD. This DLL relies on an additional DLL file, FTD2xx.dll that is installed along with the FTDI drivers in the WINNT (Windows) – Sytem32 directory.

This DLL only provides functions to interface to the USB portion of the board. The Ethernet DLL access is in the EUWDog-Eth-Int.dll file.

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

There are sample programs in C++ (Console AP) and VB.NET on the CD that can be used as a starting point for your own application. The C Console AP is in the TestAllDllConsoleAp directory. The VB.Net files are in the VB_NET_DLL_Sample directory.

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:

man1160feedback@berkprod.com

2. Functions

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

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.

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.
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. The initial timer start can be delayed by a Dip Switch until the first "tickle" or the watchdog can be disabled by command function to **DISABLE** the countdown. In either case the watchdog is still considered to be in its Armed and Active state. Therefore it is possible to get the status flags: **WD_STAT_ACTIVE_ARMED & WD_STAT_CMD_DISABLED** both set.

NOTE: The functions (commands) in the first part of this manual can work with the watchdog through the USB or the Ethernet interface. The Ethernet ones have a slightly different name and use the Ethernet DLL file. There is a function that allows you to limit which commands will be allowed from the Ethernet side. The functions (commands) that only work on the USB side are at the end of this section and have **(USB Only)** in their title declaration.

NOTE: Once the board has a valid IP address (fixed, non-volatile, or DHCP) it will respond to pings.

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 0x00 to identify the DLL as the ETHER-USB 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 version 1.09.03 would be returned as: 0x00010903. 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 EUWDog-Int.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. 0x0000-0xFFFF
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.

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.

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.

<u>Example:</u>

```
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 is armed. If the non-volatile value is not zero then this value is being used for **POD** time at every reboot.

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 place 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 it 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 holding register will get the other time sent. The non-volatile value would take effect after the next power cycle.

<u>Example:</u>

```
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.
pHoldTime - 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.

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_ GetAnalogDigitalIn

This function returns the values from the analog and digital inputs on the 1/8” stereo jack. The analog value is 8 bits and the 256 values range from 0.0V to 3.3V. Do not exceed these levels on this input. The digital input is returned as a group of flags (or bits). The only value passed to this function is a set of flags to configure the digital input for edge sensing for “tickling” the watchdog.

```
WD_STATUS WD_GetAnalogDigitalIn(WD_HANDLE wdHandle, UINT32* pAi,
                        UINT32* pDi, UINT32 iFlag)
```

Parameters:

wdHandle – Handle of the device from WD_Open().
pAi – pointer to 8 bit value for the analog input value.
pDi – pointer to 8 bit value for digital input flags
iFlag – flag bits for Set edge configure options

Return Value:

WD_OK if successful or a WD error code.

Flags for Set operations:

WD_DIG_IN_EDGE_EN – this flag (bit) must be set to change the current edge sensing.
WD_DIG_IN_EDGE – if this flag is set (along with DIG_IN_EDGE_EN), the board will enable edge sense “tickles” on the digital input. Clear this flag to disable them.
WD_DIG_IN_NVWR_EN – this bit must be set in order to update the non-volatile memory setting for edge triggers.
WD_DIG_IN_NVEDGE – if this flag is set (along with DIG_IN_NVWR_EN), the board will enable edge trigger in non-volatile memory. Clear this flag to clear non-volatile memory.

Flags returned in **pDi**:

WD_DIG_IN_EDGE_ACT – edge detection is currently active
WD_DIG_IN_NVEDGE_ACT – non-volatile edge enable is active
WD_DIGITAL_IN_EDGE – edge (rising) captured on digital input. Note that the board also checks and clears this status if it is doing edge detection for “tickles”.
WD_DIGITAL_IN_HIGH – digital input is high – otherwise it is low.

Example:

```
WD_HANDLE wdHandle ;
WD_STATUS wdStatus ;
UINT32 iDgIn, iAnIn, iDiFl;

iDiFl = 0 ;      // no flags for edge or nv edge enable
wdStatus = WD_GetAnalogDigitalIn(wdHandle, &iAnIn, &iDgIn, iDiFl) ;
if(wdStatus == WD_OK)
{
    iAnIn = ((iAnIn * 330)/255) ;             // 330 = 3.3V * 100
    printf("Analog In = %d.%02dV\n", (iAnIn/100), (iAnIn % 100));
    printf("Digital input bits = 0x%02X\n", iDgIn) ;
    if(iDgIn & WD_DIGITAL_IN_HIGH)
        printf("\t-- DIGITALIN -- Input is High\n") ;
    else
        printf("\t-- DIGITALIN -- Input is Low\n") ;
}
```

2.12 WD_ GetSetAuxRelay

This function gives you control of the Axillary Relay on the board. You can turn the relay on or off and make it perform actions each time the watchdog resets the PC. One option sets the Aux relay to generate a pulse the same length as the reset relay pulses when the watchdog resets the PC. There is an option in another function to change the pulse lengths. The other option turns on the relay after the first reset and leaves it on. You will need to call this function later if you want to turn off the relay.

**WD_STATUS WD_GetSetAuxRelay(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_AUX_WRRB_NV_EN – this flag (bit) must be set to change the non-volatile memory options.
WD_AUX_ONOFF_EN – this flag must be set to turn the relay on or off.

Flags for Set and Get operations:

WD_AUX_RST_LTCH – set this flag (along with AUX_WRRB_NV_EN) if you want the non-volatile option to turn the relay on at the first reset. **pRelayGet** will contain this flag if the option is enabled.
WD_AUX_RST_PULS set this flag (along with AUX_WRRB_NV_EN) if you want the non-volatile option to pulse the relay at every reset. **pRelayGet** will contain this flag if the option is enabled.
WD_AUX_RLY_ON - set this flag (along with AUX_ONOFF_EN) if you want to turn the relay on. If it is clear the relay will be turned off. **pRelayGet** will contain this flag if the Aux relay is currently on.

<u>Example:</u>

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

iRlySet = AUX_ONOFF_EN | AUX_RLY_ON ;
wdStatus = WD_GetSetAuxRelay(wdHandle, iRlySet, &iRlyGet);
if(wdStatus == WD_OK)
{
    printf("Aux relay bits = 0x%02X\n", iRlyGet) ;
    if(iRlyGet & WD_AUX_RLY_ON)
        printf("Aux relay is on.\n") ;
    else
        printf("Aux relay is off.\n") ;
}
```

2.13 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.

** NOTE ** If both flags are set the watchdog will be disabled.

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)
        printf("Watchdog is DISabled\n") ;
    else
        printf("Watchdog is ENabled\n") ;
}
```

2.14 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. Will also clear the bottom LED. 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) ;
}
```

2.15 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) ;
}
```

2.16 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 watchdog timeout reboots. This function also allows you to turn the buzzer on or off.

The values sent are bytes that range from 0x00 to 0xFF and represent 10 milli-second (0.01 second) tics of buzzer time. The value of 0xFF is a special case that will turn on the buzzer and leave it on forever. If sent, you will need to turn the buzzer off with a buzzer time of zero. If you send a zero for the non-volatile memory value, the board reverts back to the 0.6 second buzzer at each re-boot. You can effectively disable the reboot buzzer by setting its value to 0x01, since a 10 milli-second beep will barely be heard.

The Dip Switch #3 completely disables ALL buzzer activity.

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

Parameters:

wdHandle – Handle of the device from WD_Open().
iBuzzTime – time for buzzer to sound. 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.

Flag:

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

Example:

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

iBuzzTime = 80;             // buzzer 0.8 seconds
iNvBuzzTime = 150;          // reboot buzzer = 1.5 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") ;
}
```

2.17 WD_ GetBuzzer

This function returns the buzzer times set with the prior command. All the times are in 10 milli-second (0.01 second) tics. If the current buzzer time returned is a non-zero value, then the buzzer is on.

The Dip Switch #3 completely disables ALL buzzer activity.

**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("Buzzer Time = %d\n", iBuzzTime) ;
    printf("NV Re-Boot Buzzer Time = %d\n", iNvBuzzTime) ;
}
```

2.18 WD_ GetSetNvRelayPulse

Each time the watchdog times out and re-boots the PC, it defaults to activating the reset relay for 2.5 seconds. If the Auxiliary relay is set for pulse mode then it will also activate 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 (and Aux 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) ;
}
```

2.19 WD_ GetSetNvUsbUserCode

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_GetSetNvUsbUserCode(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_USB_SCOD_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_USB_SCOD_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") ;
}
```

2.20 WD_ EnableDisablePcReset

This command allows you to force a PC reset. There is also a 16 bit delay time that is passed as the number of seconds (0x0000-0xFFFF) before the reset occurs. Set this time to zero for an immediate reset. The delay value will be stored in the watchdog countdown timer register and can also be checked with the: **WD_GetWdogTimes** function. Once this command has been sent the watchdog will set the **WD_STAT_RESET_PEND** flag. After this command has been sent the watchdog will ignore all future "tickles" from the USB, Ethernet and Digital input. You have the option to stop this command before the reset actually occurs and revert back to normal operation.

**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 – 16 bit delay value in seconds before reset occurrs.
pGetTime – pointer to 16 bit count of time remaining. If there is not a Reset Pending this value will return as zero. In case of a write, it will equal what you just wrote.

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 – set this flag to disable the reset if there is still time left.

Note: if both flags are set, the WD_PC_RESET_DIS will be processed last and the command becomes a null operation and a zero time will be returned.

Example:

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

iResetTime = 300 ;            // 300 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 -- Reset will happen in %0d seconds\n",
                iCurTime) ;
}
```

2.21 WD_ GetSetNvEtherAllowed (USB Only)

Up to this point all the prior commands (functions) described in this manual can be accepted by the board through the USB and the Ethernet interface (via the Ethernet interface DLL). Starting here all the remaining functions (commands) are only for the USB interface.

This function allows you to tell the board which functions (commands) should be allowed on the Ethernet side. See section 3.3 for the Structure of bits definition for commands allowed.

This structure value is always stored in the non-volatile memory of the board. The boards are shipped with the whole structure set to ones so that all commands (and future commands) are allowed.

The structure is composed of one 64 bit integer (64 functions/commands). The address of the structure is passed as a byte array of 8 bytes.

WD_STATUS WD_GetSetNvEtherAllowed(WD_HANDLE wdHandle, UINT32 iFlag, UCHAR* pNvCmdAllow, UCHAR* pNvCurCmdAllow)

Parameters:

wdHandle – Handle of the device from WD_Open().
iFlag – flag to enable writing the non-volatile memory codes
pNvCmdAllow – pointer to a 8 byte (char) array to be written if the flag is set.
pNvCurCmdAllow – pointer to a 8 byte (char) array retrieve the current non-volatile allow bits. 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_ETH_ALLOW_WREN – this flag must be set to store the new code value. If it is clear then the function only returns the current stored value in non-volatile memory

Example:

```
WD_HANDLE wdHandle ;
WD_STATUS wdStatus ;
UINT32 iFlag;
UINT64 *sp ;
WD_E_COMMANDS_ALLOWED_1 structEthAllow , structCurEthAllow;

iFlag = WD_ETH_ALLOW_WREN ;       // enable write to non-volatile
sp = (UINT64*)&structEthAllow ;
*sp = 0xffffffffffffffff ;       // enable all
structEthAllow.E_Allow_WD_GetTempTickle = 0 ;   // not allow a tickle
wdStatus = WD_GetSetEtherAllowed(wdHandle, iFlag,
            (UCHAR*)&structEthAllow, (UCHAR*)&structCurEthAllow);
if(wdStatus == WD_OK)
{
    printf("-- ETH ALLOW -- Allow bits write OK - Bits =
                 %016I64X\n", structCurEthAllow) ;
}
```

2.22 WD_ SetIpAddresses (USB Only)

This function allows you to set a new IP address for the Ethernet interface. It also allows you to store a fixed set of IP addresses in the non-volatile memory on the board and use the DIP Switch option to make them active at power up. This function will allow you to set IP addresses and enable the Ethernet port even if the Ethernet Mode switches are both off for: DISABLED.

NOTE: This command (function) will not work over the Ethernet interface. The addresses can only be set through the USB interface.

This function will allow any values for the IP, Subnet, and Gateway addresses. It is up to you to make sure they are valid.

These byte arrays of addresses should be in Network order – not Intel (PC) reversed.

**WD_STATUS WD_SetIpAddresses(WD_HANDLE wdHandle, UINT32 iFlag,
UCHAR* pIpAddress, UCHAR* pSubMskAddress,
UCHAR* pGwAddress, UCHAR* pUnUsed)**

Parameters:

wdHandle – Handle of the device from WD_Open().
iFlag – flags to enable activation of new addresses and non-volatile memory write.
pIpAddress – pointer to a 4 byte IP address array to be written.
pSubMskAddress – pointer to a 4 byte Subnet Mask address.
pGwAddress – pointer to a 4 byte Gateway address.
pUnUsed – Unused pointer

Return Value:

WD_OK if successful or a WD error code.

Flag:

WD_IP_ACTIVATE_NOW – if this flag is set the watchdog will reset the Ethernet interface and start using these addresses.

WD_NV_IPADD_WREN – if this flag is set the addresses will be written to non-volatile memory.

<u>Example:</u>

```
WD_HANDLE wdHandle ;
WD_STATUS wdStatus ;
UCHAR IpAddress[4] = {192, 168, 1, 75} ;
UCHAR SubMskAddress[4] = {255, 255, 255, 0};
UCHAR GwAddress[4] = {192, 168, 1, 1};

iFlag = WD_IP_ACTIVATE_NOW ;      // enable Ethernet now
wdStatus = WD_SetIpAddresses(wdHandle, iFlag, IpAddress,
           SubMskAddress, GwAddress, NULL);
if(wdStatus == WD_OK)
{
    printf("-- IP ADD -- Set to: %0d.%0d.%0d.%0d\n",
      (int)IpAddress[0], (int)IpAddress[1],
      (int)IpAddress[2], (int)IpAddress[3]) ;
}
```

2.23 WD_ GetIpAddresses (USB Only)

This function allows you to get the current active IP address for the Ethernet interface or get the ones stored in the non-volatile memory.

NOTE: This command (function) will not work over the Ethernet interface.

These byte arrays of addresses will be in Network order – not Intel (PC) reversed.

WD_STATUS WD_GetIpAddresses(WD_HANDLE wdHandle, UINT32 iFlag,
UCHAR* pIpAddress, UCHAR* pSubMskAddress,
UCHAR* pGwAddress, UCHAR* pUnUsed)

Parameters:

wdHandle – Handle of the device from WD_Open().
iFlag – flag for read of current or non-volatile memory
pIpAddress – pointer to a 4 byte IP address array to be read.
pSubMskAddress – pointer to a 4 byte Subnet Mask address.
pGwAddress – pointer to a 4 byte Gateway address.
pUnUsed – Unused pointer (set to NULL)

Return Value:

WD_OK if successful or a WD error code.

Flag:

WD_NV_IPADD_RDEN – if this flag is set the addresses will be read from non-volatile memory, otherwise they will be the current in effect.

Example:

```
WD_HANDLE wdHandle ;
WD_STATUS wdStatus ;
UCHAR IpAddress[4] = {0, 0 ,0, 0} ;
UCHAR SubMskAddress[4] = {0, 0, 0, 0};
UCHAR GwAddress[4] = {0, 0, 0, 0};

iFlag = WD_NV_IPADD_RDEN ;      // get the non-volatile
wdStatus = WD_GetIpAddresses(wdHandle, iFlag, IpAddress,
           SubMskAddress, GwAddress, NULL);
if(wdStatus == WD_OK)
{
    printf("\t-- NV IP ADD --  %0d.%0d.%0d.%0d\n", (int)IpAddress[0],
             (int)IpAddress[1],(int)IpAddress[2], int)IpAddress[3]) ;
}
```

2.24 WD_ GetSetNvUdpPortNum (USB Only)

This function allows you to get and set the port number that the board uses for UDP communications on the Ethernet. The watchdog board defaults to listening to the UDP port number 55108. If there is a conflict on your network then pass a new port number in the **iPort** parameter. Pass a value of zero to keep the default or go back to the default. The PC will use a source port number one less then the board. The default then is 55107.

This value is stored in non-volatile memory. If you make a change to the port number the board will automatically restart the UDP socket with the new port number.

NOTE: This command (function) will not work over the Ethernet interface.

WD_STATUS WD_GetSetNvUdpPortNum(WD_HANDLE wdHandle, UINT32 iFlag, UINT32 iPort, UINT32* pCurPort)

Parameters:

wdHandle – Handle of the device from WD_Open().
iFlag – flag for write enable of new port number
iPort – new port number. Set to zero to use default port number 55108.
pCurPort – pointer to return port number in use – may be what you just wrote.

Return Value:

WD_OK if successful or a WD error code.

Flag:

WD_NV_UDP_PORT_WREN – if this flag is set, the value in iPort will be written to non-volatile memory and the UDP socket will be restarted with the new number. If it is clear then the function just returns the current port number.

Example:

```
WD_HANDLE wdHandle ;
WD_STATUS wdStatus ;
UINT32 iFlag, iPort, iCurPort ;

iFlag = WD_NV_UDP_PORT_WREN ;           // write new port
iPort = 65000 ;
wdStatus = WD_GetSetNvUdpPortNum(wdHandle, iFlag, iPort, &iCurPort);
if(wdStatus == WD_OK)
{
    printf("-- UDP Port --  Port set to %0d\n", iCurPort) ;
}
```

2.25 WD_ GetMacAddress (USB Only)

This function gets the MAC address for the Ethernet interface.

NOTE: This command (function) will not work over the Ethernet interface.

The byte array for the address will be in Network order – not Intel (PC) reversed.

WD_STATUS WD_GetMacAddress(WD_HANDLE wdHandle,
CHAR* pMacAddress)

Parameters:

wdHandle – Handle of the device from WD_Open().
pMacAddress – pointer to a 6 byte MAC address array to be read.

Return Value:

WD_OK if successful or a WD error code.

Example:

```
WD_HANDLE wdHandle ;
WD_STATUS wdStatus ;
UCHAR pMacAddress[6] ;

wdStatus = WD_GetMacAddress(wdHandle, pMacAddress) ;
if(wdStatus == WD_OK)
{
    printf("\t-- MAC ADD --  %02X:%02X:%02X:%02X:%02X:%02X\n",
           pMacAddress[0], pMacAddress[1], pMacAddress[2],
           pMacAddress[3], pMacAddress[4], pMacAddress[5]) ;
}
```

2.26 WD_ ResetRebootEthernet (USB Only)

This function allows you to reset the Ethernet hardware interface and leave it disabled, or reset it then reboot to restart.

NOTE: This command (function) will not work over the Ethernet interface.

WD_STATUS WD_ResetRebootEthernet(WD_HANDLE wdHandle,
UINT32 iFlag)

Parameters:

wdHandle – Handle of the device from WD_Open().
iFlag – Flags for reset and reboot

Return Value:

WD_OK if successful or a WD error code.

Flags:

WD_RESET_ETHERNET – if this flag is set the Ethernet hardware will be reset and left disabled unless the REBOOT flag is set.

WD_REBOOT_ETHERNET – if this flag is set the Ethernet hardware will be software reset and then rebooted per the dip switch settings if the following OverRide flags are clear.

WD_OVRR_FIXED_IP – if this flag is set the Ethernet will reboot with the fixed IP address and ignore the dip switches.

WD_OVRR_NVMEM_IP – if this flag is set the Ethernet will reboot with the non-volatile memory IP address and ignore the dip switches. *Make sure you have a valid IP address stored in non-volatile memory!*

WD_OVRR_DHCP_IP – if this flag is set the Ethernet will reboot looking for a DHCP server to provide the IP address and ignore the dip switches.

NOTE: The #defines for the overrides just match the dip switch values. The overrides must be used with the REBOOT flag – by themselves they will do nothing.

<u>Example:</u>

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

iFlag = WD_RESET_ETHERNET ;       // force hardware reset
iFlag |= WD_REBOOT_ETHERNET ;     // then software reset & reboot
iFlag |= WD_OVRR_FIXED_IP ;       // override dip switch
wdStatus = WD_ResetRebootEthernet(wdHandle, iFlag);
if(wdStatus == WD_OK)
{
    printf("-- ETH RST -- Ethernet Reset/Reboot OK\n") ;
}
```

2.27 WD_ GetSetUsbSuspendMode (USB Only)

This function deals with the USB suspend mode which happens when the PC restarts. In the normal mode the board will detect the suspend and re-enter the **POD** (Power-On-Delay) mode and wait for the suspend to finish. This allows you to reset the computer with the windows option without disabling the board first or having the board inadvertently reset the PC. If the suspend lasts longer than 30 seconds the board will reset the computer assuming it locked up during reboot. While the suspend is active the board will flash both the USB-COMM & ETH-COMM LEDs.

This function provides two non-volatile memory options. One to ignore suspend completely and the other to change the suspend wait time. The time can be between 1 and 255 (0x01 – 0xff) seconds. A value of zero returns the board back to the 30 Seconds default.

NOTE: This command (function) will not work over the Ethernet interface.

WD_STATUS WD_GetSetUsbSuspendMode(WD_HANDLE wdHandle,
UINT32 iFlag, UINT32 iSuspTime,
UINT32* pCurFlag, UINT32* pCurSuspTime)

Parameters:

wdHandle – Handle of the device from WD_Open().
iFlag – Flags for disable and time set.
ISuspTime – the new suspend wait time value.
pCurFlag – pointer to get the SUSPEND_IGNORE flag status back.
pCurSuspTime – pointer to get back the current suspend wait time. If the return is zero then the board is using the default of 30 seconds.

Return Value:

WD_OK if successful or a WD error code.

Flags:

WD_SUSPEND_IGNORE – if this flag is set the USB board will ignore the USB suspend signal and continue its current operation. This flag can be returned if enabled.
WN_NV_IGNORE_WREN – flag must be set to store the ignore status. If clear then the board just returns the current status.
WN_NV_SUSP_TIME_WREN – if this flag is set the new time value will be written to non-volatile memory. If the time is zero and this flag is set, the board sets the 30 second default.

Example:

```
WD_HANDLE wdHandle ;
WD_STATUS wdStatus ;
UINT32 iFlag, iSuspTime, iCurFlag, iCurSuspTime;

iFlag = WN_NV_SUSP_TIME_WREN ;   // write suspend time
iSuspTime =  45 ;                // new time
iFlag |= WN_NV_IGNORE_WREN ;     // test only
iFlag |= WD_SUSPEND_IGNORE ;     // ignore suspend
wdStatus = WD_GetSetUsbSuspendMode(wdHandle, iFlag, iSuspTime,
                &iCurFlag, &iCurSuspTime) ;
if(wdStatus == WD_OK)
{
    printf("-- USB MODE -- New Suspend Timer = %d\n",
                      iCurSuspTime) ;
    if(iCurFlag & WD_SUSPEND_IGNORE)
        printf("-- USB MODE -- Suspend Ignored\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 - the watchdog is armed but the POD Dip Switch was set to wait for first "tickle". In this case the countdown timer has not started yet. Use the function (command) WD_ GetTempTickle to "tickle" the board and clear this status.

WD_STAT_CMD_DISABLED - the watchdog has been disabled by command.

WD_STAT_RESET_PEND – set by the EnableDisablePcReset function to show that a command to reset the PC has been issued with a delay time.

WD_STAT_ETH_ENABLED – shows that the Ethernet IC has been configured and enabled. In case of DHCP, an IP address may not have been assigned.

WD_STAT_ETH_IP_SET – shows that the Ethernet interface now has a valid IP address.

3.2 Diagnostic Flags

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

WD_DIAG_NVMEM_OK – This bit should <u>always</u> be set. If it is clear the non-volatile memory IC on the board has failed. Board should not be used and should be returned for repair.

WD_DIAG_ETHER_OK – This bit should <u>always</u> be set. If it is clear the Ethernet IC on the board has failed. Board should not be used and should be returned for repair.

WD_DIAG_NV_CORRUPTED – This bit should <u>never</u> be set. If it is set then the Non-Volatile memory data has been corrupted and failed a checksum test. Contact Berkshire about trying to recover the memory since the Ethernet MAC address is stored there.

WD_DIAG_MAC_INVALID – This bit should <u>never</u> be set. If it is set then the board has found a problem with the MAC address and will not start the Ethernet interface. Contact Berkshire about trying to recover the MAC address.

WD_DIAG_NV_WRITE_FAIL – This flag will be set if there is a failure writing to the non-volatile memory. The IC and the firmware have built in safeguards to prevent bad writes. If the firmware detects any problems – including a low voltage situation - it will not write the data. This is not a "sticky" flag. If the next write attempt is OK this flag will be cleared.

WD_DIAG_ARP_ERROR – If this bit is set the board has found another device on the network with the same IP address. The Ethernet interface will be reset and left inactive. You will need to correct the problem and then call the Ethernet Reset/Reboot function.

3.3 Commands Allowed on Ethernet

This is the structure for the commands allowed bits to enable commands to be processed from the Ethernet port. Obviously you could also do these with #defines if you are careful to avoid duplicates. A bit set to one (1) means the command is enabled.

```
typedef struct{
  UINT64 E_Allow_WD_GetDeviceInfo          :1;     // Bit 0
  UINT64 E_Allow_WD_GetTempTickle          :1;     // Bit 1
  UINT64 E_Allow_WD_SetPowerOnDlyTimes     :1;     // Bit 2
  UINT64 E_Allow_WD_GetPowerOnDlyTimes     :1;     // Bit 3
  UINT64 E_Allow_WD_SetWdogTimes           :1;     // Bit 4
  UINT64 E_Allow_WD_GetWdogTimes           :1;     // Bit 5
  UINT64 E_Allow_WD_GetAnalogDigitalIn     :1;     // Bit 6
  UINT64 E_Allow_WD_GetSetAuxRelay         :1;     // Bit 7
  UINT64 E_Allow_WD_EnableDisable          :1;     // Bit 8
  UINT64 E_Allow_WD_GetResetCount          :1;     // Bit 9
  UINT64 E_Allow_WD_GetSetNvTempOffset     :1;     // Bit 10
  UINT64 E_Allow_WD_SetBuzzer              :1;     // Bit 11
  UINT64 E_Allow_WD_GetBuzzer              :1;     // Bit 12
  UINT64 E_Allow_WD_GetSetNvRelayPulse     :1;     // Bit 13
  UINT64 E_Allow_WD_GetSetNvUsbUserCode    :1;     // Bit 14
  UINT64 E_Allow_WD_EnableDisablePcReset   :1;     // Bit 15
} WD_E_COMMANDS_ALLOWED_1 ;
```