**DYNAMIC ENGINEERING**
150 DuBois, Suite C Santa Cruz, CA 95060
(831) 457-8891
https://www.dyneng.com
sales@dyneng.com
Est. 1988

# PCIeHarpBase & PCIeHarpChan

## Windows 10 WDF Driver Documentation

### Developed with Windows Driver Foundation Ver1.9

Manual Revision 1p1
Corresponding Hardware: Revision 01
10-2019-1601
FLASH revision 1p2

WDF Device Drivers for
PCIe-Harpoon 4-Channel Simulator and
Digital Data-Link Serial Interface

Dynamic Engineering
150 DuBois St., Suite C
Santa Cruz, CA 95060
831-457-8891

# Table of Contents


Introduction ..... 4
Driver Installation ..... 6
Windows 10 Installation ..... 6
IO Controls ..... 8
IOCTL_PCIeHARP_BASE_GET_INFO ..... 8
IOCTL_PCIeHARP_BASE_LOAD_PLL_DATA ..... 9
IOCTL_PCIeHARP_BASE_READ_PLL_DATA ..... 9
IOCTL_PCIeHARP_BASE_SET_MASTER_INT_CONFIG ..... 9
IOCTL_PCIeHARP_BASE_GET_MASTER_INT_CONFIG ..... 9
IOCTL_PCIeHARP_BASE_BRIDGE_RECONFIG ..... 10
IOCTL_PCIeHARP_BASE_SET_GENERIC ..... 10
IOCTL_PCIeHARP_BASE_GET_GENERIC ..... 10
IOCTL_PCIeHARP_CHAN_GET_INFO ..... 11
IOCTL_PCIeHARP_CHAN_SET_CONFIG ..... 11
IOCTL_PCIeHARP_CHAN_GET_CONFIG ..... 11
IOCTL_PCIeHARP_CHAN_SET_STATUS ..... 11
IOCTL_PCIeHARP_CHAN_GET_STATUS ..... 11
IOCTL_PCIeHARP_CHAN_REGISTER_EVENT ..... 12
IOCTL_PCIeHARP_CHAN_SET_MASTER_INT_CONFIG ..... 12
IOCTL_PCIeHARP_CHAN_GET_MASTER_INT_CONFIG ..... 12
IOCTL_PCIeHARP_CHAN_FORCE_INTERRUPT ..... 12
IOCTL_PCIeHARP_CHAN_CLR_FORCE_INTERRUPT ..... 12
IOCTL_PCIeHARP_CHAN_GET_ISR_STATUS ..... 13
IOCTL_PCIeHARP_CHAN_SET_TESTSTART ..... 13
IOCTL_PCIeHARP_CHAN_SET_TXAMTCNT ..... 13
IOCTL_PCIeHARP_CHAN_GET_TXAMTCNT ..... 13
IOCTL_PCIeHARP_CHAN_SET_RXAFLCNT ..... 13
IOCTL_PCIeHARP_CHAN_GET_RXAFLCNT ..... 13
IOCTL_PCIeHARP_CHAN_SET_TXRXDATA ..... 14
IOCTL_PCIeHARP_CHAN_GET_TXRXDATA ..... 14
IOCTL_PCIeHARP_CHAN_SET_CHANSWCNTL ..... 14
IOCTL_PCIeHARP_CHAN_GET_ CHANSWCNTL ..... 14
Warranty and Repair ..... 15
Service Policy ..... 15
Support ..... 15
For Service Contact: ..... 15

## Introduction

The PCIeHarpBase and PCIeHarpChan are Windows device drivers to support PCIe-Harpoon from Dynamic Engineering. This driver was developed with the Windows Driver Foundation version 1.9 (WDF) from Microsoft, specifically the Kernel-Mode Driver Framework (KMDF).

The Driver software package has two parts. The drivers for Windows® 10 OS, and the User Application "UserAp" executable.

The driver is delivered electronically. The files supplied are installed into the client system to allow access to the hardware. The UserAp code is delivered in source form [C] and is for the purpose of providing a reference to using the driver.

UserAp is a stand-alone code set with a simple, and powerful menu plus a series of "tests" that can be run on the installed hardware. Each of the tests execute calls to the driver, pass parameters and structures, and get results back. With the sequence of calls demonstrated, the functions of the hardware are utilized for loop-back testing. The software is used for manufacturing test at Dynamic Engineering.

The test software can be ported to your application to provide a running start. It is recommended to port the Register tests to your application to get started. The tests are simple and will quickly demonstrate the end-to-end operation of your application making calls to the driver and interacting with the hardware.

The menu allows the user to add tests, to run sequences of tests, to run until a failure occurs and stop or to continue, to program a set number of loops to execute and more. The user can add tests to the provided test suite to try out application ideas before committing to your system configuration. In many cases the test configuration will allow faster debugging in a more controlled environment before integrating with the rest of the system. The test suite is designed to accommodate up to 5 boards. The number of boards can be expanded. See Main.c to increase the number of handles.

The reference SW was developed with Visual Studio. The solution can be opened and used directly if you have this tool. If using another tool the source and include files will need to be imported into your project.

The hardware manual defines the pinout, the bitmaps and detailed configurations for each feature of the design. The driver handles all aspects of interacting with the hardware. For added explanations about what some of the driver functions do, please refer to the hardware manual.

We strive to make a useable product. If you have suggestions for extended features, special calls for particular set-ups or whatever please share them with us.

When PCIe-Harpoon is recognized by the system, the base driver will start which will create a device object for the board. The Base driver will launch 4 copies of the Channel driver. If more than one board is found additional copies of the driver are loaded. From within the PCIe-Harpoon driver the user can access the switch and device number information to determine the specific device being accessed when more than one is installed.

The reference software application has a loop to check for devices and allows selection of the device to be used. The message is suppressed when only 1 device is found.

IO Control calls (IOCTLs) are used to configure the board and read status. Read and Write calls are used to move data in and out of the device.

**Note**

This documentation will provide information about all calls made to the drivers, and how the drivers interact with the device for each of these calls. For more detailed information on the hardware implementation, refer to the PCIe-Harpoon user manual (also referred to as the hardware manual).

## Driver Installation

There are several files provided in each driver package. These files include PCIeHarpBase.sys, PCIeHarpBase.cat, PCIeHarpBase.inf for the base and PCIeHarpChan.sys, PCIeHarpChan.cat, PCIeHarpChan.inf for the channels. Copy to a folder on your target computer or memory device [Flash Drive].

PCIeHarpChanPublic.h and PCIeHarpBasePublic.h are C header files that define the Application Program Interface (API) to the driver. These files are required at compile time by any application that wishes to interface with the driver, but are not needed for driver installation. The header files are supplied with UserAp.

## Windows 10 Installation

Copy the supplied system files to a folder of your choice.

With the hardware installed, power-on the host computer.

- Open the ***Device Manager*** from the control panel.
- Under ***Other devices*** there should be an item for each module installed.
- Right-click on the first device and select ***Update Driver Software***.
- Insert the removable memory device prepared above if necessary.
- Select ***Browse my computer for driver software***.
- Select ***Browse*** and navigate to the memory device or other location prepared above.
- Select ***Next***. The PCIeHarpBase device driver should now be installed.
- Select ***Close*** to close the update window.
- Right-click on the remaining Channel icons and repeat the above procedure as necessary.
  PCIeHarpChan will be installed for the channels.

***** If the devices are not displayed, click on the ***Scan for hardware changes*** icon on the Device Manager tool-bar.

## Driver Startup

Once the driver has been installed it will start automatically when the system recognizes the hardware. Handles can be opened to a specific board by using the CreateFile() function call and passing in the device name obtained from the system. The interfaces to the device and I/O channels are identified using globally unique identifiers (GUIDs), which are defined within the driver files.

See an example of opening handles for device *devNum* in main.c of the UserAp.

## IO Controls

The driver uses IO Control calls (IOCTLs) to configure the device. IOCTLs refer to a single Device Object, which controls a single board. IOCTLs are called using the function DeviceIoControl() (see below), and passing in the handle to the device opened with CreateFile() (see above). IOCTLs generally have input parameters, output parameters, or both. Often a custom structure is used.

```
BOOL DeviceIoControl(
  HANDLE        hDevice,          // Handle opened with CreateFile()
  DWORD         dwIoControlCode, // Control code defined in API header file
  LPVOID        lpInBuffer,       // Pointer to input parameter
  DWORD         nInBufferSize,    // Size of input parameter
  LPVOID        lpOutBuffer,      // Pointer to output parameter
  DWORD         nOutBufferSize,   // Size of output parameter
  LPDWORD       lpBytesReturned, // Pointer to return length parameter
  LPOVERLAPPED lpOverlapped,     // Optional pointer to overlapped structure
);                                //   used for asynchronous I/O
```

### IOCTL_PCIeHARP_BASE_GET_INFO

***Function:*** Returns the Driver version, Xilinx design revision, PLL device ID, User Switch value, and Instance number.
***Input:*** None
***Output:*** PCIeHARP_BASE_DRIVER_DEVICE_INFO structure
***Notes:*** The PLL device ID is used by the driver to communicate with the onboard PLL over its I2C serial bus, Switch value is the current setting of the onboard dipswitch that has been selected by the User (see the board silk screen for bit position and polarity). Instance number is the zero-based device number. See PCIeHarpBasePublic.h for the definition of PCIeHARP_BASE_DRIVER_DEVICE_INFO. See PCIeHarpBasePublic.h for the definition of the structure.

### IOCTL_PCIeHARP_BASE_LOAD_PLL_DATA

***Function:*** Loads the internal registers of the PLL.
***Input:*** PCIeHARP_BASE_PLL_DATA structure
***Output:*** None
***Notes:*** The PCIeHARP_BASE_PLL_DATA structure has only one field: Data – an array of 40 bytes containing the PLL register data to write to the PLL device. During the driver initialization, the PLL is loaded with default data that sets the 8x clock to 800 kHz to provide a data-rate of 100 Kbits/second. This call is only needed if some other data-rate is desired. See PCIeHarpBasePublic.h for the definition of the structure.

### IOCTL_PCIeHARP_BASE_READ_PLL_DATA

***Function:*** Returns the contents of the PLL's internal registers
***Input:*** None
***Output:*** PCIeHARP_BASE_PLL_DATA structure
***Notes:*** The register data is output in the PCIeHARP_BASE_PLL_DATA structure in an array of 40 bytes. See PCIeHarpBasePublic.h for the definition of the structure.

### IOCTL_PCIeHARP_BASE_SET_MASTER_INT_CONFIG

***Function:*** Set or clear the Master Interrupt Enable.
***Input:*** PCIeHARP_BASE_MASTER_INT_CONFIG structure
***Output:*** None
***Notes:*** The PCIeHARP_BASE_MASTER_INT_CONFIG structure has only two fields: MasterIntEn, and Port Interrupt Status. The Status bits are read only. See PCIeHarpBasePublic.h for the definition of the structure.

### IOCTL_PCIeHARP_BASE_GET_MASTER_INT_CONFIG

***Function:*** Returns the contents of the Master Interrupt Register
***Input:*** None
***Output:*** PCIeHARP_BASE_MASTER_INT_CONFIG structure
***Notes:*** The state of Master Interrupt Enable along with the state of the potential interrupters is returned. MasterIntEn = TRUE to be able to cause an interrupt to the system. Port Interrupts are ANDed with MasterIntEn. Port interrupt status can be used to poll if desired. See PCIeHarpBasePublic.h for the definition of the structure.

### IOCTL_PCIeHARP_BASE_BRIDGE_RECONFIG

***Function:*** Reprogram Bridge with enhanced settings for better DMA performance.
***Input:*** None
***Output:*** None
***Notes:*** PCIe-Harpoon does not currently support DMA. This IOCTL is not currently required for use.

### IOCTL_PCIeHARP_BASE_SET_GENERIC

***Function:*** Generic write function, any address any data.
***Input:*** PCIeHARP_BASE_AD structure
***Output:*** None
***Notes:*** The PCIeHARP_BASE_AD structure has fields: Address and Data. Be careful to stay within the address range of the Harpoon. Normally other IOCTLs are used. If you prefer a flat design this IOCTL can access all ports as a flat design. See PCIeHarpBasePublic.h for the definition of the structure.

### IOCTL_PCIeHARP_BASE_GET_GENERIC

***Function:*** Generic read function, any address
***Input:*** None
***Output:*** PCIeHARP_BASE_AD structure
***Notes:*** Set the address, returns with the data from that address. Be careful to stay within the address range of the Harpoon. Normally other IOCTLs are used. If you prefer a flat design this IOCTL can access all ports as a flat design. See PCIeHarpBasePublic.h for the definition of the structure.

### IOCTL_PCIeHARP_CHAN_GET_INFO

***Function:*** Returns the Driver version and Instance number.
***Input:*** None
***Output:*** PCIeHARP_CHAN_DRIVER_DEVICE_INFO structure
***Notes:*** Instance number is the zero-based device number. See PCIeHarpChanPublic.h for the definition of the structure.

### IOCTL_PCIeHARP_CHAN_SET_CONFIG

***Function:*** Configures the channel control register for an I/O channel on the PCI-Harpoon.
***Input:*** ULONG
***Output:*** None
***Notes:*** Sets the configuration parameters for the channel interface, including state-machine enables, interrupt mode, termination enables and the receive data invert (used for testing data loop-back). See the hardware manual for detailed descriptions of the control parameter functions.

### IOCTL_PCIeHARP_CHAN_GET_CONFIG

***Function:*** Returns the configuration of the channel control register.
***Input:*** None
***Output:*** ULONG
***Notes:*** Returns the fields set in the previous call.

### IOCTL_PCIeHARP_CHAN_SET_STATUS

***Function:*** Clears the latched status bits.
***Input:*** Value of status bits to clear (ULONG)
***Output:*** None
***Notes:*** The bits in STAT_LATCH_MASK will be cleared if they are set high in the input field to this call.

### IOCTL_PCIeHARP_CHAN_GET_STATUS

***Function:*** Returns the value of the channel status register.
***Input:*** None
***Output:*** ULONG
***Notes:*** The interrupt status bits will be latched even if they are not enabled, but only the enabled interrupts will cause a system interrupt to occur.

### IOCTL_PCIeHARP_CHAN_REGISTER_EVENT

***Function:*** Registers an event to be signaled when a channel interrupt occurs.
***Input:*** Handle to the Event object
***Output:*** None
***Notes:*** The caller creates an event with CreateEvent() and supplies the handle returned from that call as the input to this IOCTL. The driver then obtains a system pointer to the event and signals the event when a user interrupt is serviced. The user interrupt service routine waits on this event, allowing it to respond to the interrupt. See interrupt.c for an example using the Force Interrupt function.

### IOCTL_PCIeHARP_CHAN_SET_MASTER_INT_CONFIG

***Function:*** Set or clear the Master Interrupt Enable.
***Input:*** PCIeHARP_CHAN_MASTER_INT_CONFIG structure
***Output:*** None
***Notes:*** See PCIeHarpChanPublic.h for the definition of the structure.

### IOCTL_PCIeHARP_CHAN_GET_MASTER_INT_CONFIG

***Function:*** Returns the contents of the Master Interrupt Register
***Input:*** None
***Output:*** PCIeHARP_CHAN_MASTER_INT_CONFIG structure
***Notes:*** The state of Master Interrupt Enable is returned. MasterInEn is ANDed with the latched Channel Status to create the Port/Channel level interrupt. See PCIeHarpChanPublic.h for the definition of the structure.

### IOCTL_PCIeHARP_CHAN_FORCE_INTERRUPT

***Function:*** Causes a channel interrupt to occur.
***Input:*** None
***Output:*** None
***Notes:*** Causes an interrupt to be asserted at the port level as long as the master interrupt (for the port) is enabled. This IOCTL is used for development, to test interrupt processing. To create an interrupt at the system level MasterInEn in the Base must also be enabled.

### IOCTL_PCIeHARP_CHAN_CLR_FORCE_INTERRUPT

***Function:*** Clears the Force Interrupt bit
***Input:*** None
***Output:*** None
***Notes:*** The ISR will disable via the MasterIntEn leaving the cause of the interrupt alone. Use this call to clear the Force Interrupt bit.

**IOCTL_PCIeHARP_CHAN_GET_ISR_STATUS**

***Function:*** Returns the interrupt status that was read in the ISR from the channel's last user interrupt.
***Input:*** None
***Output:*** Interrupt status value (unsigned long integer).
***Notes:*** Returns the value of the status register that was read in the interrupt service routine of the last interrupt serviced. The Interrupt Status is cleared in the ISR after storing into this field. Reading the standard status register will return the current value of the status register with the bits cleared unless another interrupt has be requested.

**IOCTL_PCIeHARP_CHAN_SET_TESTSTART**

***Function:*** Explicitly controls the Test Enable and Test Clock outputs.
***Input:*** ULONG
***Output:*** None
***Notes:*** See PCIeHarpChanPublic.h for the definitions of the bits. Set the Auto Start bit for a HW controlled transfer. Use the discrete clock and enable for SW controlled operation. Used when PCIe-Harpoon is acting as a bus master. Not used with an external master.

**IOCTL_PCIeHARP_CHAN_SET_TXAMTCNT**

**IOCTL_PCIeHARP_CHAN_GET_TXAMTCNT**

**IOCTL_PCIeHARP_CHAN_SET_RXAFLCNT**

**IOCTL_PCIeHARP_CHAN_GET_RXAFLCNT**

***Function:*** Spare registers for SW test and BIT purposes
***Input:*** ULONG
***Output:*** ULONG
***Notes:*** 16 bit registers on LW boundaries. Set writes, Get reads. See Register.c for an example.

**IOCTL_PCIeHARP_CHAN_SET_TXRXDATA**

***Function:*** Writes the 17-bit data-word (16-bit data plus parity) to the shift register.
***Input:*** Transmit data value (unsigned long integer)
***Output:*** None
***Notes:*** State machine detects data written into holding register and moves into the output SR when Tx is enabled and clock starts. TX interrupt can be used to know when safe to load next data.

**IOCTL_PCIeHARP_CHAN_GET_TXRXDATA**

***Function:*** Reads the 17-bit data word from the shift register.
***Input:*** None
***Output:*** Receive data value (unsigned long integer)
***Notes:*** RX Interrupt when set indicates data is ready to read from register.

**IOCTL_PCIeHARP_CHAN_SET_CHANSWCNTL**

***Function:*** Controls the opto-isolated switches that control the 28-volt outputs and returns.
***Input:*** ULONG
***Output:*** None
***Notes:*** There are seven +28V switches and four ground switches. See PCIeHarpChanPublic.h for the definitions of the bits. More information in the HW manual.

**IOCTL_PCIeHARP_CHAN_GET_ CHANSWCNTL**

***Function:*** Returns the bits set in the previous call.
***Input:*** None
***Output:*** ULONG
***Notes:***

# Warranty and Repair

Please refer to the warranty page on our website for the current warranty offered and options.
http://www.dyneng.com/warranty.html.

## Service Policy

The driver has gone through extensive testing, and while not infallible, problems experienced will likely be “cockpit error” rather than an error with the driver. We will work with you to determine the cause of the issue. If the effort is more than a quick conversation, we will offer a support contract. We can write updates to the driver to add features, create middleware etc.

## Support

The software described in this manual is provided at no cost to clients who have purchased the corresponding hardware. Minimal support is included along with the documentation. For help with integration into your project please contact sales@dyneng.com for a support contract. Several options are available. With a contract in place Dynamic Engineers can help with system debugging, special software development, or whatever you need to get going.

## For Service Contact:

Customer Service Department
Dynamic Engineering
150 DuBois Street, Suite C
Santa Cruz, CA 95060
831-457-8891
support@dyneng.com