

DYNAMIC ENGINEERING

150 DuBois St., Suite B&C Santa Cruz, CA 95060

831-457-8891 <https://www.dyneng.com>

sales@dyneng.com

Est. 1988

IP-Parallel-HV & IP-Parallel-HV-Test

Windows 10 WDF Driver Documentation

Developed with Windows Driver Foundation

Windows 10/11 Device Driver for IP-Parallel-HV & IP-Parallel-HV-Test

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

©1988-2024 by Dynamic Engineering.
Trademarks and registered trademarks are owned by their
respective manufactures.

This document contains information of proprietary interest to Dynamic Engineering. It has been supplied in confidence and the recipient, by accepting this material, agrees that the subject matter will not be copied or reproduced, in whole or in part, nor its contents revealed in any manner or to any person except to meet the purpose for which it was delivered.

Dynamic Engineering has made every effort to ensure that this manual is accurate and complete. Still, the company reserves the right to make improvements or changes in the product described in this document at any time and without notice. Furthermore, Dynamic Engineering assumes no liability arising out of the application or use of the device described herein.

The electronic equipment described herein generates, uses, and can radiate radio frequency energy. Operation of this equipment in a residential area is likely to cause radio interference, in which case the user, at his own expense, will be required to take whatever measures may be required to correct the interference.

Dynamic Engineering's products are not authorized for use as critical components in life support devices or systems without the express written approval of the president of Dynamic Engineering.

This product has been designed to operate with IP Module carriers and compatible user-provided equipment. Connection of incompatible hardware is likely to cause serious damage.



Table of Contents

Introduction	5
Driver Installation	5
Windows 10 Installation	6
Driver Startup	6
IO Controls	7
IOCTL_IP_TEST_GET_INFO	7
IOCTL_IP_TEST_SET_IP_CONTROL	8
IOCTL_IP_TEST_GET_IP_STATE	8
IOCTL_IP_TEST_SET_BASE_CONFIG	9
IOCTL_IP_TEST_GET_BASE_CONFIG	9
IOCTL_IP_TEST_SET_TTL_DATA	9
IOCTL_IP_TEST_GET_TTL_DATA	9
IOCTL_IP_TEST_SET_TTL_INT_EN	10
IOCTL_IP_TEST_GET_TTL_INT_EN	10
IOCTL_IP_TEST_SET_TTL_EDGE_LEVEL	10
IOCTL_IP_TEST_GET_TTL_EDGE_LEVEL	10
IOCTL_IP_TEST_SET_TTL_POLARITY	11
IOCTL_IP_TEST_GET_TTL_POLARITY	11
IOCTL_IP_TEST_READ_DIRECT	11
IOCTL_IP_TEST_READ_FILTERED	11
IOCTL_IP_TEST_SET_WR_MEM_OFFSET	11
IOCTL_IP_TEST_SET_RD_MEM_OFFSET	12
IOCTL_IP_TEST_GET_MEM_ADDRESS	12
IOCTL_IP_TEST_REGISTER_EVENT	12
IOCTL_IP_TEST_ENABLE_INTERRUPT	12
IOCTL_IP_TEST_DISABLE_INTERRUPT	13
IOCTL_IP_TEST_FORCE_INTERRUPT	13
IOCTL_IP_TEST_SET_VECTOR	13
IOCTL_IP_TEST_GET_VECTOR	13
IOCTL_IP_TEST_GET_ISR_STATUS	14
IOCTL_IP_TEST_PUT_MEM_DATA	14
IOCTL_IP_TEST_GET_MEM_DATA	14
 WARRANTY AND REPAIR	 15
Service Policy	15
Support	15





Introduction

The IP-Parallel-HV driver is a Windows device driver for the IP-Parallel-HV Industry-pack (IP) module from Dynamic Engineering. This driver was developed with the Windows Driver Foundation version (WDF) from Microsoft, specifically the Kernel-Mode Driver Framework (KMDF). With this release the “Test” and Standard modules have been combined into one design.

IP-Parallel-HV is capable of providing multiple protocols. The standard version provides 24 uncommitted outputs and 24 uncommitted inputs. With revision 5 HW the outputs utilize FETs to provide a high voltage low side switch. The on-board regulator provides a 6.5V standard reference. Other voltages can be supplied up to 12V. An external supply can be used when desired or if voltages above 12V need to be generated. Each input channel has a resistor and Zener diode to scale the input voltage back to “TTL” levels.

When the IP carrier driver is started it will enumerate the carrier’s IP bus by reading the ID proms of installed IPs. If the IP module driver has been previously installed, it will be loaded and a Device Object will be created for each installed IP. A separate handle to each IP module can be obtained using CreateFile() calls. IO Control calls (IOCTLs) are used to configure the IP module and read the module’s status. WriteFile() and ReadFile() calls are used to move blocks of data in and out of the IP module.

This documentation will provide information about all calls made to the driver, and how the driver interacts with the hardware for each of these calls. For more detailed information on the hardware implementation, refer to the IP-Parallel-HV user manual (also referred to as the hardware manual). In addition, the “UserAp” reference software package demonstrates the use of the IOCTLs to perform various tests of the HW. The reference software can be modified to suite your purposes.

Driver Installation

There are several files provided in each driver package. These files include IpTest.sys, IpTestPublic.h, IpPublic.h, IpTest.inf and IpTest.cat.

IpTestPublic.h and IpPublic.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. Please note: IpPublic.h is common to all Dynamic Engineering IP Module drivers and is in the root of the IpDriver Reference SW folder.

Note: Other IP module drivers are included in the package since they were all signed together and must be present to validate the digital signature. These other IP module driver files must be present when the IpTest driver is installed, to verify the digital signature in IpDevices.cat, otherwise they can be ignored.



Warning: The appropriate IP carrier driver must be installed before any IP modules can be detected by the system.

Windows 10 Installation

Copy IpTest.inf, IpTest.cat, IpTest.sys and the other IP module drivers to a removable memory device or other accessible location as preferred.

With the IP 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 IP module installed on the IP carrier. The label for a module installed in the first slot of the first PCIe3IP carrier would read **PcieCar0 IP Slot A***.
- 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 IpTest device driver should now be installed.
- Select **Close** to close the update window.
- Right-click on the remaining IP slot icons and repeat the above procedure as necessary.

This process must be completed for each new IP device that is installed.

* If no IP devices are displayed, check to see that an IP Carrier Device is present in the Device Manager and click on the **Scan for hardware changes** icon on the tool-bar or select it in the Action menu.

Driver Startup

Once the driver has been installed it will start automatically when the system recognizes the hardware.

A handle can be opened to a specific board by using the CreateFile() function call and passing in the device name obtained from the system.

The interface to the device is identified using a globally unique identifier (GUID), which is defined in IpTestPublic.h.

The *main.c* file provided with the user test software can be used as an example to show how to obtain a handle to an IP-Parallel-HV device.



IO Controls

The driver uses IO Control calls (IOCTLs) to configure the device. IOCTLs refer to a single Device Object, which controls a single module. 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
```

The IOCTLs defined for the IpTest driver are described below:

IOCTL_IP_TEST_GET_INFO

Function: Returns the driver and firmware revisions, module instance number and location and other information.

Input: None

Output: DRIVER_IP_DEVICE_INFO structure

Notes: This call does not access the hardware, only stored driver parameters. NewIpCntl indicates that the module's carrier has expanded slot control capabilities. See the definition of DRIVER_IP_DEVICE_INFO below.

```
// Driver version and instance/slot information  
typedef struct _DRIVER_IP_DEVICE_INFO {  
    USHORT    DriverRev;  
    USHORT    FirmwareRev;  
    USHORT    InstanceNum;  
    UCHAR     CarrierSwitch; // 0..0xFF  
    UCHAR     CarrierSlotNum; // 0..7 -> IP slots A, B, C, D, E, F, G or H  
    BOOLEAN   NewIpCntl;     // New IP slot control bits  
    WCHAR     LocationString[IP_LOC_STRING_SIZE];  
} DRIVER_IP_DEVICE_INFO, *PDRIVER_IP_DEVICE_INFO;
```

IOCTL_IP_TEST_SET_IP_CONTROL

Function: Sets various control parameters for the IP slot the module is installed in.

Input: IP_SLOT_CONTROL structure

Output: None

Notes: Controls the IP clock speed, interrupt enables and data manipulation options for the IP slot that the board occupies. See the definition of IP_SLOT_CONTROL below. For more information refer to the IP carrier hardware manual.

```
typedef struct _IP_SLOT_CONTROL {
    BOOLEAN    Clock32Sel;
    BOOLEAN    ClockDis;
    BOOLEAN    ByteSwap;
    BOOLEAN    WordSwap;
    BOOLEAN    WrIncDis;
    BOOLEAN    RdIncDis;
    UCHAR     WrWordSel;
    UCHAR     RdWordSel;
    BOOLEAN    BsErrTmOutSel;
    BOOLEAN    ActCountEn;
} IP_SLOT_CONTROL, *PIP_SLOT_CONTROL;
```

IOCTL_IP_TEST_GET_IP_STATE

Function: Returns control/status information for the IP slot the module is installed in.

Input: None

Output: IP_SLOT_STATE structure

Notes: Returns the slot control parameters set in the previous call as well as status information for the IP slot that the board occupies. See the definition of IP_SLOT_STATE below.

```
typedef struct _IP_SLOT_STATE {
    BOOLEAN    Clock32Sel;
    BOOLEAN    ClockDis;
    BOOLEAN    ByteSwap;
    BOOLEAN    WordSwap;
    BOOLEAN    WrIncDis;
    BOOLEAN    RdIncDis;
    UCHAR     WrWordSel;
    UCHAR     RdWordSel;
    BOOLEAN    BsErrTmOutSel;
    BOOLEAN    ActCountEn;
    // Slot Status
    BOOLEAN    IpInt0En;
    BOOLEAN    IpInt1En;
    BOOLEAN    IpBusErrIntEn;
    BOOLEAN    IpInt0Actv;
    BOOLEAN    IpInt1Actv;
    BOOLEAN    IpBusError;
    BOOLEAN    IpForceInt;
    BOOLEAN    WrBusError;
    BOOLEAN    RdBusError;
} IP_SLOT_STATE, *PIP_SLOT_STATE;
```


IOCTL_IP_TEST_SET_BASE_CONFIG

Function: Sets configuration parameters in the IP base control register.

Input: IP_TEST_BASE_CONFIG structure

Output: None

Notes: Controls the output behavior of the control register values. See the definition of OUT_SEL and IP_TEST_BASE_CONFIG below. Detailed definition can be found in the 'Base_CNTL' section under Register Definitions in the Hardware manual.

```
typedef struct _IP_TEST_BASE_CONFIG {  
    BOOLEAN OutRegEnable;  
} IP_TEST_BASE_CONFIG, *PIP_TEST_BASE_CONFIG;
```

IOCTL_IP_TEST_GET_BASE_CONFIG

Function: Returns the configuration of the IP base control register.

Input: None

Output: IP_TEST_BASE_CONFIG structure

Notes: Returns the values set in the previous call.

IOCTL_IP_TEST_SET_TTL_DATA

Function: Sets the value of the 24 data outputs on the board.

Input: Unsigned long integer

Output: None

Notes: The value of each of the 24 TTL data output bits is determined by the corresponding bit in the input word for this call. The Output lines of IP-PAR-HV module are equipped with FETs and pull-ups. When the drivers are configured to output a low level, the external line will be driven low. When set to a high level the reference voltage or other devices on the line will set the level. Called TTL as the control side is TTL. IO side can be up to 50V.

IOCTL_IP_TEST_GET_TTL_DATA

Function: Returns the state of the bits in the output data register.

Input: None

Output: Unsigned long integer

Notes: The value returned depends only on the value that was written by IOCTL_IP_TEST_SET_TTL_DATA, not on the voltage level of the external line. To return the Input line voltage levels use IOCTL_IP_TEST_READ_DIRECT.

IOCTL_IP_TEST_SET_TTL_INT_EN

Function: Selects which inputs are latched and can cause an interrupt.

Input: Unsigned long integer

Output: None

Notes: This call defines the mask of which of the 24 input lines will be enabled to cause an interrupt if the specified polarity and edge/level conditions are met. A one in a certain bit position enables the respective input line to be considered. A zero eliminates that line as a candidate for interrupt generation.

IOCTL_IP_TEST_GET_TTL_INT_EN

Function: Returns the interrupt enable values set in the previous call.

Input: None

Output: Unsigned long integer

Notes: The value returned will be equal to the value that was written by the last IOCTL_IP_TEST_SET_TTL_INT_EN call.

IOCTL_IP_TEST_SET_TTL_EDGE_LEVEL

Function: Selects whether an input is edge-sensitive or level sensitive.

Input: Unsigned long integer

Output: None

Notes: Determines whether the interrupt for each of the enabled Input lines responds to a static logic level or a transition between levels. A one in a certain bit position configures the respective input line to be edge-sensitive. A zero configures the line to be level-sensitive. Which level or edge the latch responds to is determined by the IOCTL_IP_TEST_SET_TTL_POLARITY call.

IOCTL_IP_TEST_GET_TTL_EDGE_LEVEL

Function: Returns the interrupt edge/level values set in the previous call.

Input: None

Output: Unsigned long integer

Notes: The value returned will be equal to the value that was written by the last IOCTL_IP_TEST_SET_TTL_EDGE_LEVEL call.

IOCTL_IP_TEST_SET_TTL_POLARITY

Function: Selects whether an input is active high or active low.

Input: Unsigned long integer

Output: None

Notes: Determines the polarity of the level or edge to which the corresponding input line will respond. A one in a certain bit position configures the respective input line to respond to a high level or a rising edge depending on the state of the corresponding edge/level bit. A zero configures the line to respond to a low level or a falling edge.

IOCTL_IP_TEST_GET_TTL_POLARITY

Function: Returns the interrupt polarity values set in the previous call.

Input: None

Output: Unsigned long integer

Notes: The value returned will be equal to the value that was written by the last IOCTL_IP_TEST_SET_TTL_POLARITY call.

IOCTL_IP_TEST_READ_DIRECT

Function: Reads the input data bus directly.

Input: None

Output: Unsigned long integer

Notes: This call reads the raw real-time input data from the Input lines and returns an unsigned long integer corresponding to that value.

IOCTL_IP_TEST_READ_FILTERED

Function: Reads the state of the input data latches.

Input: None

Output: Unsigned long integer

Notes: This call reads the contents of the interrupt latches after the enable mask, edge/level, and polarity bits have been applied. A one means that the specified conditions for that bit have been met. The values are returned in an unsigned long integer. The latched bits are automatically cleared after being read by this call.

IOCTL_IP_TEST_SET_WR_MEM_OFFSET

Function: Specifies the starting offset into the 4K byte RAM for multi-word writes.

Input: Unsigned long integer

Output: None

Notes: This call should be run before a WriteFile() is performed the first time or if the target RAM address has changed since the last time this call was made.

IOCTL_IP_TEST_SET_RD_MEM_OFFSET

Function: Specifies the starting offset into the 4K byte RAM for multi-word reads.

Input: Unsigned long integer

Output: None

Notes: This call should be run before a ReadFile() is performed the first time or if the target RAM address has changed since the last time this call was made.

IOCTL_IP_TEST_GET_MEM_ADDRESS

Function: Returns the state of the 22 IP module address lines when the last MEM select occurred.

Input: None

Output: Unsigned long integer

Notes: When the 8 Mbyte IP MEM space is accessed, the lower six address bits are specified with the six IP address lines and the remaining upper 16 address bits are written to the 16 IP data bits. The address bits are concatenated into a 22-bit address that accesses the 4 Mbyte by 16-bit IP MEM space. The byte selects specify which byte(s) are accessed. The 22-bit address is returned by this call for test purposes.

IOCTL_IP_TEST_REGISTER_EVENT

Function: Registers an event to be signaled when an interrupt occurs.

Input: Handle to 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 an interrupt is serviced. The user interrupt service routine waits on this event, allowing it to respond to the interrupt. In order to un-register the event, set the event handle to NULL while making this call.

IOCTL_IP_TEST_ENABLE_INTERRUPT

Function: Sets the master interrupt enable.

Input: None

Output: None

Notes: Sets the master interrupt enable, leaving all other bit values in the base register unchanged. This IOCTL is used in the user interrupt processing function to re-enable the interrupts after they were disabled in the driver ISR. This allows the driver to set the master interrupt enable without knowing the state of the other base configuration bits.

IOCTL_IP_TEST_DISABLE_INTERRUPT

Function: Clears the master interrupt enable.

Input: None

Output: None

Notes: Clears the master interrupt enable, leaving all other bit values in the base register unchanged. This IOCTL is used when interrupt processing is no longer desired.

IOCTL_IP_TEST_FORCE_INTERRUPT

Function: Causes a system interrupt to occur.

Input: None

Output: None

Notes: Causes an interrupt to be asserted on the IP bus using IP interrupt line. This IOCTL is used for development, to test interrupt processing.

IOCTL_IP_TEST_SET_VECTOR

Function: Sets the value of the interrupt vector.

Input: Unsigned character

Output: None

Notes: This value will be driven onto the low byte of the data bus in response to an INT_SEL strobe, which is used in vectored interrupt cycles. This value will be read in the interrupt service routine and stored for future reference. This call accesses the register where that vector is stored.

IOCTL_IP_TEST_GET_VECTOR

Function: Returns the current interrupt vector value.

Input: None

Output: Unsigned character

Notes: This call accesses the register where the interrupt vector is stored so that the vector value can be verified without generating an interrupt.

IOCTL_IP_TEST_GET_ISR_STATUS

Function: Returns the interrupt status and vector read in the last ISR.

Input: None

Output: IP_TEST_INT_STAT structure

Notes: The status contains the contents of the INT_STAT register read in the last ISR execution. Also, if bit 12 is set, it indicates that a bus error occurred for this IP slot.

```
// Interrupt status and vector
typedef struct _IP_TEST_INT_STAT {
    USHORT    InterruptStatus;
    USHORT    InterruptVector;
    ULONG     FilteredData;
} IP_TEST_INT_STAT, *PIP_TEST_INT_STAT;
```

IOCTL_IP_TEST_PUT_MEM_DATA

Function: Writes a single 16-bit word to a specific address offset in the IP MEM space.

Input: IP_TEST_DATA_WRITE structure

Output: None

Notes: The address can be any value below 4,194,304 (0x400000) and the data can be any 16-bit value. This address covers the entire 8 Mbyte IP MEM space and although there is only a 4 Kbyte RAM implemented, the call succeeds for the entire IP MEM space as if the entire 8 Mbyte memory was populated. This behavior allows the Address Bit read-back function to test all of the address bits.

```
typedef struct _IP_TEST_DATA_WRITE {
    ULONG     Address;
    USHORT    Data;
} IP_TEST_DATA_WRITE, *PIP_TEST_DATA_WRITE;
```

IOCTL_IP_TEST_GET_MEM_DATA

Function: Returns the 16-bit data word read at the address offset.

Input: Address offset (unsigned long integer)

Output: Data (unsigned short integer)

Notes: This call reads the MEM space location at the input address offset and returns the 16-bit data word. Reads above the 4 Kbyte address limit of the RAM that is actually implemented will result in a bus error.

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

Before returning a product for repair, verify as well as possible that the driver is at fault. The driver has gone through extensive testing, and in most cases it will be “cockpit error” rather than an error with the driver. When you are sure or at least willing to pay to have someone help then call or e-mail and arrange to work with an engineer. We will work with you to determine the cause of the issue.

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 B&C
Santa Cruz, CA 95060
831-457-8891
support@dyneng.com

All information provided is Copyright Dynamic Engineering

