DYNAMIC ENGINEERING
150 DuBois, Suite C
Santa Cruz, CA 95060
(831) 457-8891 **Fax** (831) 457-4793
http://www.dyneng.com
sales@dyneng.com
Est. 1988

IpGeneric

WDF Driver Documentation

Developed with Windows Driver Foundation
Ver1.9

Manual Revision A 7/22/16

IpGeneric
WDF Device Driver for an
Unknown IP Module

Dynamic Engineering
150 DuBois, Suite C
Santa Cruz, CA 95060
831-457-8891
831-457-4793 FAX

Table of Contents

Introduction

IpGeneric is a Win7 device driver for any Industry-Pack (IP) module for which a more specialized driver does not exist. This driver can control any IP module by mapping the IO, MEM, and INT memory spaces so that they can be accessed by driver calls. A separate Device Object controls each IP module, and a separate handle references each Device Object. IO Control calls (IOCTLs) are used to configure the hardware and read status information. ReadFile() and WriteFile() calls are used to transfer blocks of data to and from the MEM space over the IP bus.

Note

This documentation will provide information about all calls made to the driver, and how the driver interacts with the device for each of these calls. For more detailed information on the hardware implementation, refer to the hardware manual for the particular device being used.

Driver Installation

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

There are several files provided in each driver package. These files include IpDevices.inf, IpDevices.cat, WdfCoInstaller01009.dll, IpGeneric.sys and several other IP driver files (Ip*.sys). Also header files that define the driver API including IpPublic.h, IpGenericPublic.h and (Ip*Public.h) files for several other IP module device drivers.

Windows 7 Installation

Copy IpDevices.inf, IpDevices.cat, WdfCoInstaller01009.dll, and all the IP device driver files (*.sys) to a removable memory device, or other accessible location as preferred.

With one or more IP device installed in a supported IP module carrier, power-on the host computer.

- Open the ***Device Manager*** from the control panel.
- Under ***Other devices*** there should be, for each valid IP module installed, a device icon with an index appended carrier device name followed by an IP Slot designation where the module is installed*.
- Right-click on the target 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 location where the appropriate files are stored.
- Select ***Next***. The appropriate IP device driver or *IpGeneric* driver should now be installed.
- Select ***Close*** to close the update window.

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.

IpPublic and IpGenericPublic.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 IpGeneric driver, but they are not needed for driver installation. The device interface identifier (GUID) for the IpGeneric driver is defined in IpGenericPublic.h.

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 module by using the CreateFile() function call and passing in the device name obtained from the system.

See the *main.c* file provided with the user test software for an example of obtaining a device handle to a specific module.

IO Controls

The driver uses IO Control calls (IOCTLs) to configure the device. IOCTLs refer to a single Device Object in the driver, which controls a single board. IOCTLs are called using the Win32 function DeviceIoControl(), and passing in the handle to the device opened with CreateFile(). 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 in the IpGeneric driver are described below:

IOCTL_IP_GENERIC_GET_INFO

Function: Returns the current driver revision, instance number, module location and other carrier information.
Input: None
Output: DRIVER_IP_DEVICE_INFO structure
Notes: This call does not access the hardware, only driver parameters. CarrierSwitch returns the value of the 8-position IP carrier dip-switch when this IP was enumerated. FirmwareRev is not valid for this driver. 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_GENERIC_SET_IP_CONTROL

Function: Sets the control configuration of the moduleôs IP slot.
Input: IP_SLOT_CONTROL structure
Output: None
Notes: Specifies the IP clock speed, data access and other control parameters for the IP slot that the board occupies. See the definition of IP_SLOT_CONTROL below.

```
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_GENERIC_GET_IP_STATE

Function: Returns the control configuration of the moduleôs IP slot plus interrupt and bus error status.
Input: None
Output: IP_SLOT_STATE structure
Notes: Returns the slot control configuration from the previous call along with interrupt enable and activity information. 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_GENERIC_SET_WR_MEM_OFFSET

Function: Sets the address offset for block write operations.
Input: Unsigned long integer
Output: None
Notes: Sets the address offset into the IP MEM space that will be used for WriteFile calls.

IOCTL_IP_GENERIC_GET_WR_MEM_OFFSET

Function: Returns the address offset for block write operations.
Input: None
Output: Unsigned long integer
Notes: Returns the address offset into the IP MEM space that will be used for WriteFile calls.

IOCTL_IP_GENERIC_SET_RD_MEM_OFFSET

Function: Sets the address offset for block read operations.
Input: Unsigned long integer
Output: None
Notes: Sets the address offset into the IP MEM space that will be used for ReadFile calls.

IOCTL_IP_GENERIC_GET_RD_MEM_OFFSET

Function: Returns the address offset for block read operations.
Input: None
Output: Unsigned long integer
Notes: Returns the address offset into the IP MEM space that will be used for ReadFile calls.

IOCTL_IP_GENERIC_PUT_DATA

Function: Writes a byte, word or long-word to the IPôs IO or MEM space.
Input: IP_GENERIC_DATA_WRITE structure
Output: None
Notes: This call is used to write data to the IO or MEM space. The structure contains an address [offset] field, a length field (which can be 1, 2, or 4 corresponding to the number of bytes in the target object) and a data field. For this call the space selector, address, length and data fields must all be initialized and the structure passed to the driver which performs the write operation. See for the definition of SPACE_SEL and IP_GENERIC_DATA_WRITE below.

```
typedef enum _SPACE_SEL {
   IO_SPACE,
   MEM_SPACE,
} SPACE_SEL, *PSPACE_SEL;

typedef struct _IP_GENERIC_DATA_WRITE {
   SPACE_SEL   MemIoSelect;
   ULONG       Address;
   UCHAR       Length;
   ULONG       Data;
} IP_GENERIC_DATA_WRITE, *PIP_GENERIC_DATA_WRITE;
```

IOCTL_IP_GENERIC_GET_DATA

Function: Reads a byte, word or long-word from the IPôs IO or MEM space.
Input: IP_GENERIC_DATA_ADDRESS structure
Output: Unsigned long integer
Notes: This call is used to read data from the IO or MEM space. The IP_GENERIC_DATA_ADDRESS structure contains a selector to indicate whether the operation targets IO space or MEM space, an address [offset] field and a length field (which can be 1, 2, or 4 corresponding to the number of bytes in the target object). For this call the space selector, address and length fields must all be initialized and the structure passed to the driver which performs the read operation and returns an unsigned long integer that contains the data that was read. See the definition of IP_GENERIC_DATA_ADDRESS below.

```
// IO data access structures
typedef struct _IP_GENERIC_DATA_ADDRESS {
   SPACE_SEL   MemIoSelect;
   ULONG       Address;
   UCHAR       Length;
} IP_GENERIC_DATA_ADDRESS, *PIP_GENERIC_DATA_ADDRESS;
```

IOCTL_IP_GENERIC_PUT_DATA64

Function: Writes a byte, word, long word or 64-bit word to the IPǒs IO or MEM space.
Input: IP_GENERIC_DATA64_WRITE structure
Output: None
Notes: This call is used to write data to the IO or MEM space. It is only valid in a 64-bit operating environment when the carrier that the IP is mounted on supports 64-bit writes. The IP_GENERIC_DATA64_WRITE structure contains a selector to indicate whether the operation targets IO space or MEM space, an address [offset] field, a length field (which can be 1, 2, 4 or 8 corresponding to the number of bytes in the target object) and a data field. For this call the space selector, address, length, and data fields must all be initialized and the structure is passed to the driver which performs the write operation. See for the definition of IP_GENERIC_DATA64_WRITE below.

```
typedef struct _IP_GENERIC_DATA64_WRITE {
   SPACE_SEL   MemIoSelect;
   ULONG       Address;
   UCHAR       Length;
   ULONG64     Data;
} IP_GENERIC_DATA64_WRITE, *PIP_GENERIC_DATA64_WRITE;
```

IOCTL_IP_GENERIC_GET_DATA64

Function: Reads a byte, word, longword or 64-bit word from the IPǒs IO or MEM space.
Input: IP_GENERIC_DATA_ADDRESS structure
Output: Unsigned long long integer (64-bit)
Notes: This call is used to read data from the IO or MEM space. It is only valid in a 64-bit operating environment when the carrier that the IP is mounted on supports 64-bit reads. The IP_GENERIC_DATA_ADDRESS structure contains a selector to indicate whether the operation targets IO space or MEM space, an address [offset] field and a length field (which can be 1, 2, 4 or 8 corresponding to the number of bytes in the target object). For this call the space selector, address and length fields must be initialized and the structure passed to the driver which performs the read operation and returns an unsigned long long integer that contains the data that was read. See the definition of IP_GENERIC_DATA_ADDRESS below.

```
typedef struct _IP_GENERIC_DATA_ADDRESS {
   SPACE_SEL   MemIoSelect;
   ULONG       Address;
   UCHAR       Length;
} IP_GENERIC_DATA_ADDRESS, *PIP_GENERIC_DATA_ADDRESS;
```

IOCTL_IP_GENERIC_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.

IOCTL_IP_GENERIC_ENABLE_INTERRUPT

Function: Sets the master interrupt enable bits to true.
Input: None
Output: None
Notes: Sets both of the IP slot interrupt enables, leaving all other bit values in the IP slot control register the same. This IOCTL is used in the user interrupt processing function to re-enable the interrupts after they were disabled in the driver interrupt service routine. This allows that function to enable the interrupts without knowing the particulars of the other configuration bits.

IOCTL_IP_GENERIC_DISABLE_INTERRUPT

Function: Sets the master interrupt enable bits to false.
Input: None
Output: None
Notes: Clears both of the IP slot interrupt enables, leaving all other bit values in the IP slot control register the same. This IOCTL is used when interrupt processing is no longer desired.

IOCTL_IP_GENERIC_FORCE_INTERRUPT

Function: Causes a system interrupt to occur.
Input: None
Output: None
Notes: Causes an interrupt to be asserted if the master interrupt for the IP slot is enabled. This IOCTL is used for development, to test interrupt processing.

IOCTL_IP_GENERIC_GET_ISR_STATUS

Function: Returns the interrupt status and interrupt vector.
Input: None
Output: IP_GENERIC_INT_STAT structure
Notes: Returns the interrupt vector and the contents of the interrupt status register that were read in the last ISR call. The IP_GENERIC_INT_STAT structure, returned by this call, contains these two values. See the definition of IP_GENERIC_INT_STAT below.

```
 // Interrupt status and vector
typedef struct _IP_GENERIC_INT_STAT {
   USHORT   InterruptStatus;
   USHORT   InterruptVector;
} IP_GENERIC_INT_STAT, *PIP_GENERIC_INT_STAT;
```

Write

Blocks of data to be written to the IP MEM space can use the WriteFile() call. The user supplies the device handle, a pointer to the buffer containing the data, the number of bytes to write, a pointer to a variable to store the amount of data actually transferred, and an optional pointer to an Overlapped structure for performing asynchronous IO. The number of bytes is checked to see if the transfer length plus the address offset will overrun the end of memory. If this occurs, the length will be reduced accordingly. The driver takes advantage of the carrier's 32-bit double-write/64-bit quad write capability to load two/four IP words with a single PCI/PCIe write until less than four/eight bytes remain in the buffer. If the transfer is not to start at the beginning of the MEM space, the IOCTL_IP_GENERIC_SET_WR_MEM_OFFSET call can be used to specify the start address offset. See Win32 help files for details of the WriteFile() call.

Read

Blocks of data to be read from the IP MEM space can use the ReadFile() call. The user supplies the device handle, a pointer to the buffer to store the data in, the number of bytes to read, a pointer to a variable to store the amount of data actually transferred, and a pointer to an optional Overlapped structure for performing asynchronous IO. The number of bytes is checked to see if the transfer length plus the address offset will overrun the end of memory. If this occurs, the length will be reduced accordingly. The driver takes advantage of the carrier 32-bit double-read/64-bit quad read capability to read two/four IP words with a single PCI/PCIe read until less than four/eight bytes remain to be read. If the transfer is not to start at the beginning of the MEM space, the IOCTL_IP_GENERIC_SET_RD_MEM_OFFSET call can be used to specify the start address offset. See Win32 help files for the details of the ReadFile() call.

Warranty and Repair

Dynamic Engineering warrants this product to be free from defects under normal use and service and in its original, unmodified condition, for a period of one year from the time of purchase. If the product is found to be defective within the terms of this warranty, Dynamic Engineering's sole responsibility shall be to repair, or at Dynamic Engineering's sole option to replace, the defective product.

Dynamic Engineering's warranty of and liability for defective products is limited to that set forth herein. Dynamic Engineering disclaims and excludes all other product warranties and product liability, expressed or implied, including but not limited to any implied warranties of merchantability or fitness for a particular purpose or use, liability for negligence in manufacture or shipment of product, liability for injury to persons or property, or for any incidental or consequential damages.

Dynamic Engineeringõs 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.

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 a ñcockpit errorò rather than an error with the driver. When you are sure or at least willing to pay to have someone help then call the Customer Service Department and arrange to speak with an engineer. We will work with you to determine the cause of the issue. If the issue is one of a defective driver we will correct the problem and provide an updated module(s) to you [no cost]. If the issue is of the customerõs making [anything that is not the driver] the engineering time will be invoiced to the customer. Pre-approval may be required in some cases depending on the customerõs invoicing policy.

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, Suite C
Santa Cruz, CA 95060
(831) 457-8891
(831) 457-4793 fax
support@dyneng.com