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

IP-OptoISO-16

16 Channel Optically Isolated Drivers

Driver Documentation

Developed with Windows Driver Foundation Ver1.9

Manual Revision A
Corresponding Hardware: Revision C
10-2003-0104
FLASH revision A1

IP-OptoISO-16

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

Table of Contents

Introduction

The IP-OptoIso-16 driver is a Windows device driver for the IP-Test Industry-pack (IP) module 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 IP-OptoIso-16 driver package has two parts. The driver is installed into the Windows® OS, and the User Application ñUserAppò executable.

The driver is delivered as installed or executable items to be used directly or indirectly by the user. The UserApp code is delivered in source form [C] and is for the purpose of providing a reference to using the driver.

UserApp is a stand-alone code set with a simple, and powerful menu plus a series of ñtestsò 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 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 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, and while we can guarantee operation we canôt foresee all concepts for client implementation. If you have suggestions for extended features, special calls for particular set-ups or whatever please share them with us, [engineering@dyneng.com] and we will consider and in many cases add them.

IP-OptoIso-16 has a Spartan2 Xilinx FPGA to implement the IP Interface, protocol control and status for the IO. IP-OptoIso-16 is designed to provide optically isolated FET switches suitable for high and low side high voltage switching applications. Additional features include two counter timers. The counters can be used to create periodic interrupts.

When the IP-OptoIso-16 board is recognized by the IP Carrier Driver, the carrier driver will start the IP-OptoIso-16 driver which will create a device object for the board. If more than one is found additional copies of the driver are loaded. The carrier driver will load the info storage register on the IP-OptoIso-16 with the carrier switch setting and the slot number of the IP-OptoIso-16 device. From within the IP-OptoIso-16 driver the user can access the switch and slot information to determine the specific device being accessed when more than one are installed.

The reference software application has a loop to check for devices. The number of devices found, the locations, and device count are printed out at the top of the menu.

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 IP-OptoIso-16 user manual (also referred to as the hardware manual).

Driver Installation

There are several files provided in each driver package. These files include IpOptoIso16.sys, IpOptoIso16Public.h, IpPublic.h, WdfCoInstaller01009.dll, IpDevices.inf and IpDevices.cat.

IpOptoIso16Public.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.

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 IpOptoIso16 driver is installed, to verify the digital signature in IpDevices.cat, otherwise they can be ignored.

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

Windows 7 Installation

Copy IpDevices.inf, IpDevices.cat, WdfCoInstaller01009.dll, IpOptoIso16.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 IpOptoIso16 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.

***** If the [***Carrier] IP Slot [x]*** 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.

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 IpOptoIso16Public.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 IpOptoIso16 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 Win32 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 IpOptoIso16 driver are described below:

IOCTL_IP_OPTO_ISO16_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 moduleȇs 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   FirmwareRevMin;
   USHORT   InstanceNum;
   UCHAR    CarrierSwitch;
   UCHAR    CarrierSlotNum;
   BOOLEAN  NewIpCntl;
   WCHAR    LocationString[IP_LOC_STRING_SIZE];
} DRIVER_IP_DEVICE_INFO, *PDRIVER_IP_DEVICE_INFO;
```

IOCTL_IP_OPTO_ISO16_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_OPTO_ISO16_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_OPTO_ISO16_GET_IP_SIGNATURE

Function: Returns IP module information
Input: None
Output: IP_OPTO_ISO16_SIGNATURE
Notes: See the definition of IP_OPTO_ISO16_SIGNATURE below.

```
typedef struct _IP_OPTO_ISO16_SIGNATURE {
   UCHAR    IpManuf;
   UCHAR    IpModel;
   UCHAR    IpRevision;
   UCHAR    IpCustomer;
   USHORT   IpVersion;
   UCHAR    Slot;
} IP_OPTO_ISO16_SIGNATURE, *PIP_OPTO_ISO16_SIGNATURE;
```

IOCTL_IP_OPTO_ISO16_ENABLE_FET

Function: Sets the enable for FET operation
Input: None
Output: None
Notes: Leaves all other bit values in the base register unchanged. Detailed definition can be found under óip_optoiso_baseôsection under Register Definitions in the Hardware manual.

IOCTL_IP_OPTO_ISO16_DISABLE_FET

Function: Clears the enable for FET operation
Input: None
Output: None
Notes: Leaves all other bit values in the base register unchanged. Detailed definition can be found under óip_optoiso_baseôsection under Register Definitions in the Hardware manual.

IOCTL_IP_OPTO_ISO16_CTA_INT_EN

Function: Sets the enable that allows counter timer A to cause an interrupt.
Input: None
Output: None
Notes: Leaves all other bit values in the base register unchanged. Detailed definition can be found under óip_optoiso_baseôsection under Register Definitions in the Hardware manual.

IOCTL_IP_OPTO_ISO16_CTA_INT_DIS

Function: Clears the enable that allows counter timer A to cause an interrupt.
Input: None
Output: None
Notes: Leaves all other bit values in the base register unchanged. Detailed definition can be found under óip_optoiso_baseôsection under Register Definitions in the Hardware manual.

IOCTL_IP_OPTO_ISO16_CTB_INT_EN

Function: Sets the enable that allows counter timer B to cause an interrupt.
Input: None
Output: None
Notes: Leaves all other bit values in the base register unchanged. Detailed definition can be found under óip_optoiso_baseôsection under Register Definitions in the Hardware manual.

IOCTL_IP_OPTO_ISO16_CTB_INT_DIS

Function: Clears the enable that allows counter timer A to cause an interrupt.
Input: None
Output: None
Notes: Leaves all other bit values in the base register unchanged. Detailed definition can be found under óp_optoiso_baseôsection under Register Definitions in the Hardware manual.

IOCTL_IP_OPTO_ISO16_SET_FET_CONTROL

Function: Enables or disables each of the 16 individual FET
Input: USHORT
Output: None
Notes: Set to 1 to enable the FET. 0 to disable. Detailed definition can be found under óp_optoiso_fetôsection under Register Definitions in the Hardware manual.

IOCTL_IP_OPTO_ISO16_GET_FET_CONTROL

Function: Returns the status of the FET control set with the above call.
Input: None
Output: USHORT
Notes: 1 is enabled and 0 is disabled. Detailed definition can be found under óp_optoiso_fetôsection under Register Definitions in the Hardware manual.

IOCTL_IP_OPTO_ISO16_SET_WAVE_CONTROL

Function: Sets OUT signal control to bit mapped or waveform
Input: USHORT
Output: None
Notes: Writing a 1 to a bit will switch to CTA waveform control. The default 0 is FET bitmapped control. Detailed definition can be found under óp_optoiso_wavôsection under Register Definitions in the Hardware manual.

IOCTL_IP_OPTO_ISO16_GET_WAVE_CONTROL

Function: Returns the status of the control set with the above call.
Input: None
Output: USHORT
Notes: Detailed definition can be found under óp_optoiso_wavôsection under Register Definitions in the Hardware manual.

IOCTL_IP_OPTO_ISO16_SET_TIMER_CONT

Function: Set the timer control register
Input: IP_OPTO_ISO16_TC structure
Output: None
Notes: See the definition of IP_OPTO_ISO16_TC below. Detailed bit definitions can be found under óip_optoiso_tcô section under Register Definitions in the Hardware manual.

```
typedef struct _IP_OPTO_ISO16_TC {
   BOOLEAN  LoadTimerA;
   BOOLEAN  ClearTimerB;
   BOOLEAN  HoldTimerB;
} IP_OPTO_ISO16_TC, *PIP_OPTO_ISO16_TC;
```

IOCTL_IP_OPTO_ISO16_GET_TIMER_CONT

Function: Get the timer control
Input: None
Output: IP_OPTO_ISO16_TC structure
Notes: Detailed bit definitions can be found under óip_optoiso_tcô section under Register Definitions in the Hardware manual.

IOCTL_IP_OPTO_ISO16_SET_PRELOAD

Function: Set value to the preload register
Input: ULONG
Output: None
Notes: Detailed definition can be found under óPre-Load Registerô section under Register Definitions in the Hardware manual.

IOCTL_IP_OPTO_ISO16_GET_PRELOAD

Function: Get value set to the preload register
Input: None
Output: ULONG
Notes: Detailed definition can be found under óPre-Load Registerô section under Register Definitions in the Hardware manual.

IOCTL_IP_OPTO_ISO16_SET_TIMER_MASK

Function: Set value to the timer mask register
Input: ULONG
Output: None
Notes: Detailed definition can be found under óMask Registerô section under Register Definitions in the Hardware manual.

IOCTL_IP_OPTO_ISO16_GET_TIMER_MASK

Function: Get value set to timer mask register
Input: None
Output: ULONG
Notes: Detailed definition can be found under ôMask Registerö section under Register Definitions in the Hardware manual.

IOCTL_IP_OPTO_ISO16_GET_READBACK_CNT

Function: Get value of Counter/Timer B
Input: None
Output: ULONG
Notes: Detailed definition can be found under ôRead-Back Registerö section under Register Definitions in the Hardware manual.

IOCTL_IP_OPTO_ISO16_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_OPTO_ISO16_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_OPTO_ISO16_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_OPTO_ISO16_FORCE_INTERRUPT

Function: Causes a system interrupt to occur.
Input: IP_429II_INT_SEL structure
Output: None
Notes: Causes an interrupt to be asserted on the IP bus. This IOCTL is used for development, to test interrupt processing.

IOCTL_IP_OPTO_ISO16_SET_VECTOR

Function: Writes an 8 bit value to the interrupt vector register.
Input: UCHAR
Output: None
Notes: Required when used in non auto-vectored systems.

IOCTL_IP_OPTO_ISO16_GET_VECTOR

Function: Returns a stored vector value.
Input: None
Output: UCHAR
Notes:

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 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 Street, Suite C
Santa Cruz, CA 95060
831-457-8891
831-457-4793 Fax
support@dyneng.com