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

IpParTTL/IpPar_1/ IpPar_2/IpPar_3/ IpPar_4/IpPar_5/ IpPar485

Driver Documentation

Developed with Windows Driver Foundation Ver1.9

Revision A
Corresponding Hardware: Revision C.0
10-2009-0203

IpParTTL/_1/_2/_3/_4/_5/485
WDM Device Drivers for the
IP-Parallel-IO IP Modules

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

Manual Revision A. Revised June 21, 2017.

Table of Contents

Introduction

Note: In this document IpPar?? and IP- Parallel-?? refer to one of seven versions of the IpParIO driver and IP-Parallel-IO hardware. The versions and the I/O distributions are as follows:

Board Type	Driver Name	IO configuration
IP-Parallel-TTL	IpParTTL	48 TTL / 0 RS485
IP-Parallel-1	IpPar_1	40 TTL / 4 RS485
IP-Parallel-2	IpPar_2	32 TTL / 8 RS485
IP-Parallel-3	IpPar_3	24 TTL / 12 RS485
IP-Parallel-4	IpPar_4	16 TTL / 16 RS485
IP-Parallel-5	IpPar_5	8 TTL / 20 RS485
IP-Parallel-485	IpPar485	0 TTL / 24 RS485

The IP-PARALLEL-?? 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 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 “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 RegisterTest to your application to get started. The test is 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 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 can't 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.

When the IP-PARALLEL-?? board is recognized by the IP Carrier Driver, the carrier driver will start the IP- PARALLEL-?? 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- PARALLEL-?? with the carrier switch setting and the slot number of the IP- PARALLEL-?? device. From within the IP-PARALLEL-?? 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 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-IO device user manual (also referred to as the hardware manual).

Driver Installation

There are several files provided in each driver package. These files include IpPar??.sys, IpPar??Public.h, IpPublic.h, WdfCoInstaller01009.dll, IpModDrivers.inf and ipmoddrivers.cat.

IpPar??.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 IpPar?? driver is installed, to verify the digital signature in ipmoddrivers.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 7 Installation

Copy IpModDrivers.inf, ipmoddrivers.cat, WdfCoInstaller01009.dll, IpPar??.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 IpPar?? 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 IpPar??Public.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 IpPar?? 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 IpPar?? driver are described below:

IOCTL_IP_PAR??_GET_INFO

Function: Returns the device instance number, driver version, carrier switch value and carrier slot number.
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 {
   UCHAR    DriverRev;         // Driver revision
   UCHAR    FirmwareRev;       // Firmware major revision
   UCHAR    FirmwareRevMin;    // Firmware minor revision
   UCHAR    InstanceNum;       // Zero-based device number
   UCHAR    CarrierSwitch;     // 0..0xFF
   UCHAR    CarrierSlotNum;    // 0..7 -> IP slots A, B, C, D, E, F, G or H
   UCHAR    CarDriverRev;      // Carrier driver revision
   UCHAR    CarFirmwareRev;    // Carrier firmware major revision
   UCHAR    CarFirmwareRevMin;// Carrier firmware minor revision
   UCHAR    CarCPLDRev;        //**Used for PCIe carriers only**0xFF for others
   UCHAR    CarCPLDRevMin;     //**Used for PCIe carriers only**0xFF for others
   BOOLEAN  Ip32MCapable;      // IP capable of both 8MHz and 32MHz operation
   BOOLEAN  NewIpCntl;         // New IP slot control interface
   WCHAR    LocationString[IP_LOC_STRING_SIZE];
} DRIVER_IP_DEVICE_INFO, *PDRIVER_IP_DEVICE_INFO;
```

IOCTL_IP_PAR??_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_PAR??_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_PAR??_SET_BASE_CONFIG

Function: Sets configuration parameters in the base control register.
Input: IP_PAR_IO_BASE_CONFIG structure
Output: None
Notes: Controls the output data latch behavior, the timer/counter A and B interrupt enables and enables the square wave output on the upper data bit. The output data latch can be set to enable, disable or auto. When in auto the outputs from all data registers are enabled onto the output bus simultaneously after each data update call. See the definition of IP_PAR_IO_BASE_CONFIG below. Bit definitions can be found under the 'BASE_CNTL' section under Register Definitions in the Hardware manual.

```
typedef struct _IP_PAR_IO_BASE_CONFIG {
   OUT_SEL  Outen;
   BOOLEAN  CntTmrAIntEn;
   BOOLEAN  CntTmrAWaveEn;
   BOOLEAN  CntTmrBIntEn;
} IP_PAR_IO_BASE_CONFIG, *PIP_PAR_IO_BASE_CONFIG;
```

IOCTL_IP_PAR??_GET_BASE_CONFIG

Function: Returns the configuration of the base control register.
Input: None
Output: IP_PAR_IO_BASE_CONFIG structure
Notes: Returns the values set in the previous call.

IOCTL_IP_PAR??_SET_TTL_DATA

Function: Sets the value of the TTL outputs on the board.
Input: IP_PAR_TTL_BITS structure
Output: None
Notes: The number of valid bits varies with the number of TTL outputs on the board. This call is not valid on the IP-Parallel-485 board as it has no TTL drivers. See the definition of IP_PAR_TTL_BITS below. Note that this same structure is used for all IOCTLs that concern TTL data. Only the bits that are relevant to the specific hardware are valid. See CNTL0, CNTL1, and CNTL2 for the specific hardware in the hardware manual for more details.

```
typedef struct _IP_PAR_TTL_BITS
{
   ULONG    LoWord;
   ULONG    HiWord;
} IP_PAR_TTL_BITS, *PIP_PAR_TTL_BITS;
```

IOCTL_IP_PAR??_GET_TTL_DATA

Function: Returns the state of the TTL outputs in the output data register.
Input: None
Output: IP_PAR_TTL_BITS structure

Notes: The number of valid bits varies with the number of TTL outputs. This call is not valid on the IP-Parallel-485 board as it has no TTL drivers. See the definition of IP_PAR_TTL_BITS above.

IOCTL_IP_PAR??_SET_485_DIR

Function: Sets the directions for the RS485 drivers on the board.
Input: unsigned long int
Output: None
Notes: The number of valid bits varies with the number of RS485 drivers from 24 on the IP-Parallel-485 to zero on the IP-Parallel-TTL. A one in a bit position corresponds to setting that driver to be an output, while a zero corresponds to an input. This call is not valid on the IP-Parallel-TTL board. See CNTL0, CNTL1, and CNTL2 for the specific hardware in the hardware manual for more details.

IOCTL_IP_PAR??_GET_485_DIR

Function: Returns the direction bits for the RS485 drivers on the board.
Input: None
Output: unsigned long int
Notes: Returns the bits set in the previous call. This call is not valid on the IP-Parallel-TTL board.

IOCTL_IP_PAR??_SET_485_DATA

Function: Sets the value of the RS485 outputs on the board.
Input: unsigned long int
Output: None
Notes: The number of valid bits varies with the number of RS485 drivers on the board and only the drivers that are set to be outputs can drive the output bus. This call is not valid on the IP-Parallel-TTL board as it has no RS485 drivers. See CNTL0, CNTL1, and CNTL2 for the specific hardware in the hardware manual for more details.

IOCTL_IP_PAR??_GET_485_DATA

Function: Returns the state of the RS485 bits in the output data register.
Input: None
Output: unsigned long int
Notes: The number of valid bits varies with the number of RS485 drivers. This call is not valid on the IP-Parallel-TTL board as it has no RS485 drivers.

IOCTL_IP_PAR??_SET_TTL_INT_EN

Function: Selects which TTL inputs can cause an interrupt.
Input: IP_PAR_TTL_BITS structure
Output: None

Notes: This call defines the mask of which of the TTL input lines will be enabled to cause an interrupt when the specified conditions are met (1 = enabled, 0 = disabled). The number of valid bits varies with the number of TTL I/O. This call is not valid on the IP-Parallel-485 board as it has no TTL drivers. See the definition of IP_PAR_TTL_BITS above. See Int_en0, Int_en1, and Int_en2 in the hardware manual for more details.

IOCTL_IP_PAR??_GET_TTL_INT_EN

Function: Returns the interrupt enable values set in the previous call.
Input: None
Output: IP_PAR_TTL_BITS structure
Notes: The number of valid bits varies with the number of TTL I/O lines. This call is not valid on the IP-Parallel-485 board as it has no TTL drivers. See the definition of IP_PAR_TTL_BITS above.

IOCTL_IP_PAR??_SET_485_INT_EN

Function: Selects which RS485 inputs can cause an interrupt.
Input: unsigned long int
Output: None
Notes: This call defines the mask of which of the RS485 input lines will be enabled to cause an interrupt when the specified conditions are met (1 = enabled, 0 = disabled). The number of valid bits varies with the number of RS485 drivers. This call is not valid on the IP-Parallel-TTL board as it has no RS485 drivers. See Int_en0, Int_en1, and Int_en2 in the hardware manual for more details.

IOCTL_IP_PAR??_GET_485_INT_EN

Function: Returns the interrupt enable value set in the previous call.
Input: None
Output: unsigned long int
Notes: The number of valid bits varies with the number of RS485 I/O lines. This call is not valid on the IP-Parallel-TTL board as it has no RS485 drivers.

IOCTL_IP_PAR??_SET_TTL_EDGE_LEVEL

Function: Selects whether a TTL input is edge-sensitive or level sensitive.
Input: IP_PAR_TTL_BITS structure
Output: None
Notes: Determines whether the interrupt for each of the enabled TTL input lines responds to a static logic level or a transition between levels (1 = edge, 0 = level). The number of valid bits varies with the number of TTL I/O. This call is not valid on the IP-Parallel-485 board as it has no TTL drivers. See the definition of IP_PAR_TTL_BITS above. See Edg_Lvl0, Edg_Lvl1, and Edg_Lvl2 in the hardware manual for more details.

IOCTL_IP_PAR??_GET_TTL_EDGE_LEVEL

Function: Returns the interrupt edge/level values set in the previous call.
Input: None
Output: IP_PAR_TTL_BITS structure
Notes: The number of valid bits varies with the number of TTL I/O lines. This call is not valid on the IP-Parallel-485 board as it has no TTL drivers. See the definition of IP_PAR_TTL_BITS above.

IOCTL_IP_PAR??_SET_485_EDGE_LEVEL

Function: Selects whether an RS485 input is edge or level sensitive.
Input: unsigned long int
Output: None
Notes: Determines whether the interrupt for each of the enabled RS485 input lines will respond to a static logic level or a transition between levels (1 = edge, 0 = level). The number of valid bits varies with the number of RS485 I/O. This call is not valid on the IP-Parallel-TTL board as it has no RS485 drivers. See Edg_Lvl0, Edg_Lvl1, and Edg_Lvl2 in the hardware manual for more details.

IOCTL_IP_PAR??_GET_485_EDGE_LEVEL

Function: Returns the interrupt edge/level values set in the previous call.
Input: None
Output: unsigned long int
Notes: The number of valid bits varies with the number of RS485 I/O. This call is not valid on the IP-Parallel-TTL board as it has no RS485 drivers.

IOCTL_IP_PAR??_SET_TTL_POLARITY

Function: Selects whether a TTL input is active high or active low.
Input: IP_PAR_TTL_BITS structure
Output: None
Notes: Determines the polarity of the level or edge to which the interrupt for each of the input lines will respond (1 = inverted: active low or falling edge, 0 = non-inverted: active high or rising edge). The number of valid bits varies with the number of TTL I/O. This call is not valid on the IP-Parallel-485 board as it has no TTL drivers. See the definition of IP_PAR_TTL_BITS above. See Pol0, Pol1, and Pol2 in the hardware manual for more details.

IOCTL_IP_PAR??_GET_TTL_POLARITY

Function: Returns the interrupt polarity values set in the previous call.
Input: None

Output: IP_PAR_TTL_BITS structure
Notes: The number of valid bits varies with the number of TTL I/O lines. This call is not valid on the IP-Parallel-485 board as it has no TTL drivers. See the definition of IP_PAR_TTL_BITS above.

IOCTL_IP_PAR??_SET_485_POLARITY

Function: Selects whether an RS485 input is active high or active low.
Input: unsigned long int
Output: None
Notes: Determines the polarity of the level or edge to which the interrupt for each of the input lines will respond (1 = inverted: active low or falling edge, 0 = non-inverted: active high or rising edge). The number of valid bits varies with the number of RS485 I/O. This call is not valid on the IP-Parallel-TTL board as it has no RS485 drivers. See Pol0, Pol1, and Pol2 in the hardware manual for more details.

IOCTL_IP_PAR??_GET_485_POLARITY

Function: Returns the interrupt polarity values set in the previous call.
Input: None
Output: unsigned long int
Notes: The number of valid bits varies with the number of RS485 I/O. This call is not valid on the IP-Parallel-TTL board as it has no RS485 drivers.

IOCTL_IP_PAR??_READ_DIRECT

Function: Reads the input data bus directly.
Input: None
Output: IP_PAR_IO_READ_DATA structure
Notes: This call reads the raw real-time input data from the TTL and RS485 input lines and returns an. IP_PAR_IO_READ_DATA structure. This structure has separate fields for the TTL and RS485 data bits. The number of valid bits in each field varies with the number of each type of I/O. See the definition of IP_PAR_IO_READ_DATA below and the definition of IP_PAR_TTL_BITS above. See Datain_dir0, Datain_dir1, and Datain_dir2 in the hardware manual for more details.

```
 // Read input data structure
typedef struct _IP_PAR_IO_READ_DATA
{
   IP_PAR_TTL_BITS   InDataTTL;
   ULONG             InData485;
} IP_PAR_READ_DATA, *PIP_PAR_READ_DATA;
```

IOCTL_IP_PAR??_READ_ FILTERED

Function: Reads the filtered input data.
Input: None
Output: IP_PAR_IO_READ_DATA structure

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 a IP_PAR_IO_READ_DATA structure, which has separate fields for the TTL and RS485 data bits. The number of valid bits in each field varies with the number of each type of I/O. The latched bits are automatically cleared when read by this call. See Datain_fil0, Datain_fil1, and Datain_fil2 in the hardware manual for more details.

IOCTL_IP_PAR??_SET_COUNTER_PRELOAD

Function: Stores a value to be loaded into the A-Counter when a new count is loaded.
Input: unsigned long int
Output: None
Notes: The A-Counter counts down from the loaded count value. When it reaches zero, this count is re-loaded and an interrupt will be generated if the Counter A interrupt is enabled.

IOCTL_IP_PAR??_GET_COUNTER_PRELOAD

Function: Returns the value stored in the A-Counter preload registers.
Input: None
Output: unsigned long int
Notes: Returns the value written with the previous call.

IOCTL_IP_PAR??_SET_TIMER_MASK

Function: Stores a value in the B-Timer mask registers
Input: unsigned long int
Output: None
Notes: The B-Timer counts up from zero. When a counter bit is high that corresponds to a bit that asserted in the mask register an interrupt will be generated if the Timer B interrupt is enabled.

IOCTL_IP_PAR??_GET_TIMER_MASK

Function: Returns the value stored in the B-Timer mask registers.
Input: None
Output: unsigned long int
Notes: Returns the value written with the previous call.

IOCTL_IP_PAR??_LOAD_COUNTER

Function: Causes the A-Counter to be loaded with the preload value.
Input: None
Output: None

Notes:

IOCTL_IP_PAR??_CLEAR_TIMER

Function: Clears the B-Timer count to zero.
Input: None
Output: None
Notes:

IOCTL_IP_PAR??_GET_TIMER_COUNT

Function: Reads the current value of the Timer B count.
Input:
Output: unsigned long int
Notes: The hold count bit is automatically set before the count is read and cleared afterward. This guarantees that a consistent count is read since two accesses are required to read all 32 bits.

IOCTL_IP_PAR??_GET_STATUS

Function: Returns the status bits in the INT_STAT register.
Input: None
Output: unsigned short int
Notes: The interrupt status bits are read by this call and the latched bits are then automatically cleared.

IOCTL_IP_PAR??_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_PAR??_ENABLE_INTERRUPT

Function: Enables the master interrupt.
Input: None
Output: None
Notes: Sets the master interrupt enable, leaving all other bit values in the IPPAR??_BASE register unchanged. Also checks the state of the IP slot control register interrupt 0 enable bit in the saved configuration, and sets it if needed. 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_PAR??_DISABLE_INTERRUPT

Function: Disables the master interrupt.
Input: None
Output: None
Notes: Clears the master interrupt enable, leaving all other bit values in the IPPAR??_BASE register unchanged. This IOCTL is used when interrupt processing is no longer desired.

IOCTL_IP_PAR??_FORCE_INTERRUPT

Function: Causes a system interrupt to occur.
Input: None
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_PAR??_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.

IOCTL_IP_PAR??_GET_VECTOR

Function: Returns the current interrupt vector value.
Input: None
Output: unsigned character
Notes:

IOCTL_IP_PAR??_GET_ISR_STATUS

Function: Returns the interrupt status and vector read in the last ISR.
Input: None
Output: IPPAR??_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.

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

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