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-429-II
ARINC 429 Interface
1-4 Transmitters
2-8 Receivers

Driver Documentation

Developed with Windows Driver Foundation Ver1.9

Manual Revision A
Corresponding Hardware: Revision A/B
10-2007-0501/2
FLASH revision A1

IP-429II

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

Table of Contents

Introduction

The Ip429II 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 Ip429II 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 test software can be ported to your application to provide a running start. It is recommended to port the Base Reg2 or Parallel Port 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 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.

IP-429-II has a Spartan2 Xilinx FPGA to implement the IP Interface, protocol control and status for the IO. The main feature of the design are the 429 interface devices.

When the IP-429-II board is recognized by the IP Carrier Driver, the carrier driver will start the IP429II 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-429-II with the carrier switch setting and the slot number of the IP-429-II device. From within the IP429II driver the user can access the switch and slot information to determine the specific device being accessed when more than one are installed. In addition the driver determines the type of IP-429-II installed (-1, -2, -3, -4). The number of channels is reported in the test menu header and is available from the driver.

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-429-II user manual (also referred to as the hardware manual).

Driver Installation

There are several files provided in each driver package. These files include Ip429II.sys, Ip429IIPublic.h, IpPublic.h, WdfCoInstaller01009.dll, IpDevices.inf and IpDevices.cat.

Ip429IIPublic.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 Ip429II 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 7 Installation

Copy IpDevices.inf, IpDevices.cat, WdfCoInstaller01009.dll, Ip429II.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 Ip429II 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 Ip429IIPublic.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 Ip429II 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 Ip429ll driver are described below:

IOCTL_IP_429II_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 moduleÕs 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_429II_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_429II_GET_IP_CONTROL

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_429II_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_429II_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_429II_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_429II_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 using either of the two IP interrupt lines. This IOCTL is used for development, to test interrupt processing.

```
typedef struct _IP_429II_INT_SEL {
   BOOLEAN  IntSet0;
   BOOLEAN  IntSet1;
} IP_429II_INT_SEL, *PIP_429II_INT_SEL;
```

IOCTL_IP_429II_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_429II_GET_VECTOR

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

Notes:

IOCTL_IP_429II_SET_BASE0_CONFIG

Function: Sets configuration parameters in the IP base 0 control register.
Input: IP_429II_BASE0_CONFIG structure
Output: None
Notes: See the definition of IP_429II_BASE0_CONFIG below. Bit definitions can be found under ó BASE_REG0ô section under Register Definitions in the Hardware manual.

```
typedef struct _IP_429II_BASE0_CONFIG {
   BOOLEAN     IpClock32;
   BOOLEAN     ForceInt;
   BOOLEAN     ClrTimeStamp;
   BOOLEAN     TxEnable1;
   BOOLEAN     TxEnable2;
   BOOLEAN     TxEnable3;
   BOOLEAN     TxEnable4;
} IP_429II_BASE0_CONFIG, *PIP_429II_BASE0_CONFIG;
```

IOCTL_IP_429II_GET_BASE0_CONFIG

Function: Returns the configuration of the IP base 0 control register.
Input: None
Output: IP_429II_BASE0_CONFIG structure
Notes: Returns the values set in the previous call.

IOCTL_IP_429II_SET_BASE1_CONFIG

Function: Sets configuration parameters in the IP base 1 control register.
Input: IP_429II_BASE1_CONFIG structure
Output: None
Notes: See the definition of IP_429II_BASE1_CONFIG below. Bit definitions can be found under ó BASE_REG1ô section under Register Definitions in the Hardware manual.

```
typedef struct _IP_429II_BASE1_CONFIG{
   BOOLEAN     DbCen1;
   BOOLEAN     DbCen2;
   BOOLEAN     DbCen3;
   BOOLEAN     DbCen4;
   BOOLEAN     Enable429n1;
   BOOLEAN     Enable429n2;
   BOOLEAN     Enable429n3;
   BOOLEAN     Enable429n4;
} IP_429II_BASE1_CONFIG, *PIP_429II_BASE1_CONFIG;
```

IOCTL_IP_429II_GET_BASE1_CONFIG

Function: Returns the configuration of the IP base 1 control register.
Input: None
Output: IP_429II_BASE1_CONFIG structure
Notes: Returns the values set in the previous call.

IOCTL_IP_429II_SET_BASE2_CONFIG

Function: Sets configuration parameters in the IP base 2 control register.
Input: IP_429II_BASE2_CONFIG structure
Output: None
Notes: See the definition of IP_429II_BASE2_CONFIG below. Bit definitions can be found under ó_BASE_REG2ôsection under Register Definitions in the Hardware manual.

```
typedef struct _IP_429II_BASE2_CONFIG{
   BOOLEAN    HighLow1;
   BOOLEAN    HighLow2;
   BOOLEAN    HighLow3;
   BOOLEAN    HighLow4;
} IP_429II_BASE2_CONFIG, *PIP_429II_BASE2_CONFIG;
```

IOCTL_IP_429II_GET_BASE2_CONFIG

Function: Returns the configuration of the IP base 2 control register.
Input: None
Output: IP_429II_BASE2_CONFIG structure
Notes: Returns the values set in the previous call.

IOCTL_IP_429II_SET_BASE3_CONFIG

Function: Sets configuration parameters in the IP base 3 control register.
Input: IP_429II_BASE3_CONFIG structure
Output: None
Notes: See the definition of IP_429II_BASE3_CONFIG below. Bit definitions can be found under ó BASE_REG3ô section under Register Definitions in the Hardware manual.

```
typedef struct _IP_429II_BASE3_CONFIG{
   BOOLEAN    IntEnRx1;
   BOOLEAN    IntEnRx2;
   BOOLEAN    IntEnRx3;
   BOOLEAN    IntEnRx4;
   BOOLEAN    IntEnTx1;
   BOOLEAN    IntEnTx2;
   BOOLEAN    IntEnTx3;
   BOOLEAN    IntEnTx4;
} IP_429II_BASE3_CONFIG, *PIP_429II_BASE3_CONFIG;
```

IOCTL_IP_429II_GET_BASE3_CONFIG

Function: Returns the configuration of the IP base 3 control register.
Input: None
Output: IP_429II_BASE3_CONFIG structure
Notes: Returns the values set in the previous call.

IOCTL_IP_429II_GET_VERSION

Function: Returns the IP429II version (1, 2, 3, or 4) as well as the driver flash minor and major revisions.
Input: None
Output: IP_429II_VESION structure
Notes: See the definition of IP_429II_VERSION below. Definitions can be found under the ó STATUS1ô section under Register Definitions in the Hardware manual.

```
typedef struct _IP_429II_VERSION {
   UCHAR    version;
   UCHAR    minorFlashRev;
   UCHAR    majorFlashRev;
} IP_429II_VERSION, *PIP_429II_VERSION;
```

IOCTL_IP_429II_GET_ISR_STATUS

Function: Interrupt status, and vector read in the ISR from the last user interrupt.
Input: None
Output: IP_429II_ISR_STAT
Notes: Returns the interrupt status that was read in the interrupt service routine of the last interrupt caused by one of the enabled channel interrupts.

```
typedef struct _IP_429II_ISR_STAT {
   UCHAR    RxDevStatus;
   UCHAR    TxDevStatus;
   USHORT   InterruptStatus;  // stored Interrupt status from ISR
   USHORT   InterruptVector;  // stored Interrupt vector from ISR
} IP_429II_ISR_STAT, *PIP_429II_ISR_STAT;
```

IOCTL_IP_429II_GET_INT_STATUS

Function: Returns the current interrupt status.
Input: None
Output: IP_429II_INT_STAT
Notes:

```
typedef struct _IP_429II_INT_STAT

{
   USHORT   RxStatus;         // current RxStatus
   USHORT   TxStatus;         // current TxStatus
} IP_429II_INT_STAT, *PIP_429II_INT_STAT;
```

IOCTL_IP_429II_SET_PAR_DATA

Function: Write to TTL Parallel Port
Input: IP_429II_PARA_DATA
Output: None
Notes: Lower 4 bits are read writeable. See the definition of IP_429II_PARA_DATA below. Bit definitions can be found under ó Parallelôsection under Register Definitions in the Hardware manual.

```
typedef struct _IP_429II_PARA_DATA
{
   BOOLEAN     Pio_0;
   BOOLEAN     Pio_1;
   BOOLEAN     Pio_2;
   BOOLEAN     Pio_3;
   BOOLEAN     Pi_4;
   BOOLEAN     Pi_5;
   BOOLEAN     Pi_6;
   BOOLEAN     Pi_7;
   USHORT      ReadBack;
```

```
} IP_429II_PARA_DATA, *PIP_429II_PARA_DATA;
```

IOCTL_IP_429II_GET_PAR_DATA

Function: Read Parallel Data plus read-back of written data
Input: none
Output: IP_429II_PARA_DATA
Notes: 7-0 = Parallel data in, 15-8 = lower byte from register. Returns the values set in the previous call.

Please note that the IOCTLôs defined for devices not installed should not be used.

IOCTL_IP_429II_GET_TIMESTAMP

Function: Read the timestamp for each of the devices
Input: none
Output: IP_429II_TS_DEV
Notes: The counter is 32 bits wide and counts up at a rate of 1MHz. When the receive interrupt for a particular channel is detected, the count is stored into the time tag register. See the definition of IP_429II_TS_DEV below. Bit definitions can be found under ó_CHXX_TSô section under Register Definitions in the Hardware manual.

```
typedef struct _IP_429II_TS_DEV
{
   ULONG    Dev1Ch1;
   ULONG    Dev1Ch2;
   ULONG    Dev2Ch1;
   ULONG    Dev2Ch2;
   ULONG    Dev3Ch1;
   ULONG    Dev3Ch2;
   ULONG    Dev4Ch1;
   ULONG    Dev4Ch2;
} IP_429II_TS_DEV, *PIP_429II_TS_DEV;
```

IOCTL_IP_429II_SET_TXD_DATA_DEV1

Function: Write to Device 1 upper and lower transmit data
Input: IP_429II_TX_DEV
Output: none
Notes: See the definition of IP_429II_TX_DEV below.

```
typedef struct _IP_429II_TX_DEV
{
   USHORT   LowerData;
   USHORT   UpperData;
} IP_429II_TX_DEV, *PIP_429II_TX_DEV;
```

IOCTL_IP_429II_GET_RXD_DATA_DEV1_1

Function: Write to Device 1 Channel 1 upper and lower transmit data
Input: none
Output: IP_429II_RX_DEV
Notes:

```
typedef struct _IP_429II_RX_DEV
{
   USHORT   LowerData;
   USHORT   UpperData;
} IP_429II_RX_DEV, *PIP_429II_RX_DEV;
```

IOCTL_IP_429II_GET_RXD_DATA_DEV1_2

Function: Write to Device 1 Channel 2 upper and lower transmit data
Input: none
Output: IP_429II_RX_DEV
Notes:

```
typedef struct _IP_429II_RX_DEV
{
   USHORT   LowerData;
   USHORT   UpperData;
} IP_429II_RX_DEV, *PIP_429II_RX_DEV;
```

IOCTL_IP_429II_SET_CNTL_DEV1

Function: Write to Device 1 control register
Input: IP_429II_CONTROL
Output: none
Notes: Structure for 429 interface device options. Speed, Parity, masking etc.

```
typedef struct _IP_429II_CONTROL
{
   BOOLEAN     SDEnable1;
   SD_COMPARE  SDMatch1;
   BOOLEAN     SDEnable2;
   SD_COMPARE  SDMatch2;
   BOOLEAN     TransParEn;
   BOOLEAN     ExtLB;
   BOOLEAN     ParityTest;
   BOOLEAN     Tx12_5K;
   BOOLEAN     Rx12_5K;
   BOOLEAN     WordLength25;
} IP_429II_CONTROL, *PIP_429II_CONTROL;
```

IOCTL_IP_429II_SET_TXD_32_DEV1

Function: Write to Device 1 as a long word
Input: ULONG
Output: none
Notes: Write to both halves with one call using the carrier auto conversion

IOCTL_IP_429II_GET_RXD_32_DEV1_1

Function: Read from Device 1 Channel 1 as a long word
Input: none
Output: ULONG
Notes: Read from both halves with one call using the carrier auto conversion. In self-test mode data from the channel is inverted.

IOCTL_IP_429II_GET_RXD_32_DEV1_2

Function: Read from Device 1 Channel 2 as a long word
Input: none
Output: ULONG
Notes: Read from both halves with one call using the carrier auto conversion. In self-test mode data from the channel is inverted.

IOCTL_IP_429II_SET_TXD_DATA_DEV2

Function: Write to Device 2 upper and lower transmit data
Input: IP_429II_TX_DEV
Output: none
Notes: See the definition of IP_429II_TX_DEV below.

```
typedef struct _IP_429II_TX_DEV
{
   USHORT   LowerData;
   USHORT   UpperData;
} IP_429II_TX_DEV, *PIP_429II_TX_DEV;
```

IOCTL_IP_429II_GET_RXD_DATA_DEV2_1

Function: Write to Device 2 Channel 1 upper and lower transmit data
Input: none
Output: IP_429II_RX_DEV
Notes:

```
typedef struct _IP_429II_RX_DEV
{
   USHORT   LowerData;
   USHORT   UpperData;
} IP_429II_RX_DEV, *PIP_429II_RX_DEV;
```

IOCTL_IP_429II_GET_RXD_DATA_DEV2_2

Function: Write to Device 2 Channel 2 upper and lower transmit data
Input: none
Output: IP_429II_RX_DEV
Notes:

```
typedef struct _IP_429II_RX_DEV
{
   USHORT   LowerData;
   USHORT   UpperData;
} IP_429II_RX_DEV, *PIP_429II_RX_DEV;
```

IOCTL_IP_429II_SET_CNTL_DEV2

Function: Write to Device 2 control register
Input: IP_429II_CONTROL
Output: none
Notes: Structure for 429 interface device options. Speed, Parity, masking etc.

```
typedef struct _IP_429II_CONTROL
{
   BOOLEAN     SDEnable1;
   SD_COMPARE  SDMatch1;
   BOOLEAN     SDEnable2;
   SD_COMPARE  SDMatch2;
   BOOLEAN     TransParEn;
   BOOLEAN     ExtLB;
   BOOLEAN     ParityTest;
   BOOLEAN     Tx12_5K;
   BOOLEAN     Rx12_5K;
   BOOLEAN     WordLength25;
} IP_429II_CONTROL, *PIP_429II_CONTROL;
```

IOCTL_IP_429II_SET_TXD_32_DEV2

Function: Write to Device 2 as a long word
Input: ULONG
Output: none
Notes: Write to both halves with one call using the carrier auto conversion

IOCTL_IP_429II_GET_RXD_32_DEV2_1

Function: Read from Device 2 Channel 1 as a long word
Input: none
Output: ULONG
Notes: Read from both halves with one call using the carrier auto conversion. In self-test mode data from the channel is inverted.

IOCTL_IP_429II_GET_RXD_32_DEV2_2

Function: Read from Device 2 Channel 2 as a long word
Input: none
Output: ULONG
Notes: Read from both halves with one call using the carrier auto conversion. In self-test mode data from the channel is inverted.

IOCTL_IP_429II_SET_TXD_DATA_DEV3

Function: Write to Device 3 upper and lower transmit data
Input: IP_429II_TX_DEV
Output: none
Notes: See the definition of IP_429II_TX_DEV below.

```
typedef struct _IP_429II_TX_DEV
{
   USHORT   LowerData;
   USHORT   UpperData;
} IP_429II_TX_DEV, *PIP_429II_TX_DEV;
```

IOCTL_IP_429II_GET_RXD_DATA_DEV3_1

Function: Write to Device 3 Channel 1 upper and lower transmit data
Input: none
Output: IP_429II_RX_DEV
Notes:

```
typedef struct _IP_429II_RX_DEV
{
   USHORT   LowerData;
   USHORT   UpperData;
} IP_429II_RX_DEV, *PIP_429II_RX_DEV;
```

IOCTL_IP_429II_GET_RXD_DATA_DEV3_2

Function: Write to Device 3 Channel 2 upper and lower transmit data
Input: none
Output: IP_429II_RX_DEV
Notes:

```
typedef struct _IP_429II_RX_DEV
{
   USHORT   LowerData;
   USHORT   UpperData;
} IP_429II_RX_DEV, *PIP_429II_RX_DEV;
```

IOCTL_IP_429II_SET_CNTL_DEV3

Function: Write to Device 3 control register
Input: IP_429II_CONTROL
Output: none
Notes: Structure for 429 interface device options. Speed, Parity, masking etc.

```
typedef struct _IP_429II_CONTROL
{
   BOOLEAN     SDEnable1;
   SD_COMPARE  SDMatch1;
   BOOLEAN     SDEnable2;
   SD_COMPARE  SDMatch2;
   BOOLEAN     TransParEn;
   BOOLEAN     ExtLB;
   BOOLEAN     ParityTest;
   BOOLEAN     Tx12_5K;
   BOOLEAN     Rx12_5K;
   BOOLEAN     WordLength25;
} IP_429II_CONTROL, *PIP_429II_CONTROL;
```

IOCTL_IP_429II_SET_TXD_32_DEV3

Function: Write to Device 3 as a long word
Input: ULONG
Output: none
Notes: Write to both halves with one call using the carrier auto conversion

IOCTL_IP_429II_GET_RXD_32_DEV3_1

Function: Read from Device 3 Channel 1 as a long word
Input: none
Output: ULONG
Notes: Read from both halves with one call using the carrier auto conversion. In self-test mode data from the channel is inverted.

IOCTL_IP_429II_GET_RXD_32_DEV3_2

Function: Read from Device 3 Channel 2 as a long word
Input: none
Output: ULONG
Notes: Read from both halves with one call using the carrier auto conversion. In self-test mode data from the channel is inverted.

IOCTL_IP_429II_SET_TXD_DATA_DEV4

Function: Write to Device 4 upper and lower transmit data
Input: IP_429II_TX_DEV
Output: none
Notes: See the definition of IP_429II_TX_DEV below.

```
typedef struct _IP_429II_TX_DEV
{
   USHORT   LowerData;
   USHORT   UpperData;
} IP_429II_TX_DEV, *PIP_429II_TX_DEV;
```

IOCTL_IP_429II_GET_RXD_DATA_DEV4_1

Function: Write to Device 4 Channel 1 upper and lower transmit data
Input: none
Output: IP_429II_RX_DEV
Notes:

```
typedef struct _IP_429II_RX_DEV
{
   USHORT   LowerData;
   USHORT   UpperData;
} IP_429II_RX_DEV, *PIP_429II_RX_DEV;
```

IOCTL_IP_429II_GET_RXD_DATA_DEV4_2

Function: Write to Device 4 Channel 2 upper and lower transmit data
Input: none
Output: IP_429II_RX_DEV
Notes:

```
typedef struct _IP_429II_RX_DEV
{
   USHORT   LowerData;
   USHORT   UpperData;
} IP_429II_RX_DEV, *PIP_429II_RX_DEV;
```

IOCTL_IP_429II_SET_CNTL_DEV4

Function: Write to Device 4 control register
Input: IP_429II_CONTROL
Output: none
Notes: Structure for 429 interface device options. Speed, Parity, masking etc.

```
typedef struct _IP_429II_CONTROL
{
   BOOLEAN     SDEnable1;
   SD_COMPARE  SDMatch1;
   BOOLEAN     SDEnable2;
   SD_COMPARE  SDMatch2;
   BOOLEAN     TransParEn;
   BOOLEAN     ExtLB;
   BOOLEAN     ParityTest;
   BOOLEAN     Tx12_5K;
   BOOLEAN     Rx12_5K;
   BOOLEAN     WordLength25;
} IP_429II_CONTROL, *PIP_429II_CONTROL;
```

IOCTL_IP_429II_SET_TXD_32_DEV4

Function: Write to Device 4 as a long word
Input: ULONG
Output: none
Notes: Write to both halves with one call using the carrier auto conversion

IOCTL_IP_429II_GET_RXD_32_DEV4_1

Function: Read from Device 4 Channel 1 as a long word
Input: none
Output: ULONG
Notes: Read from both halves with one call using the carrier auto conversion. In self-test mode data from the channel is inverted.

IOCTL_IP_429II_GET_RXD_32_DEV1_2

Function: Read from Device 4 Channel 2 as a long word
Input: none
Output: ULONG
Notes: Read from both halves with one call using the carrier auto conversion. In self-test mode data from the channel is inverted.

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 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.

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 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 customer's 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 customer's 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