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

IP-Parallel-TTL-PATT

"IpParTtlPatt"

Windows 10 WDF Driver Documentation

Developed with Windows Driver Foundation Ver1.9

11/28/22
Manual Revision 1p0
Corresponding Hardware: Revision 04
10-2001-0104
FLASH revision 1p1

IpParTtlPatt

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

Table of Contents

Introduction

The IpParTtlPatt driver is a Windows device driver for IP-Parallel-TTL-PATT Industry-pack (IP) module from Dynamic Engineering. The module provides Pattern Generator and GPIO ports. This driver was developed with the Windows Driver Foundation version 1.9 (WDF) from Microsoft, specifically the Kernel-Mode Driver Framework (KMDF).

The IpParTtlPatt software package has two parts. The driver for Windows® 10/11 OS, and the User Application "UserAp" executable.

The driver is delivered electronically. The files supplied are installed into the client system to allow access to the hardware. The UserAp code is delivered in source form [C] and is for the purpose of providing a reference to using the driver.

UserAp 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 Register 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. If you have suggestions for extended features, special calls for particular set-ups or whatever please share them with us.

When the IpParTtlPatt board is recognized by the IP Carrier Driver, the carrier driver will start the IpParTtlPatt 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 IpParTtlPatt with the carrier switch setting and the slot number of the IpParTtlPatt device. From within the IpParTtlPatt driver the user can access the switch and slot information to determine the specific device being accessed when more than one is 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 IpParTtlPatt user manual (also referred to as the hardware manual).

Driver Installation

There are several files provided in each IP driver package. These files include IpParTtlPatt.sys, IpParTtlPatt.cat, IpParTtlPatt.inf.

Please note: Your carrier driver may need to be updated to use the IP module. The list of IP modules is compiled along with the driver and due to signing requirements.

IpParTtlPatt_Public.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. IpPublic.h is supplied with the carrier driver. IpParTtlPatt_Public.h. is supplied with UserAp.

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

Windows 10/11 Installation

Copy the supplied system files to a folder of your choice.

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 IpParTtlPatt 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 IpParTtlPatt_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 IpParTtlPatt device.

IO Controls

The driver uses IO Control calls (IOCTLs) to configure the device. IOCTLs refer to a single Device Object, which controls a single module. IOCTLs are called using the function DeviceIoControl() (see below), and passing in the handle to the device opened with CreateFile() (see above). IOCTLs generally have input parameters, output parameters, or both. Often a custom structure is used.

```
BOOL DeviceIoControl(
  HANDLE        hDevice,          // Handle opened with CreateFile()
  DWORD         dwIoControlCode,  // Control code defined in API header file
  LPVOID        lpInBuffer,       // Pointer to input parameter
  DWORD         nInBufferSize,    // Size of input parameter
  LPVOID        lpOutBuffer,      // Pointer to output parameter
  DWORD         nOutBufferSize,   // Size of output parameter
  LPDWORD       lpBytesReturned,  // Pointer to return length parameter
  LPOVERLAPPED  lpOverlapped,     // Optional pointer to overlapped structure
);                                //   used for asynchronous I/O
```

IOCTLs defined for the IpBis6Gpio driver are described below:

IOCTL_IP_PARttIPATT_GET_INFO

Function: Returns the driver and firmware revisions, module instance number and location and other information.
Input: None
Output: DRIVER_IP_DEVICE_INFO structure
Notes: This call does not access the hardware, only stored driver parameters. NewIpCntl indicates that the module's carrier has expanded slot control capabilities. See the definition of DRIVER_IP_DEVICE_INFO below.

```
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_PARttIPATT_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_PARttIPATT_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_PARttIPATT_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_PARttIPATT_ENABLE_INTERRUPT

Function: Sets the Carrier interrupt enable.
Input: None
Output: None
Notes: Sets the interrupt enable on the carrier. Does not affect the IP Module Master Interrupt Enable

IOCTL_IP_PARttIPATT_DISABLE_INTERRUPT

Function: Clears the carrier interrupt enable.
Input: None
Output: None
Notes: Clears the interrupt enable on the carrier. Does not affect the IP Module Master Interrupt Enable. This IOCTL is used when interrupt processing is no longer desired.

IOCTL_IP_PARttIPATT_FORCE_INTERRUPT

Function: Causes a system interrupt to occur.
Input: none
Output: none
Notes: Sets Force Interrupt bit in Base Register. Also requires MasterInterruptEn and Carrier level interrupt to be enabled. This IOCTL is used for development, to test interrupt processing.

IOCTL_IP_PARttIPATT_CLR_FORCE_INTERRUPT

Function: Clear Force Interrupt Bit
Input: none
Output: none
Notes: Clears Force Interrupt bit in Base Register.

IOCTL_IP_PARttIPATT_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_PARttIPATT_GET_VECTOR

Function: Returns the current interrupt vector value.
Input: none
Output: UCHAR
Notes:

IOCTL_IP_PARttlPATT_GET_ISR_STATUS

Function: Returns the interrupt status, vector read in the last ISR, and the filtered data bits.
Input: none
Output: IP_BIS6_GPIO_ISR_STAT structure
Notes: The status contains the contents of the Interrupt register, GPIO Direct Data, Rising status, and Falling Status from the ISR.

```
// Interrupt status and vector
typedef struct _IP_PARttlPATT_ISR_STAT {
   USHORT   InterruptStatus;
   USHORT   DirectBits;
   USHORT   RisingBits;
   USHORT   FallingBits;
} IP_PARttlPATT_ISR_STAT, *PIP_PARttlPATT_ISR_STAT;
```

IOCTL_IP_PARttlPATT_SET_MASTER_INT_EN

Function: Sets the Master Interrupt enable on the IP module.
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_PARttlPATT_CLR_MASTER_INT_EN

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_PARttlPATT_SET_BASE_CONFIG

Function: Sets base control register configuration.
Input: IP_PARttlPATT_BASE_CONFIG structure
Output: none
Notes: See the definition of IP_PARttlPATT_BASE_CONFIG below. Bit definitions can be found in the '_Base' section under Register Definitions in the Hardware manual.

```
typedef struct _IP_PARttlPATT_BASE_CONFIG
{
    BOOLEAN   Reset;        // Set to Reset, must be Cleared after for normal
operation
    BOOLEAN   PattIntEn;    // Set to enable Pattern Function Interrupt
    BOOLEAN   RxAflIntEn;   // Set to enable Rx FIFO Almost Full Interrupt
    BOOLEAN   RxEnable;     // Set to enable reception of Data In into Rx
FIFO based on External Clock
    BOOLEAN   LoopBack;     // Set to enable TX-RX FIFO loop-back
    BOOLEAN   TestIO;       // Set to enable Test Registers onto output bus
} IP_PARttlPATT_BASE_CONFIG, * PIP_PARttlPATT_BASE_CONFIG;
```

IOCTL_IP_PARttlPATT_GET_BASE_CONFIG

Function: Returns the base control configuration.
Input: none
Output: IP_PARttlPATT_BASE_CONFIG structure
Notes: See the definition of IP_PARttlPATT_BASE_CONFIG above. Bit definitions can be found in the '_Base' section under Register Definitions in the Hardware manual.

IOCTL_IP_PARttlPATT_GET_STATUS

Function: Returns the status bits in the status register.
Input: none
Output: USHORT
Notes:. Bit definitions can be found in the in the Hardware manual. The grouped Rising, Falling, and Filtered Data interrupt requests are available in this register. i.e. if any Rising Status bit and associated interrupt enable bit are set, the Rising status is set.

IOCTL_IP_PARttlPATT_GET_REVISION

Function: Returns the Module FLASH minor and major revisions.
Input: None
Output: USHORT
Notes: See the definition of Bit definitions can be found under the in the Hardware manual. Repeated here: 15-8 = Major, 7-0 = Minor.

IOCTL_IP_PARttlPATT_SET_SLOTSWITCH

Function: Write data to the SlotSwitch register.
Input: USHORT
Output: none
Notes: Definition can be found in the in the Hardware manual. Used to store the IP location by carrier driver during initialization. Later read by IP driver and stored into a structure. User R/W without consequence.

IOCTL_IP_PARttlPATT_GET_SLOTSWITCH

Function: Read data from the SlotSwitch register.
Input: none
Output: USHORT
Notes: Definition can be found in the Hardware manual.

IOCTL_IP_PARttlPATT_GET_IP_ID

Function: Returns IP module information.
Input: None
Output: IP-IDENTITY structure
Notes: See the definition of I IP_IDENTITY below.

```
typedef struct _IP_IDENTITY {
   UCHAR    IpManuf;
   UCHAR    IpModel;
   UCHAR    IpRevision;
   UCHAR    IpCustomer;
   USHORT   IpVersion;
} IP_IDENTITY, *PIP_IDENTITY;
```

IOCTL_IP_PARttlPATT_SET_DATA_OUT

Function: Write a value to the Tx Data registers.
Input: USHORT
Output: none
Notes: Definition can be found in the in the Hardware manual. 15-0 correspond to the IO bits 47-32. When the corresponding Direction bit is also set, the state of the bit is driven.

IOCTL_IP_PARttlPATT_GET_DATA_OUT

Function: Read from the Tx Data register.
Input: none
Output: USHORT value of Tx Data Register
Notes: Definition can be found in the in the Hardware manual.

IOCTL_IP_PARttlPATT_SET_DIR

Function: Write a value to the direction register.
Input: USHORT
Output: none
Notes: Definition can be found in the '_Direction' section under Register Definitions in the Hardware manual. 15-0 correspond to the IO bits 47-32.

IOCTL_IP_PARttlPATT_GET_DIR

Function: Read from the direction register.
Input: none
Output: USHORT
Notes: Definition can be found in the '_Direction' section under Register Definitions in the Hardware manual.

IOCTL_IP_PARttlPATT_SET_POLARITY

Function: Write data to the Polarity register.
Input: USHORT
Output: none
Notes: Definition can be found in the '_Polarity' section under Register Definitions in the Hardware manual. 15-0 correspond to the IO bits 47-32.

IOCTL_IP_PARttlPATT_GET_POLARITY

Function: Read data from the Polarity register.
Input: none
Output: USHORT
Notes: Definition can be found in the '_Polarity' section under Register Definitions in the Hardware manual.

IOCTL_IP_PARttlPATT_SET_EDGE_LEVEL

Function: Write data to the EdgeLevel register.
Input: USHORT
Output: none
Notes: Definition can be found in the '_EdgeLevel' section under Register Definitions in the Hardware manual. 15-0 correspond to the IO bits 47-32 Select Edge or Level Processing.

IOCTL_IP_PARttIPATT_GET_EDGE_LEVEL

Function: Read data from the EdgeLevel register.
Input: none
Output: USHORT
Notes: Definition can be found in the '_EdgeLevel' section under Register Definitions in the Hardware manual.

IOCTL_IP_PARttIPATT_SET_INT_EN

Function: Write data to the Interrupt Enable register.
Input: USHORT
Output: none
Notes: Definition can be found in the '_IntEn' section under Register Definitions in the Hardware manual. 15-0 correspond to the IO bits 47-32. Select Interrupt Enabled for a particular IO bit.

IOCTL_IP_PARttIPATT_GET_INT_EN

Function: Read data from the Interrupt Enable register.
Input: none
Output: USHORT
Notes: Definition can be found in the '_IntEn section under Register Definitions in the Hardware manual.

IOCTL_IP_PARttIPATT_READ_DIRECT

Function: Read data from the DataIo register.
Input: none
Output: USHORT
Notes: Definition can be found in the '_ReadDirect section under Register Definitions in the Hardware manual. Direct read of IO

IOCTL_IP_PARttIPATT_SET_COS_RISING_STAT

Function: Write data to the COS Rising Status register.
Input: USHORT
Output: none
Notes: Definition can be found in the '_CosRisingSt' section under Register Definitions in the Hardware manual. 15-0 correspond to the IO bits 47-32. Write to clear bits held in the read side of the register

IOCTL_IP_PARttlPATT_GET_COS_RISING_STAT

Function: Read data from the COS Rising Status register.
Input: none
Output: USHORT
Notes: Definition can be found in the '_CosRisingSt' section under Register Definitions in the Hardware manual. Read returns status from Rising COS. See COS Rising Enable register.

IOCTL_IP_PARttlPATT_SET_COS_FALLING_STAT

Function: Write data to the COS Falling Status register.
Input: USHORT
Output: none
Notes: Definition can be found in the '_CosFallingSt' section under Register Definitions in the Hardware manual. 15-0 correspond to the IO bits 47-32. Write to clear bits held in the read side of the register

IOCTL_IP_PARttlPATT_GET_COS_FALLING_STAT

Function: Read data from the COS Falling Status register.
Input: none
Output: USHORT
Notes: Definition can be found in the '_CosFallingSt' section under Register Definitions in the Hardware manual. Read returns status from Rising COS. See COS Falling Enable register.

IOCTL_IP_PARttlPATT_SET_COS_RISING_EN

Function: Write data to the COS Rising Enable register.
Input: USHORT
Output: none
Notes: Definition can be found in the '_CosRisingEn' section under Register Definitions in the Hardware manual. 15-0 correspond to the IO bits 47-32. Write to enable capture of Rising transitions of IO lines. See CosRisingSt and HalfDiv.

IOCTL_IP_PARttlPATT_GET_COS_RISING_EN

Function: Read data from the COS Rising Enable register.
Input: none
Output: USHORT
Notes: Definition can be found in the '_CosRisingEn' section under Register Definitions in the Hardware manual. Read returns register value from Rising COS Enable.

IOCTL_IP_PARttlPATT_SET_COS_FALLING_EN

Function: Write data to the COS Falling Enable register.
Input: USHORT
Output: none
Notes: Definition can be found in the '_CosFallingEn' section under Register Definitions in the Hardware manual. 15-0 correspond to the IO bits 47-32. Write to enable capture of Falling transitions of IO lines. See CosFallingSt and HalfDiv.

IOCTL_IP_PARttlPATT_GET_COS_FALLING_EN

Function: Read data from the COS Falling Enable register.
Input: none
Output: USHORT
Notes: Definition can be found in the '_CosFallingEn' section under Register Definitions in the Hardware manual. Read returns register value from Falling COS Enable.

IOCTL_IP_PARttlPATT_READ_FILTERED

Function: Read data from the Filtered Data register.
Input: none
Output: USHORT
Notes: Definition can be found in the '_ReadFiltered section under Register Definitions in the Hardware manual. Read of IO data after EdgeLevel and Polarity settings applied. i.e. if set to Edge that bit is masked out of this register. If Polarity is set the bit is inverted compared to the IO bit.

IOCTL_IP_PARttlPATT_SET_COSHALFDIV

Function: Write a value to the HalfDiv register.
Input: USHORT
Output: none
Notes: Definition can be found in the '_HalfDiv' section under Register Definitions in the Hardware manual. Used to select the clock rate for the COS. 100 MHz reference is divided based on this register x2.

IOCTL_IP_PARttlPATT_GET_COSHALFDIV

Function: Reads from the HalfDiv register.
Input: none
Output: USHORT
Notes: Definition can be found in the '_HalfDiv' section under Register Definitions in the Hardware manual.

IOCTL_IP_PARttlPATT_SET_COMMAND

Function: Write a value to the Pattern interface control register.
Input: IP_PARTTLPATT_COMMAND
Output: none
Notes: Set the Mode of operation plus some operational parameters. Driver parses and applies to the control register. See HW manual for details.

IOCTL_IP_PARttlPATT_GET_COMMAND

Function: Reads from the Command Registers.
Input: none
Output: IP_PARTTLPATT_COMMAND
Notes:

```
typedef struct _IP_PARttlPATT_COMMAND
{
    BOOLEAN     ClockOut;    // Set to enable Clock onto GPIO 15
    BOOLEAN     ClockInv;    // Set to invert Clock [Falling edge valid] if
enabled
    TRIG_SEL    Trig;        // Select Rising, Falling, Both, No external
trigger  Uses bit GPIO bit 14
    BOOLEAN     AutoClr;     // Set to enable clearing start bit at end of
programmed sequence
    BOOLEAN     Enable;      // Set to enable operation with the mode selected
and options set
    TYPE_SEL    Type;        // Select Pattern Type
} IP_PARttlPATT_COMMAND, * PIP_PARttlPATT_COMMAND;

typedef enum _TYPE_SEL {
   UserPattern,
   RisingRamp,
   FallingRamp,
   Pyramid,
   InvertPyramid,
   SquareWave,
   Trapezoidal,
   ShiftUp,
   InValid  // do not select, if set on read invalid type read from HW
} TYPE_SEL, * PTYPE_SEL;

typedef enum _TRIG_SEL {
    None,
    Rising,
    Falling,
    Both
} TRIG_SEL, * PTRIG_SEL;
```

IOCTL_IP_PARttlPATT_GET_DATA_CNT

Function: Reads the current count for the Pattern Data FIFO
Input: none
Output: USHORT
Notes: The Pattern Data FIFO is 4095 x 32. The count represents the number of data stored for User Mode transmission or loop-back testing.

IOCTL_IP_PARttlPATT_TEST_SET_0

Function: Write a value to the Test 0 register.
Input: USHORT
Output: none
Notes: The contents of this register are sent to the IO if the TestIoEn is set in the base control register. If enabled the "Get" command should return the same value as written here. Use without loop-back.

IOCTL_IP_PARttlPATT_TEST_GET_0

Function: Reads from the IO segment 15-0
Input: none
Output: USHORT
Notes: Can read at any time. If used in conjunction with the SET command – loop-back testing can be performed. If used without the TestIoEn set the current bus status is returned.

IOCTL_IP_PARttlPATT_TEST_SET_1

Function: Write a value to the Test 1 register.
Input: USHORT
Output: none
Notes: The contents of this register are sent to the IO if the TestIoEn is set in the base control register. If enabled the "Get" command should return the same value as written here. Use without loop-back.

IOCTL_IP_PARttlPATT_TEST_GET_1

Function: Reads from the IO segment 31-16
Input: none
Output: USHORT
Notes: Can read at any time. If used in conjunction with the SET command – loop-back testing can be performed. If used without the TestIoEn set the current bus status is returned.

IOCTL_IP_PARttlPATT_LOAD_PATTERN

Function: Write a value to the Pattern FIFO
Input: ULONG
Output: none
Notes: The contents of this register are written to the Pattern Storage FIFO to support User Patterns.

IOCTL_IP_PARttlPATT_READ_PATTERN

Function: Reads from the load registers – not from the FIFO
Input: none
Output: ULONG
Notes: Can read at any time. Can be used for Self-Test. Not normally used in operation.

IOCTL_IP_PARttlPATT_SET_PATTHALFDIV

Function: Write a value to the Pattern Clock Divider
Input: UHORT
Output: none
Notes: The contents of this register are used to control the frequency of operation for Pattern Generation. 100 MHz divided by N+1 and then divided by 2 or 4 depending on Type selected. See HW manual for more information

IOCTL_IP_PARttlPATT_GET_PATTHALFDIV

Function: Reads from the Pattern Clock Divider register
Input: none
Output: USHORT
Notes:

IOCTL_IP_PARttlPATT_WRITEFILE_STD

Function: Move an Array to the Pattern FIFO
Input: TRANS_MULT_PCI
Output: none
Notes: Driver converts Array and Count into multiple writes to HW. 256 max count

```
typedef struct _TRANS_MULT_PCI
{
    USHORT   Count; // number of locations within array to use
    ULONG    Data[TransMultDataSizePci]; // array to use
} TRANS_MULT_PCI, * PTRANS_MULT_PCI;
```

IOCTL_IP_PARttlPATT_READFILE_STD

Function: Driver Fills array with Count reading from Input FIFO
Input: TRANS_MULT_PCI
Output: TRANS_MULT_PCI
Notes: can be used with FIFO loop-back for self-test or external input.

IOCTL_IP_PARttlPATT_WRITEFILE_ENH

Function: Move an Array to the Pattern FIFO
Input: TRANS_MULT_PCIe
Output: none
Notes: Driver converts Array and Count into multiple writes to HW. 512 max count

```
typedef struct _TRANS_MULT_PCIe
{
    USHORT   Count; // number of locations within array to use
    ULONG    Data[TransMultDataSizePcie]; // array to use
} TRANS_MULT_PCIe, * PTRANS_MULT_PCIe;
```

IOCTL_IP_PARttlPATT_READFILE_ENH

Function: Driver Fills array with Count reading from Input FIFO
Input: TRANS_MULT_PCIe
Output: TRANS_MULT_PCIe
Notes: can be used with FIFO loop-back for self-test or external input.

IOCTL_IP_PARttlPATT_LOAD_START

Function: Write a value to the Pattern Start Register
Input: ULONG
Output: none
Notes: Sets the initial pattern for Pattern Generation Modes other than User.

IOCTL_IP_PARttlPATT_READ_START

Function: Reads from the Start registers
Input: none
Output: ULONG
Notes:

IOCTL_IP_PARttlPATT_LOAD_STOP

Function: Write a value to the Pattern Stop Register
Input: ULONG
Output: none
Notes: Sets the Terminal pattern for Pattern Generation Modes other than User.

IOCTL_IP_PARttlPATT_READ_STOP

Function: Reads from the Stop registers
Input: none
Output: ULONG
Notes:

IOCTL_IP_PARttlPATT_LOAD_COUNT

Function: Write a value to the Pattern Count Register
Input: ULONG
Output: none
Notes: Sets the number of cycles to execute before stopping. "0" means infinite.

IOCTL_IP_PARttlPATT_READ_COUNT

Function: Reads from the Count registers
Input: none
Output: ULONG
Notes:

IOCTL_IP_PARttlPATT_READ_RxFIFO

Function: Reads from the Data Storage FIFO
Input: none
Output: ULONG
Notes: Returns next value from Rx FIFO. Will return with previous value if FIFO is empty when read.

IOCTL_IP_PARttlPATT_READ_RxFIFOCnt

Function: Reads the current count for the Rx FIFO
Input: none
Output: USHORT
Notes: The Command FIFO is 4095x32. Read to determine how many to read with READFILE command. Alternatively, Status can be polled to determine if data is present.

IOCTL_IP_PARttlPATT_LOAD_SLOPE

Function: Write a value to the Pattern Slope Register
Input: USHORT
Output: none
Notes: Sets the point-to-point adjustment value. Used with Ramp and related types.

IOCTL_IP_PARttlPATT_READ_SLOPE

Function: Reads from the Slope register
Input: none
Output: USHORT
Notes:

IOCTL_IP_PARttlPATT_LOAD_HCOUNT

Function: Write a value to the Pattern Horizontal Count Register
Input: USHORT
Output: none
Notes: Sets the number of Horizontal points to use in waveforms with this component. Trapezoidal for example.

IOCTL_IP_PARttlPATT_READ_HCOUNT

Function: Reads from the Horizontal Count register
Input: none
Output: USHORT
Notes:

IOCTL_IP_PARttlPATT_LOAD_RXFIFOAFL

Function: Write a value to the RX FIFO Almost Full Register
Input: USHORT
Output: none
Notes: When the number of data in the Rx FIFO is Greater Than the value in the register the Rx FIFO AFL bit is set. Cleared otherwise. If the interrupt is enabled can provide automatic request to perform READFILE with known amount in FIFO based on programmed value. Depending on the Rx Clock rate allocate the level to provide enough “time” to read the data before overflowing and enough data to allow efficient transfer.

IOCTL_IP_PARttlPATT_READ_RXFIFOAFL

Function: Reads from the RX FIFO Almost Full Register
Input: none
Output: USHORT
Notes:

Warranty and Repair

Please refer to the warranty page on our website for the current warranty offered and options.
https://www.dyneng.com/warranty.html

Service Policy

The driver has gone through extensive testing, and while not infallible, problems experienced will likely be “cockpit error” rather than an error with the driver. We will work with you to determine the cause of the issue. If the effort is more than a quick conversation, we will offer a support contract. We can write updates to the driver to add features, create middleware etc.

Support

The software described in this manual is provided at no cost to clients who have purchased the corresponding hardware. Minimal support is included along with the documentation. For help with integration into your project please contact sales@dyneng.com for a support contract. Several options are available. With a contract in place Dynamic Engineers can help with system debugging, special software development, or whatever you need to get going.

For Service Contact:

Customer Service Department
Dynamic Engineering
150 DuBois Street, Suite B/C
Santa Cruz, CA 95060
831-457-8891
support@dyneng.com