

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

IpParTape

Driver Documentation

Developed with Windows Driver Foundation Ver1.9

Manual Revision A

Corresponding Hardware: Revision A

10-2009-0203

FLASH revision A1

IP-Parallel-Tape

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

©2015-2107 by Dynamic Engineering.
Trademarks and registered trademarks are owned by their
respective manufactures.

This document contains information of proprietary interest to Dynamic Engineering. It has been supplied in confidence and the recipient, by accepting this material, agrees that the subject matter will not be copied or reproduced, in whole or in part, nor its contents revealed in any manner or to any person except to meet the purpose for which it was delivered.

Dynamic Engineering has made every effort to ensure that this manual is accurate and complete. Still, the company reserves the right to make improvements or changes in the product described in this document at any time and without notice. Furthermore, Dynamic Engineering assumes no liability arising out of the application or use of the device described herein.

The electronic equipment described herein generates, uses, and can radiate radio frequency energy. Operation of this equipment in a residential area is likely to cause radio interference, in which case the user, at his own expense, will be required to take whatever measures may be required to correct the interference.

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.

This product has been designed to operate with IP Module carriers and compatible user-provided equipment. Connection of incompatible hardware is likely to cause serious damage.



Table of Contents

INTRODUCTION	4
Driver Installation	6
Windows 7 Installation	6
Driver Startup	7
IO Controls	7
IOCTL_IP_PAR_TAPE_GET_INFO	8
IOCTL_IP_PAR_TAPE_SET_IP_CONTROL	8
IOCTL_IP_PAR_TAPE_GET_IP_STATE	9
IOCTL_IP_PAR_TAPE_SET_BASE_CONFIG	9
IOCTL_IP_PAR_TAPE_GET_BASE_CONFIG	10
IOCTL_IP_PAR_TAPE_WRITE_MEM_WORD	10
IOCTL_IP_PAR_TAPE_INIT_MEM_READ	10
IOCTL_IP_PAR_TAPE_GET_MEM_DATA	10
IOCTL_IP_PAR_TAPE_SET_CLOCK_CONFIG	10
IOCTL_IP_PAR_TAPE_GET_CLOCK_CONFIG	11
IOCTL_IP_PAR_TAPE_SET_INT_EN	11
IOCTL_IP_PAR_TAPE_GET_INT_EN	11
IOCTL_IP_PAR_TAPE_SET_EDGE_LEVEL	11
IOCTL_IP_PAR_TAPE_GET_EDGE_LEVEL	11
IOCTL_IP_PAR_TAPE_SET_POLARITY	12
IOCTL_IP_PAR_TAPE_GET_POLARITY	12
IOCTL_IP_PAR_TAPE_READ_DIRECT	12
IOCTL_IP_PAR_TAPE_READ_FILTERED	12
IOCTL_IP_PAR_TAPE_GET_STATUS	12
IOCTL_IP_PAR_TAPE_REGISTER_EVENT	13
IOCTL_IP_PAR_TAPE_ENABLE_INTERRUPT	13
IOCTL_IP_PAR_TAPE_DISABLE_INTERRUPT	13
IOCTL_IP_PAR_TAPE_FORCE_INTERRUPT	13
IOCTL_IP_PAR_TAPE_SET_VECTOR	14
IOCTL_IP_PAR_TAPE_GET_VECTOR	14
IOCTL_IP_PAR_TAPE_GET_ISR_STATUS	14
IOCTL_IP_PAR_TAPE_SET_OUT_DATA	15
IOCTL_IP_PAR_TAPE_GET_OUT_DATA	15
WARRANTY AND REPAIR	16
Service Policy	16
Support	16
For Service Contact:	16



Introduction

The IP-PARALLEL-TAPE 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-PARALLEL-TAPE driver package has two parts. The driver is installed into the Windows® OS, and the User Application “UserApp” 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 “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 test 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-TAPE board is recognized by the IP Carrier Driver, the carrier driver will start the IP-PARALLEL-TAPE 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-TAPE with the carrier switch setting and the slot number of the IP-PARALLEL-TAPE device. From within the IP-PARALLEL-TAPE 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-PARALLEL-TAPE user manual (also referred to as the hardware manual).

Driver Installation

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

IpParTapePublic.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 IpParTape 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, IpParTape.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 IpParTape 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 `IpParTapePublic.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 `IpParTape` 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 IpParTape driver are described below:

IOCTL_IP_PAR_TAPE_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_PAR_TAPE_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_TAPE_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_TAPE_SET_BASE_CONFIG

Function: Sets the Read and Write parity definitions.

Input: IP_PAR_IO_BASE_CONFIG structure

Output: none

Notes: Controls whether odd or even parity is used in read and write transactions. See the definition of IP_PAR_IO_BASE_CONFIG below. Bit definitions can be found under the 'Base_CNTRL' section under Register Definitions in the Hardware manual.

```
typedef struct _IP_PAR_IO_BASE_CONFIG
{
    BOOLEAN    OddReadParity;
    BOOLEAN    OddWriteParity;
} IP_PAR_IO_BASE_CONFIG, *PIP_PAR_IO_BASE_CONFIG;
```

IOCTL_IP_PAR_TAPE_GET_BASE_CONFIG

Function: Returns the Read and Write parity definitions.

Input: none

Output: IP_PAR_IO_BASE_CONFIG structure

Notes: Returns the read and write parity configuration.

IOCTL_IP_PAR_TAPE_WRITE_MEM_WORD

Function: Writes one data word to the specified memory address.

Input: MEM_STRUCT structure

Output: none

Notes: The MEM_STRUCT structure has two fields, Address and Data. This call writes the Data contents to the specified Address as a background process.

```
typedef struct _MEM_STRUCT
{
    ULONG      Address;
    USHORT     Data;
} MEM_STRUCT, *PMEM_STRUCT;
```

IOCTL_IP_PAR_TAPE_INIT_MEM_READ

Function: Starts a memory read cycle from the specified address.

Input: ULONG

Output: none

Notes: This call starts a read cycle from the specified address as a background process. Once the busy bit in the status register is cleared, the data read can be retrieved with the next IOCTL.

IOCTL_IP_PAR_TAPE_GET_MEM_DATA

Function: Returns the data read in the previous memory read cycle.

Input: none

Output: USHORT

Notes: This call is used in conjunction with the previous call to read data from the tape unit.

IOCTL_IP_PAR_TAPE_SET_CLOCK_CONFIG

Function: Writes a value to the clock control register.

Input: USHORT

Output: none

Notes: Controls the clock divisor, the input clock source, and whether the input clock or the divided clock is selected.



IOCTL_IP_PAR_TAPE_GET_CLOCK_CONFIG

Function: Returns the value from the clock control register.

Input: none

Output: USHORT

Notes: Returns the clock divisor, the input clock source, and the output clock select control bits.

IOCTL_IP_PAR_TAPE_SET_INT_EN

Function: Writes values to the interrupt enable registers.

Input: IO_BITS structure

Output: none

Notes: This call defines the mask of which of the 48 input lines will be enabled to cause an interrupt when the specified conditions are met (1 = enabled, 0 = disabled).

```
typedef struct _IO_BITS
{
    ULONG    LoWord;
    ULONG    HiWord;
} IO_BITS, *PIO_BITS;
```

IOCTL_IP_PAR_TAPE_GET_INT_EN

Function: Returns the values of the interrupt enable registers.

Input: none

Output: IO_BITS structure

Notes:

IOCTL_IP_PAR_TAPE_SET_EDGE_LEVEL

Function: Writes values to the edge/level registers.

Input: IO_BITS structure

Output: none

Notes: Determines whether the interrupt for each of the input lines will respond to a static logic level or a transition between levels (1 = edge, 0 = level).

IOCTL_IP_PAR_TAPE_GET_EDGE_LEVEL

Function: Returns the values from the edge/level registers.

Input: none

Output: IO_BITS structure

Notes:



IOCTL_IP_PAR_TAPE_SET_POLARITY

Function: Writes values to the polarity registers.

Input: IO_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, 0 = non-inverted).

IOCTL_IP_PAR_TAPE_GET_POLARITY

Function: Returns values from the polarity registers.

Input: none

Output: IO_BITS structure

Notes:

IOCTL_IP_PAR_TAPE_READ_DIRECT

Function: Reads the direct input data.

Input: none

Output: IO_BITS structure

Notes: This call reads the raw real-time input data from the 48 TTL input lines.

IOCTL_IP_PAR_TAPE_READ_FILTERED

Function: Reads the filtered input data registers.

Input: none

Output: IO_BITS 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. Reading these registers clears the latched bits.

IOCTL_IP_PAR_TAPE_GET_STATUS

Function: Returns the status bits in the INT_STAT register.

Input: none

Output: USHORT

Notes: There are only two status bits in this register: Busy, indicating that the Tape interface is active with a read or write cycle, and Parity ok, indicating that the calculated parity of the last read cycle matched the stored parity.

IOCTL_IP_PAR_TAPE_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_TAPE_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_PAR_TAPE_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_PAR_TAPE_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_TAPE_SET_VECTOR

Function: Writes an 8 bit value to the interrupt vector register.

Input: UCHAR

Output: None

Notes: Required when used in non auto-vectorized systems.

IOCTL_IP_PAR_TAPE_GET_VECTOR

Function: Returns the current interrupt vector value.

Input: none

Output: UCHAR

Notes:

IOCTL_IP_PAR_TAPE_GET_ISR_STATUS

Function: Returns the interrupt status, vector read in the last ISR, and the filtered data bits.

Input: none

Output: INT_STAT structure

Notes: The status contains the contents of the INT_STAT register and the FILTERED_DATA register read in the ISR.

```
typedef struct _INT_STAT
{
    USHORT    InterruptStatus;
    USHORT    InterruptVector;
    IO_BITS   FilteredData;
} INT_STAT, *PINT_STAT;
```

IOCTL_IP_PAR_TAPE_SET_OUT_DATA

Function: Writes a value to the TTL output data registers.

Input: IO_BITS structure

Output: none

Notes: This call can only be used with the 'B' or later revision of the IP-Tape PROM. It uses previously unused addresses to implement the IP-Parallel-TTL output data interface in order to allow loop-back testing of the IP-Tape without installing a -TTL PROM. If this call is executed on a board with a rev 'A' PROM, the call will succeed, but no data will be written to any register.

IOCTL_IP_PAR_TAPE_GET_OUT_DATA

Function: Returns the values from the TTL output data registers.

Input: none

Output: IO_BITS structure

Notes: As with the previous call this can only be used with the 'B' or later PROM revision.

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

All information provided is Copyright Dynamic Engineering

