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-QuadUART
IP-QuadUART-485

Windows 10 WDF Driver Documentation

Developed with Windows Driver Foundation Ver1.9

Manual Revision 1p1
Corresponding Hardware: Revision 04
10-2002-0204 10-2002-1702
Firmware revision 08

IpQuadUART

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

Table of Contents

INTRODUCTION **5**

Driver Installation **7**

Windows 10 Installation **7**

Driver Startup **8**

IO Controls **8**

IOCTL_IP_QuadUART_GET_INFO 9
IOCTL_IP_ QuadUART_SET_IP_CONTROL 10
IOCTL_IP_QuadUART_GET_IP_STATE 10
IOCTL_IP_QuadUART_REGISTER_EVENT 11
IOCTL_IP_QuadUART_ENABLE_INTERRUPT 11
IOCTL_IP_QuadUART_DISABLE_INTERRUPT 11
IOCTL_IP_QuadUART_FORCE_INTERRUPT 11
IOCTL_IP_QuadUART_CLR_FORCE_INTERRUPT 12
IOCTL_IP_QuadUART_SET_VECTOR 12
IOCTL_IP_QuadUART_GET_VECTOR 12
IOCTL_IP_QuadUART_GET_ISR_STATUS 12
IOCTL_IP_QuadUART_SET_MASTER_INT_CONFIG 13
IOCTL_IP_QuadUART_CLR_MASTER_INT _CONFIG 13
IOCTL_IP_QuadUART_SET_BASE_CONFIG 13
IOCTL_IP_QuadUART_GET_BASE_CONFIG 13
IOCTL_IP_QuadUART_GET_STATUS 13
IOCTL_IP_QuadUART_GET_IP_ID 14
IOCTL_IP_QuadUART_SET_CHAN_CONFIG 14
IOCTL_IP_QuadUART_GET_CHAN_CONFIG 14
IOCTL_IP_QuadUART_SET_UART_DATA_CONFIG 14
IOCTL_IP_QuadUART_GET_UART_DATA_CONFIG 14
IOCTL_IP_QuadUART_SET_UART_INTEN 15
IOCTL_IP_QuadUART_GET_UART_INTEN 15
IOCTL_IP_QuadUART_SET_UART_MODEM_CONTROL 15
IOCTL_IP_QuadUART_GET_UART_MODEM_CONTROL 15
IOCTL_IP_QuadUART_SET_UART_FLOW_CONTROL_PARAMS 15
IOCTL_IP_QuadUART_SET_UART_FLOW_CONTROL_MODE 16
IOCTL_IP_QuadUART_GET_UART_FLOW_CONTROL_MODE 16
IOCTL_IP_QuadUART_WRITE_UART_DATA_BYTE 16
IOCTL_IP_QuadUART_READ_UART_DATA_BYTE 16
IOCTL_IP_QuadUART_SET_TIMEOUT_CONFIG 16
IOCTL_IP_QuadUART_GET_TIMEOUT_CONFIG 16
IOCTL_IP_QuadUART_RESET_UART 17
IOCTL_IP_QuadUART_CONFIGURE_UART_FIFOS 17
IOCTL_IP_QuadUART_GET_UART_STATUS 17

IOCTL_IP_QuadUART_GET_FIFO_STATUS 17
IOCTL_IP_QuadUART_WRITEFILE 18
IOCTL_IP_QuadUART_READFILE 18
IOCTL_IP_QuadUART_SET_32MHZ 18
IOCTL_IP_QuadUART_SET_8MHZ 18

WARRANTY AND REPAIR 19

Service Policy 19
Support 19

For Service Contact: 19

Introduction

The IpQuadUART driver is a Windows device driver for IP-QuadUART & IP-QuadUART-485 Industry-pack (IP) modules 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 IpQuadUART software package has two parts. The driver for Windows® 10 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.

Both modules use the same driver. The driver reads the device type as part of initialization and uses that code to apply the structure inputs for the port control registers. Three bits have different meanings for the two designs. The structure has both definitions and the driver selects the one to use. User SW can ignore the settings for the card not being used or alternatively use both to have SW compatible with both cards – approach taken in the UserAp reference code.

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 for the version in use.

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 IpQuadUART board is recognized by the IP Carrier Driver, the carrier driver will start the IpQuadUART driver which will create a device object for the board. If more than one is found additional copies of the driver are loaded. **Next Firmware release feature**: The carrier driver will load the info storage register on the IpQuadUART with the carrier switch setting and the slot number of the IpQuadUART device. From within the IpQuadUART 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 IpQuadUART user manual (also referred to as the hardware manual).

Driver Installation

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

IpQuadUART_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. IpQuadUART_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 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 IpBis6Gpio 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 IpQuadUART_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 IpQuadUART device. Examples of how to use the following IOCTLs to perform UART set-up, transmission, reception, ReadMultiple and WriteMultiple are contained in the UserAp code set. Please note: some tests require the use of a loop-back fixture. See HW manual for connections. We use IP-Debug-IO for this purpose. https://www.dyneng.com/ipdbgio.html

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_QuadUART_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_ QuadUART_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_QuadUART_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_QuadUART_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_QuadUART_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_QuadUART_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_QuadUART_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_QuadUART_CLR_FORCE_INTERRUPT

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

IOCTL_IP_QuadUART_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_QuadUART_GET_VECTOR

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

IOCTL_IP_QuadUART_GET_ISR_STATUS

Function: Returns the interrupt status, vector read in the last ISR, and the filtered data bits.
Input: none
Output: _ISR_STAT structure
Notes: The status contains the contents of the Interrupt register, stored interrupt vector, etc. See IpQuadUART_Public.h for structure definition.

IOCTL_IP_QuadUART_SET_MASTER_INT_CONFIG

Function: Sets/Clear Master Interrupt Enable on IP
Input: none
Output: none
Notes: Set the master interrupt enable on IP-QuadUART

IOCTL_IP_QuadUART_CLR_MASTER_INT _CONFIG

Function: Returns the Master Interrupt Enable configuration.
Input: none
Output: none
Notes: Clear the master interrupt enable on IP-QuadUART

IOCTL_IP_QuadUART_SET_BASE_CONFIG

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

IOCTL_IP_QuadUART_GET_BASE_CONFIG

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

IOCTL_IP_QuadUART_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.

IOCTL_IP_QuadUART_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_QuadUART_SET_CHAN_CONFIG

Function: Set Channel operating parameters
Input: IP_QUART_CHAN_CONFIG structure
Output: none
Notes: See Public file for structure. See HW manual for detailed description. This structure has definitions for both IP-QuadUART and IP-QuadUART-485. Both can be applied or just the one for the module you are using. See the UserAp for examples.

IOCTL_IP_QuadUART_GET_CHAN_CONFIG

Function: Get Channel operating parameters
Input: Port
Output: IP_QUART_CHAN_CONFIG structure
Notes: See Public file for structure. See HW manual for detailed description.

IOCTL_IP_QuadUART_SET_UART_DATA_CONFIG

Function: Control Baud rate, data size, parity etc.
Input: UART_DATA_CONFIG
Output: none
Notes: See Public file for structure. Examples in UserAp.

IOCTL_IP_QuadUART_GET_UART_DATA_CONFIG

Function: Read Configuration
Input: port
Output: UART_DATA_CONFIG
Notes: See Public file for structure. Examples in UserAp.

IOCTL_IP_QuadUART_SET_UART_INTEN

Function: Control interrupt capabilities for each port
Input: UART_INT_CONFIG
Output: none
Notes: See Public file for structure. Examples in UserAp.

IOCTL_IP_QuadUART_GET_UART_INTEN

Function: Read Configuration
Input: port
Output: UART_INT_CONFIG
Notes: See Public file for structure. Examples in UserAp.

IOCTL_IP_QuadUART_SET_UART_MODEM_CONTROL

Function: Control internal loop-back plus control signals when used as GPIO bits
Input: UART_MODEM_CONFIG
Output: none
Notes: See Public file for structure. Examples in UserAp. [RTS, DTR etc. for the GPIO]

IOCTL_IP_QuadUART_GET_UART_MODEM_CONTROL

Function: Read current programming for internal loop-back and GPIO
Input: port
Output: UART_MODEM_CONFIG
Notes: See Public file for structure. Examples in UserAp.

IOCTL_IP_QuadUART_SET_UART_FLOW_CONTROL_PARAMS

Function: Trigger levels plus XON, XOFF characters
Input: UART_FLOW_PARAMS
Output: none
Notes: See Public file for structure. Examples in UserAp.

IOCTL_IP_QuadUART_SET_UART_FLOW_CONTROL_MODE

Function: HW, SW, no control modes selected
Input: UART_FLOW_CONFIG
Output:
Notes: See Public file for structure. Examples in UserAp.

IOCTL_IP_QuadUART_GET_UART_FLOW_CONTROL_MODE

Function: Read Configuration
Input: port
Output: UART_FLOW_CONFIG
Notes: See Public file for structure. Examples in UserAp.

IOCTL_IP_QuadUART_WRITE_UART_DATA_BYTE

Function: Write data to UART transmit port [THR]
Input: UART_WRITE_BYTE
Output:
Notes: See Public file for structure. Examples in UserAp.

IOCTL_IP_QuadUART_READ_UART_DATA_BYTE

Function: Read Received Data from UART port [RHR]
Input: port
Output: Data
Notes:

IOCTL_IP_QuadUART_SET_TIMEOUT_CONFIG

Function: Set Time and Pattern for TimeOut control
Input: IP_QUART_TIMEOUT_CONFIG
Output: none
Notes: See Public file for structure.

IOCTL_IP_QuadUART_GET_TIMEOUT_CONFIG

Function: Read Timeout control structure
Input: none
Output: IP_QUART_TIMEOUT_CONFIG
Notes:

IOCTL_IP_QuadUART_RESET_UART

Function: Restore initial set-up to UART
Input: none
Output: none
Notes: This routine performs nearly identical procedure to driver initialization. Resets HW, disables most extended features, interrupts etc.   Launching main.c calls this routine to restart from the same condition each time,.

IOCTL_IP_QuadUART_CONFIGURE_UART_FIFOS

Function: Enables and/or resets any of the UART FIFOs
Input: UART_FIFO_CONFIG
Output: none
Notes: Note: Reset bits are transitory, no need to re-write to clear etc.

IOCTL_IP_QuadUART_GET_UART_STATUS

Function: Refresh UART STATUS structure
Input: Port
Output: UART_STATUS
Notes: See Public file for structure. Examples in UserAp.

IOCTL_IP_QuadUART_GET_FIFO_STATUS

Function: Refresh FIFO STATUS structure
Input: Port
Output: FIFO_STATUS
Notes: See Public file for structure. Examples in UserAp.

IOCTL_IP_QuadUART_WRITEFILE

Function: Write multiple data to specified port
Input: TRANS_MULT structure [quantity, array, port]
Output: none
Notes: Make sure there is room for the desired transfer size [max = 0x80] – the driver will wait for room.

IOCTL_IP_QuadUART_READFILE

Function: Read data from the Interrupt Enable register.
Input: TRANS_MULT_READ
Output: TRANS_MULT
Notes: TRANS_MULT returns with the number of bytes actually read. If the count in the UART is smaller than the count requested, the amount available is read and the count returned.

IOCTL_IP_QuadUART_SET_32MHZ

Function: Set bit in base control register to indicate 32 MHz IP clock in use
Input: none
Output: none
Notes: If operating at 32 MHz this bit should be set to change the timing to the UART. UserAp routines to select 32 and 8 MHz also set and clear this bit.

IOCTL_IP_QuadUART_SET_8MHZ

Function: Clear bit in base control register to indicate 8 MHz IP clock in use
Input: none
Output: none
Notes: If operating at 8 MHz this bit should be cleared to change the timing to the UART. UserAp routines to select 32 and 8 MHz also set and clear this bit.

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

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