DYNAMIC ENGINEERING

150 DuBois, Suite B/C

Santa Cruz, CA 95060
(831) 457-8891
https://www.dyneng.com
sales@dyneng.com
Est. 1988

PMC-OctalUART-232 Software Manual

8 Port UART Interface

Windows 10 WDF Driver Documentation

Developed with Windows Driver Foundation Ver1.9

Manual Revision 1p0

Corresponding Hardware: 10-2018-0201
PMC-OctalUART-232

PMC-OctalUART-232

8-Channel UART Interface

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

Manual Revision A: Revised 4/29/22

Table of Contents

Introduction

The PmcOctalUART driver is a Windows device driver for the PMC-OctalUART-232 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 PmcOctalUART 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.

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 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 PmcOctalUART device is recognized by the system the host will start the PmcOctalUART driver which will create a device object for the board. If more than one is found additional copies of the driver are loaded. From within the PmcOctalUART 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 PMC-OctalUART-232 user manual (also referred to as the hardware manual).

.

Driver Installation

There are several files provided in each driver package. These files include Public.h, PmcOctalUart.inf, pmcoctaluart.cat, PmcOctalUart.sys.

Public.h is the C header file that defines the Application Program Interface (API) for the PmcOctalUart driver. This file is required at compile time by any application that wishes to interface with the drivers, but is not needed for driver installation. In addition, a second .h file is provided with handy definitions for the UARTs. PmcOctalUART_Defs.h. UserAp includes this file and it is incorporated into the g_all.h global include within that package.

Windows 10 Installation

Copy PmcOctalUart.inf, pmcoctaluart.cat, PmcOctalUart.sys, and copy to your preferred medium.

With the PMC-OctalUART-232 hardware installed, power-on the host computer.

- Open the ***Device Manager*** from the control panel.
- Under ***Other devices*** there should be an ***Other PCI Bridge Device****.
- Right-click on the ***Other PCI Bridge Device*** and select ***Update Driver Software***.
- Insert the disk or memory device prepared above in the desired drive.
- Select ***Browse my computer for driver software***.
- Select ***Let me pick from a list of device drivers on my computer***.
- Select ***Next***.
- Select ***Have Disk*** and enter the path to the device prepared above.
- Select ***Next***.
- Select ***Close*** to close the update window.

The system should now display the PmcOctalUart PCI adapter in the Device Manager.

***** If the ***Other PCI Bridge Device*** is not displayed, click on the ***Scan for hardware changes*** icon on the 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 globally unique identifiers (GUID), which are defined in Public.h. See main.c in the PmcOctalUartUserAp project for an example of how to acquire a handle to the device.

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 PmcOctalUART 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 HDEterm68 for this purpose.
https://www.dyneng.com/HDEterm68.html

IO Controls

The drivers use IO Control calls (IOCTLs) to configure the device. IOCTLs refer to a single Device Object, which controls a single board or I/O channel. IOCTLs are called using the Win32 function DeviceIoControl(), 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 PMC-OctalUART-232 driver are described below:

IOCTL_PMC_OctalUART_GET_INFO

Function: Returns the device driver version, design version, design type, user switch value, device instance number and PLL device ID.
Input: None
Output: PDRIVER_UART_DEVICE_INFO structure
Notes: The switch value is the configuration of the 8-bit onboard dipswitch that has been selected by the user (see the board silk screen for bit position and polarity). Instance number is the zero-based device number. See the definition of DRIVER_UART_DEVICE_INFO below. Bit definitions can be found in the 'BaseRev' section under Register Definitions in the Hardware manual.

```
typedef struct _DRIVER_UART_DEVICE_INFO {
   UCHAR    DriverVersion;
   UCHAR    SwitchValue;
   UCHAR    DeviceRevMaj;
   UCHAR    DeviceRevMin;
   UCHAR    InstanceNumber;
} DRIVER_UART_DEVICE_INFO, *PDRIVER_UART_DEVICE_INFO;
```

IOCTL_PMC_OctalUART_SET_UART_CONFIG

Function: Set UART reference clock settings.
Input: UART_BASE_CONFIG
Output: None
Notes: See the definition of UART_BASE_CONFIG below. Definition can be found in the 'UartCntl' section under Register Definitions in the Hardware manual.

```
typedef struct _UART_BASE_CONFIG {
   BOOLEAN     Osc_sel;
} UART_BASE_CONFIG, *PUART_BASE_CONFIG;
```

IOCTL_PMC_OctalUART_GET_UART_CONFIG

Function: Read the UART reference clock settings.
Input: None
Output: UART_BASE_CONFIG
Notes: Returns the values set in the previous call. See the definition of UART_BASE_CONFIG above.

IOCTL_PMC_OctalUART_SET_DATA_CONFIG

Function: Set UART channel baud rate, data word size, parity, and TX break configuration.
Input: UART_DATA_CONFIG
Output: None
Notes: See the definition of UART_DATA_CONFIG below. See the 'Line Control Register' Section of the EXAR XR16C854/854D manual for detailed definitions.

```
typedef struct _UART_DATA_CONFIG {
   UCHAR       Channel;
   USHORT      BaudDiv;
   DATA_WORD   DataWord;
   PARITY      Parity;
   BOOLEAN     TxBreak;
} UART_DATA_CONFIG, *PUART_DATA_CONFIG;
```

IOCTL_PMC_OctalUART_GET_DATA_CONFIG

Function: Read the UART channel baud rate, data word size, parity, and TX break configuration.
Input: Channel (UCHAR)
Output: UART_DATA_CONFIG
Notes: Returns the values set in the previous call. See the definition of UART_DATA_CONFIG above.

IOCTL_PMC_OctalUART_SET_UART_INTEN

Function: Set UART channel interrupt enable configuration.
Input: UART_INT_CONFIG
Output: None
Notes: See the definition of UART_INT_CONFIG below. See the 'Interrupt Enable Register' Section of the EXAR XR16C854/854D manual for detailed definitions.

```
typedef struct _UART_INT_CONFIG {
   UCHAR       Channel;
   BOOLEAN     RxInten;
   BOOLEAN     TxInten;
   BOOLEAN     LineInten;
   BOOLEAN     ModemInten;
   BOOLEAN     XoffInten;
   BOOLEAN     RtsInten;
   BOOLEAN     CtsInten;
} UART_INT_CONFIG, *PUART_INT_CONFIG;
```

IOCTL_PMC_OctalUART_GET_UART_INTEN

Function: Read the UART channel interrupt enable configuration.
Input: Channel (UCHAR)
Output: UART_INT_CONFIG
Notes: Returns the values set in the previous call. See the definition of UART_INT_CONFIG above.

IOCTL_PMC_OctalUART_SET_UART_MODEM_CONTROL

Function: Set UART channel modem/loopback control configuration.
Input: UART_MODEM_CONFIG
Output: None
Notes: See the definition of UART_MODEM_CONFIG below. See the 'Modem Control Register' Section of the EXAR XR16C854/854D manual for detailed definitions.

```
typedef struct _UART_MODEM_CONFIG {
   UCHAR       Channel;
   BOOLEAN     IntrnlLpbk;
   BOOLEAN     Dtr;
   BOOLEAN     Rts;
   BOOLEAN     Op1;
   BOOLEAN     XonAny;
   BOOLEAN     IrModeEn;
   BOOLEAN     BrgPrescaler;
} UART_MODEM_CONFIG, *PUART_MODEM_CONFIG;
```

IOCTL_PMC_OctalUART_GET_UART_MODEM_CONTROL

Function: Read the UART channel modem/loopback control configuration
Input: Channel (UCHAR)
Output: UART_MODEM_CONFIG
Notes: Returns the values set in the previous call. See the definition of UART_MODEM_CONFIG above.

IOCTL_PMC_OctalUART_SET_UART_FLOW_CONTROL_PARAMS

Function: Set UART channel flow control parameters.
Input: UART_FLOW_PARAMS
Output: None
Notes: See the definition of UART_FLOW_PARAMS below. See the 'Xon Character 1', 'Xon Character 2', 'Xoff Character 1', 'Xoff Character 2', and the 'FIFO Trigger Level Reg' Sections of the EXAR XR16C854/854D manual for detailed definitions.

```
typedef struct _TRIG_PARAMS {
   UCHAR    RxTrig;
   UCHAR    TxTrig;
   UCHAR    RxHyst;
} TRIG_PARAMS, *PTRIG_PARAMS;

typedef struct _X_CHARS {
   UCHAR    Xon1;
   UCHAR    Xon2;
   UCHAR    Xoff1;
   UCHAR    Xoff2;
} X_CHARS, *PX_CHARS;

typedef struct _UART_FLOW_PARAMS {
   UCHAR          Channel;
   TRIG_PARAMS    TrigLevels;
   X_CHARS        XChars;
} UART_FLOW_PARAMS, *PUART_FLOW_PARAMS;
```

IOCTL_PMC_OctalUART_SET_UART_FLOW_CONTROL_MODE

Function: Set UART channel flow control configuration.
Input: UART_FLOW_CONFIG
Output: None
Notes: See the definition of UART_FLOW_CONFIG below. See the 'Enhanced Function Register' Section of the EXAR XR16C854/854D manual for detailed definitions.

```
typedef enum _CNTRL_TYPE {
   NONE,
   HW,
   SW
} CNTRL_TYPE, *PCNTRL_TYPE;

typedef enum _HW_FLOW_SEL {
   RTS,
   CTS,
   BOTH
} HW_FLOW_SEL, *PHW_FLOW_SEL;

typedef enum _SW_FLOW_SEL {
   NO,
   X_1,
   X_2,
   X_1_2
} SW_FLOW_SEL, *PSW_FLOW_SEL;

typedef struct _UART_FLOW_CONFIG {
   UCHAR          Channel;
   CNTRL_TYPE     FlowMode;
   HW_FLOW_SEL    HwMode;
   SW_FLOW_SEL    RxMode;
   SW_FLOW_SEL    TxMode;
   BOOLEAN        SpclCharEn;
} UART_FLOW_CONFIG, *PUART_FLOW_CONFIG;
```

IOCTL_PMC_OctalUART_GET_UART_FLOW_CONTROL_MODE

Function: Read the UART channel modem/loopback control configuration
Input: Channel (UCHAR)
Output: UART_FLOW_CONFIG
Notes: Returns the values set in the previous call. See the definition of UART_FLOW_CONFIG above.

IOCTL_PMC_OctalUART_WRITE_UART_DATA_BYTE

Function: Write a data byte to the UART channel.
Input: UART_WRITE_BYTE
Output: None
Notes: See the definition of UART_WRITE_BYTE below. See the 'Transmit Holding Register' Section of the EXAR XR16C854/854D manual for detailed definitions.

```
typedef struct _UART_WRITE_BYTE {
   UCHAR       Channel;
   UCHAR       Byte;
} UART_WRITE_BYTE, *PUART_WRITE_BYTE;
```

IOCTL_PMC_OctalUART_READ_UART_DATA_BYTE

Function: Read a data byte to the UART channel.
Input: Channel (UCHAR)
Output: Data (UCHAR)
Notes: Returns the values set in the previous call. See the 'Receive Holding Register' Section of the EXAR XR16C854/854D manual for detailed definitions.

IOCTL_PMC_OctalUART_RESET_UART

Function: Reset all the UARTs.
Input: None
Output: None
Notes: Resetting UARTs puts them into the mode where FLVL register replaces the SPAD register as described in the introduction above. All other registers, not associated with the FLVL, are reset as the EXAR XR16C854/854D manual describes. See the 'TX/RX FIFO Level Counter Register' Section of the EXAR XR16C854/854D manual for detailed definitions.

IOCTL_PMC_OctalUART_GET_FIFO_STATUS

Function: Returns the status from the FIFO ready ports values for all UARTs channel.
Input: None
Output: FIFO_STATUS
Notes: See the definition of FIFO_STATUS above. Definition can be found in the 'FR1_4' and 'FR5_8' section under Register Definitions in the Hardware manual.

IOCTL_PMC_OctalUART_CONFIGURE_FIFOS

Function: Write a data byte to the UART channel.
Input: UART_FIFO_CONFIG
Output: None
Notes: See the definition of UART_FIFO_CONFIG below. See the 'FIFO Control Register' Section of the EXAR XR16C854/854D manual for detailed definitions.

```
typedef enum _FIFO_CONTROL {
   FF_DIS,
   FF_EN,
   FF_EN_RST_RX,
   FF_EN_RST_TX,
   FF_EN_RST_BOTH
} FIFO_CONTROL, *PFIFO_CONTROL;

typedef struct _UART_FIFO_CONFIG {
   UCHAR          Channel;
   FIFO_CONTROL   FifoConfig[UART_NUM_CHANNELS];
} UART_FIFO_CONFIG, *PUART_FIFO_CONFIG;
```

IOCTL_PMC_OctalUART_GET_UART_STATUS

Function: Returns various status values for a UART channel.
Input: None
Output: UART_STATUS
Notes: See the definition of UART_STATUS above. See the 'Interrupt Status Register', 'Line Status Register', 'Modem Status Register', and the 'FIFO Level Register' Sections of the EXAR XR16C854/854D manual for detailed definitions.

```
typedef struct _UART_STATUS {
   UCHAR       Channel;
   UCHAR       Isr;
   UCHAR       Lsr;
   UCHAR       Msr;
   UCHAR       ID;  // ID of the UART
   UCHAR       Rev; // Revision for the UART
   UCHAR       RxFFCount;
   UCHAR       TxFFCount;
} UART_STATUS, *PUART_STATUS;
```

IOCTL_PMC_OctalUART_GET_STATUS

Function: Returns the status bits in the INT_STATUS register.
Input: None
Output: (ULONG)
Notes: See the definition of UART_BASE_CONFIG above.

IOCTL_PMC_OctalUART_REGISTER_EVENT

Function: Registers an event to be signaled when an interrupt occurs.
Input: Handle to the 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 a user interrupt is serviced. The user interrupt service routine waits on this event, allowing it to respond to the interrupt.

IOCTL_PMC_OctalUART_SET_MASTER_INT_CONFIG

Function: Enables the master interrupt.
Input: None
Output: None
Notes: This command must be run to allow the board to respond to user interrupts. The master interrupt enable is disabled in the driver interrupt service routine when a user interrupt is serviced. Therefore this command must be run after each user interrupt occurs to re-enable it.

IOCTL_PMC_OctalUART_CLR_MASTER_INT_CONFIG

Function: Disables the master interrupt.
Input: None
Output: None
Notes: This call is used when user interrupt processing is no longer desired.

IOCTL_PMC_OctalUART_FORCE_INTERRUPT

Function: Causes a system interrupt to occur.
Input: None
Output: None
Notes: Causes an interrupt to be asserted on the PCI bus as long as the master interrupt is enabled. This IOCTL is used for development, to test interrupt processing.

IOCTL_PMC_OctalUART_CLR_FORCE_INTERRUPT

Function: Clears request bit
Input: None
Output: None
Notes:

IOCTL_PMC_OctalUART_GET_ISR_STATUS

Function: Returns the interrupt status read in the ISR from the last user interrupt.
Input: None
Output: Interrupt status value (UART_INT_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. InterruptStatus shows in which channel the interrupt occurred and IntChanStat shows the state of the ISR register for each channel. See 'Interrupt Status Register' Section of the EXAR XR16C854/854D manual for detailed definitions.

```
typedef struct _UART_INT_STAT {
   ULONG    InterruptStatus;
   UCHAR    IntChanStat[UART_NUM_CHANNELS];
} UART_INT_STAT, *PUART_INT_STAT;
```

IOCTL_PMC_OctalUART_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_PMC_OctalUART_READFILE

Function: Read data from the Interrupt Enable register.
Input: TRANS_MULT
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.

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

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 or e-mail and arrange to work with an engineer. We will work with you to determine the cause of the issue.

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
support@dyneng.com