

Windows & Linux API interface code for PCI/PCIe UARTs' drivers

1.0 INTRODUCTION

PCI and PCIe UARTs will work with standard serial COM ports APIs. However, for advanced features such as Auto RS-485 Half-duplex flow control and controlling MPIOs, special APIs are necessary. This application note will discuss the Windows and Linux software API codes to control the MPIOs and Auto RS-485 Half-duplex flow control.

2.0 WINDOWS API INTERFACE CODE

Exar's PCI/PCIe UART drivers should be first installed for corresponding Windows OS. Then refer to the following sections.

2.1 CONTROLLING MPIOS

The following steps show the method to control the MPIOs.

2.1.1 Header definition file

The definitions shown in Figure 1 below needs to be added to the header file.

FIGURE 1. DEFINITION IN THE HEADER FILE

#include <winioctl.h>

#define FILE_DEVICE_XRPC #define XRPORT_IOCTL_INL #define IOCTL_XRPORT_RE CTL_CODE(FILE_DEVICE_X	DRT 0x00008005 DEX 0x805 GAD_CONFIG_REG (RPORT , \ XRPORT_IOCTL_INDEX + 8, \ METHOD_BUFFERED, \ FILE_ANY_ACCESS)			
#define IOCTL_XRPORT_WF CTL_CODE(FILE_DEVICE_X	RITE_CONFIG_REG (RPORT , \ XRPORT_IOCTL_INDEX + 9, \ METHOD_BUFFERED, \ FILE_ANY_ACCESS)			
typedef struct				
i BYTE bReg; BYTE bData;				
} CONFIG_WRITE, *PCONFIG_WRITE;				

APPLICATION NOTE



AN209

2.1.2 Application source code functions

The functions shown in Figure 2 below needs to be added to the application source code.

FIGURE 2. FUNCTIONS IN THE SOURCE CODE

```
// include the header file which has the above defines
// The application should supply bReg in the range of
// 0x00 - 0x13 respectively for 0x80 - 0x93
// Driver appends the CONFIG REG BASE 0x80 to this register
BOOL cfgWrite( BYTE bReg, BYTE bValue) {
  BOOL Status:
  DWORD cbReturned;
  CONFIG WRITE cfgWrite;
  cfgWrite.bReg = bReg;
  cfgWrite.bData = bValue;
  Status = DeviceloControl( m hPortHandle, //handle that you got from CreateFile
              (DWORD) IOCTL_XRPORT_WRITE_CONFIG_REG,
              &cfgWrite,
              sizeof(CONFIG_WRITE),
              NULL,
              0.
              &cbReturned,
              0);
  if (!Status)
        MessageBox("Error in Writing...!", "Error", MB_OK\MB_ICONERROR); }
  {
  return Status;}
BOOL cfgRead( BYTE bReg, BYTE *pbValue ) {
  BOOL Status;
  DWORD cbReturned;
  Status = DeviceloControl( m hPortHandle, //handle that you got from CreateFile
              IOCTL XRPORT READ CONFIG REG,
              &bReg,
              sizeof(BYTE),
              pbValue,
              sizeof(BYTE),
              &cbReturned,
              0);
  if (!Status)
        MessageBox("Error in Reading...!", "Error", MB_OK|MB_ICONERROR); }
  {
  return Status;}
```



AN209

In the function in **Figure 2**, only the first port handle of the card can be used to access the configuration registers. The configuration register value should be in the range of 0x00-0x13 for PCI and 0x00-0x1B for PCIe UARTs.

2.1.3 Function call

To access the MPIOs, use this code at the appropriate place in your application.

```
FIGURE 3. FUNCTION CALL
```

```
HANDLE m_hPortHandle;
yourfunction()
       if ((hPortHandle = CreateFile("\\\\.\\COM5", // this should be the first port number of the card.
                                              // In your case it may not be COM5
                                 GENERIC READ | GENERIC WRITE,
                                 FILE_SHARE_WRITE,
                                 NULL,
                                 OPEN EXISTING,
                                 FILE ATTRIBUTE NORMAL,
                                 NULL
                                 )) == ((HANDLE)-1))
       {
              return FALSE; // failure
       }
       else
       { // success }
       cfgWrite(0x13, 0x00); // Config reg is offset by 0x80 in the driver, bits set to '0' for output
       // write to MPIOLVL register (cfgWrite (0x10, value) to control the output levels
      // or read to read the current output/input levels
```

APPLICATION NOTE



AN209

2.2 ENABLE RS-485 HALF DUPLEX FLOW CONTROL

To enable RS-485 half duplex flow control, the "xrapi.h" file needs to be included in the project. The "xrapi.h" file can be found in the folder with the driver source code.

FIGURE 4. ENABLE RS-485 HALF-DUPLEX CONTROL MODE

```
After opening and setting up the port using CreateFile and SetCommState, call DeviceIoControl with
IOCTL_XRPORT_SET_RS485, as below:
//START --
COM_PORT_ENHANCED
                            portEnhanced;
DWORD cbReturned;
portEnhanced.RS485 = 1; // 0 for reset
if (!DeviceIoControl( m_hPortHandle,
             (DWORD) IOCTL_XRPORT_SET_RS485,
             &portEnhanced,
             sizeof(COM_PORT_ENHANCED),
             NULL,
             0,
             &cbReturned,
             0
             ))
{
      MessageBox( NULL, "Unable to set RS485", NULL, MB_OK);
//END --
```



AN209

In the Windows 2K/XP (version 4.1.0.0 and newer) and Vista/7 (version 2.2.0.0 and newer) drivers, the RS-485 half-duplex feature can also be enabled (by checking the box next to the "RS-485") through the properties page (see Figure 5).

FIGURE 5. PROPERTIES PAGE

OEM Communications Port (COM144) Properties General Port Settings Driver Details Resources	?×	
Bits per second: 115200		
Data bits: 8		
Parity: None		
Stop bits: 1		
Flow control: None		
RS-485		
Active Low (XR17V35x) (Only if RS-485 is set.)		
Turn Around Time (Only if RS-485 is set.) 0		
Rx FIFO Trigger 51 💌 Tx FIFO Trigger 13 💌		
[Note: PCI UARTs have 64 byte FIFOs. Trigger levels will default to 32 bytes in the driver if trigger level selection is greater than 64.]		
Advanced Restore Defaults		
OK Ca	ancel	



AN209

3.0 LINUX API INTERFACE CODE

The Linux standard serial port driver has support for XR17x15x UARTs in kernel version 2.6.8 and newer (see the "8250_pci.c" file). Exar provides custom drivers for both Linux kernel versions 2.4 and 2.6 for PCI (XR17x15x and XR17V25x) and PCIe (XR17V35x) UARTs. To use some advanced feature like Auto RS-485 half-duplex flow control, custom API codes (see section 3.3 and section 3.4) are necessary.

3.1 LINUX KERNEL VERSION 2.4

If Exar's default Vendor ID and Device ID are used, it is recommended to use the custom driver from Exar's website. If custom Vendor ID and Device ID are used, it is recommended to use the custom driver from Exar's website and modify to the custom Vendor ID and Device ID in the driver. An external EEPROM can be programmed to store the custom Vendor ID and Device ID.

3.2 LINUX KERNEL VERSION 2.6

3.2.1 Linux kernel version 2.6.7 and older

If Exar's default Vendor ID and Device ID are used, it is recommended to use the custom driver from Exar's website. If custom Vendor ID and Device ID are used, it is recommended to use custom driver from Exar's website and modify to the custom Vendor ID and Device ID in the driver. An external EEPROM can be programmed to store the custom Vendor ID and Device ID.

3.2.2 Linux kernel version 2.6.8 and newer

Linux kernel version 2.6.8 and newer have built-in support for Exar's XR17x15x PCI UARTs (see the "8250_pci.c" file) but do not have support for the XR17V25x and XR17V35x UARTs.

3.2.2.1 XR17x15x

If Exar's default Vendor ID and Device ID are used, it is recommended to disable the "8250_pci.c" driver, rebuild the kernel and then use the custom drivers from Exar's website. If custom Vendor ID and Device ID are used, it is recommended to use the custom driver from Exar's website and modify to the custom Vendor ID and Device ID in the driver. An external EEPROM can be programmed to store the custom Vendor ID and Device ID.

3.2.2.2 XR17V25x & XR17V35x

If Exar's default Vendor ID and Device ID are used, it is recommended to use custom driver from Exar's website. If custom Vendor ID and Device ID are used, it is recommended to use custom driver from Exar's website and modify to the custom Vendor ID and Device ID in the driver. An external EEPROM can be programmed to store the custom Vendor ID and Device ID.

See the summary in Table 1.

	LINUX KERNEL VERSION	DEFAULT EXAR VENDOR ID AND DEVICE ID	CUSTOM VENDOR ID AND DEVICE ID
	2.4		
XR17x15x	2.6.1 2.6.7	Use custom driver from Exar's website	
	2.6.8 latest	Disable "8250_pci.c" driver, rebuild kernel and then use the rebuilt driver	Use custom driver from Exar's website and modify VID and DID in custom driver
XR17V25x & XR17V35x	2.4	Use custom driver from Exar's website without any changes	
	2.6.1 2.6.7		
	2.6.8 latest		

TABLE 1: VID AND DID REQUIREMENTS FOR USING LINUX CUSTOM DRIVERS

Linux kernel version 2.4 and 2.6 drivers are available for download from Exar's website. If a driver for a specific kernel version is not available, send an email to uarttechsupport@exar.com.



3.3 ACCESSING MPIOS

To access MPIOs register set, the following code is necessary. For the PCI UARTs, please note that the file handle in the following Figure 6 should be that of the first port of the card.

FIGURE 6. ACCESSING MPIOS DESCRIPTION

```
//Accessing MPIO registers from the user space:
// define these in the app.
   EXAR ioctls */
//#define FIOQSIZE
                          0x5460
#define EXAR READ REG
                                    (FIOQSIZE + 1)
#define EXAR_WRITE_REG
                                    (FIOQSIZE + 2)
struct xrioctl_rw_reg {
   unsigned char reg;
   unsigned char regvalue;
};
//To read MPIO register:
xrioctl_rw_reg input;
input. reg = 0x93; //MPIO register address is different from Windows API
ioctl(fd, EXAR_READ_REG, &input); // fd is the file handle obtained with 'open'
// input.regvalue will have the data read from the reg
//To write to MPIO register:
xrioctl_rw_reg input;
input.reg = 0x93; //MPIO register address is different from Windows API
input.regvalue = 0xFF;
ioctl(fd, EXAR_WRITE_REG, &input); // fd is the file handle obtained with 'open'
```

APPLICATION NOTE



AN209

3.4 ENABLE AUTO RS-485 MODE

To enable auto RS-485 half duplex mode, the following code in Figure 7 is necessary.

FIGURE 7. ENABLE AUTO RS-485 MODE

```
//Enabling RS-485 mode from the user space
// define these in the app.
/* EXAR ioctls */
//#define FIOQSIZE
                           0x5460
#define
        EXAR_READ_REG
                                    (FIOQSIZE + 1)
        EXAR_WRITE_REG
#define
                                     (FIOQSIZE + 2)
struct xrioctl rw reg {
   unsigned char reg;
    unsigned char regvalue;
};
//To enable the auto RS-485 mode, the FCTR bit-5 needs to be set to '1'
//To write to FCTR register:
xrioctl_rw_reg input;
input.reg = 0x8; //FCTR register offset address is 0x8;
ioctl(fd, EXAR_READ_REG, &input); // fd is the file handle obtained with 'open';
                                 // input.reg value will have the data read from the reg
input.regvalue = 0x20; //Set FCTR bit-5 to '1'
ioctl(fd, EXAR_WRITE_REG, &input); //fd is the file handle obtained with 'open'
//To disable the auto RS-485 mode
xrioctl_rw_reg input;
input.reg = 0x8; //FCTR register offset address is 0x8;
ioctl(fd, EXAR_READ_REG, &input); // fd is the file handle obtained with 'open;
                                 //input.reg value will have the data read from the reg
input.regvalue &= 0xdf; // clear FCTR bit-5
ioctl(fd, EXAR_WRITE_REG, &input); // fd is the file handle obtained with 'open'
```



4.0 SUPPORT

If there are any questions, send them to uarttechsupport@exar.com.

NOTICE

EXAR Corporation reserves the right to make changes to the products contained in this publication in order to improve design, performance or reliability. EXAR Corporation assumes no responsibility for the use of any circuits described herein, conveys no license under any patent or other right, and makes no representation that the circuits are free of patent infringement. Charts and schedules contained here in are only for illustration purposes and may vary depending upon a user's specific application. While the information in this publication has been carefully checked; no responsibility, however, is assumed for inaccuracies.

EXAR Corporation does not recommend the use of any of its products in life support applications where the failure or malfunction of the product can reasonably be expected to cause failure of the life support system or to significantly affect its safety or effectiveness. Products are not authorized for use in such applications unless EXAR Corporation receives, in writing, assurances to its satisfaction that: (a) the risk of injury or damage has been minimized; (b) the user assumes all such risks; (c) potential liability of EXAR Corporation is adequately protected under the circumstances.

Copyright 2011 EXAR Corporation

Datasheet December 2011.

Send your UART technical inquiry with technical details to hotline: <u>uarttechsupport@exar.com</u>.

Reproduction, in part or whole, without the prior written consent of EXAR Corporation is prohibited.