串口通讯模块

WD663X硬件手册

品质•服务

(版本V 1.00)

支持产品：

●cPCI -WD663X　●PXI -WD663X　●cPCIE -WD663X

●PXIE -WD663X

成都云索科技有限公司

　微信公众号：cdyunsuo　Tel：028-62655393

目 录

更新版本

硬件使用手册的初始版本是 Ver1.00，微小更改时版本号在尾数加 1，当更新涉及到较为重要的内容时版本号中间的数加 1，当更新涉及到核心内容时，版本号第一位数加 1。手册更新后云索科技不对原发布手册进行更换，也不对使用者单独通知，请关注公司网站信息。

公司网址：Copyright (C)www.ys-amc.com。

版本号	日期	更改内容	备注
Ver1.00	2021 年 10		对应硬件版本号：V1.03

1.函数开发环境

在本手册中描述的函数为 C 和 C++格式。此 API 函数可以使用在 windows 环境下所有 32bit 开发工具和图形化开发工具。

2. 驱动安装

1、在驱动安装之前，要求正确安装硬件。

2、在产品配套光盘的“driver”目录中，您可以找到 WD663X 板卡的驱动。

3、计算机在安装新硬件后，将发现新设备，指定驱动程序从光盘安装。按提示到安装完成。如下图示：

图 2.1

这里选择“从列表或指定位置安装（高级）”选项后，点击下一步（图 2.1）

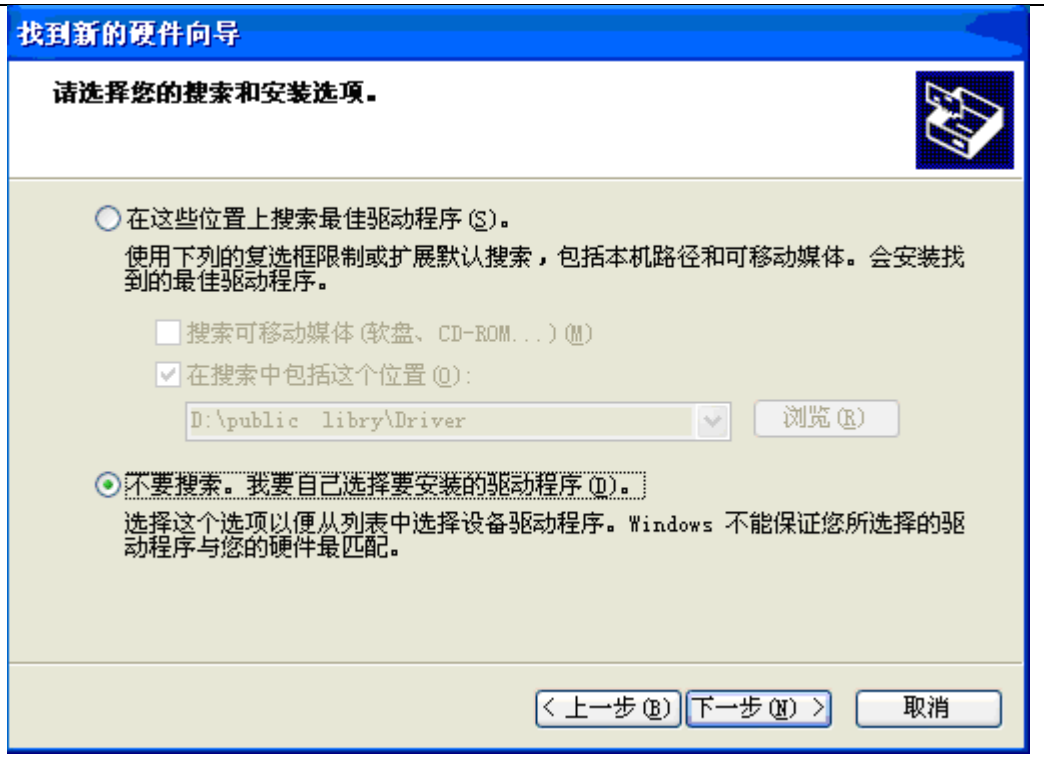


图 2.2

选择“不要搜索。我要自己选择要安装的驱动程序”选项，点击下一步（图2.2）

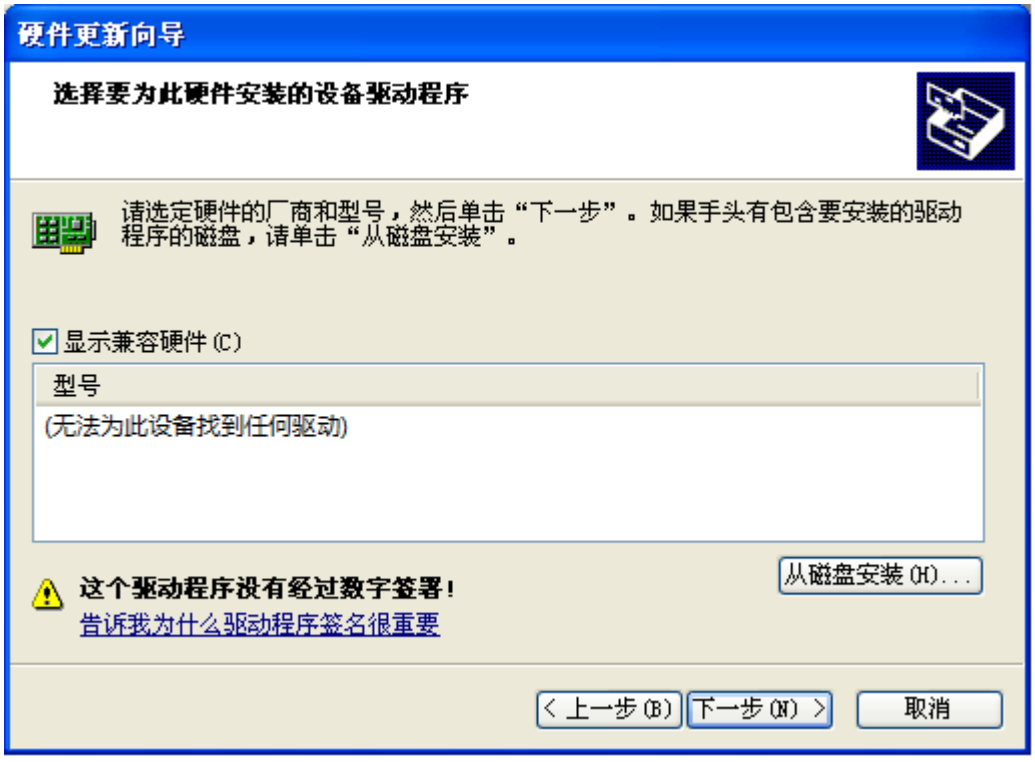


图 2.3

点击从磁盘安装（图2.3）：

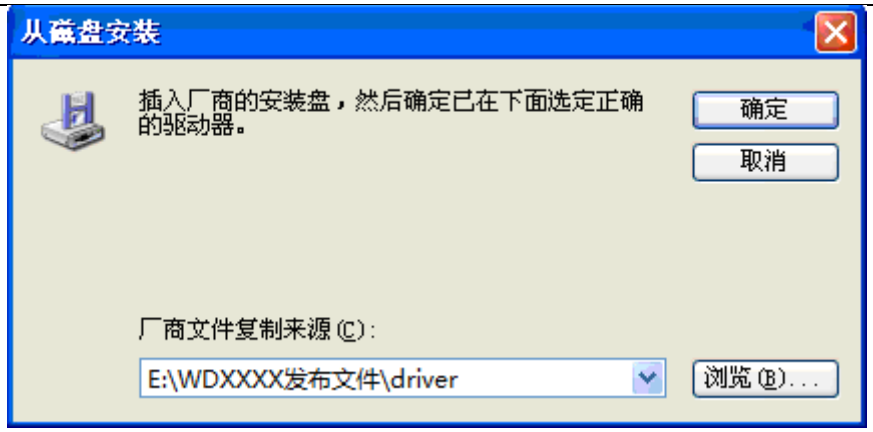


图 2.4

编辑框中为光盘路径:\WD663X\driver（图 2.4），如：E:\ WD663X\driver

点击下一步，直到驱动程序安装成功（图 2.5）。

图 2.5

在完成WD663X板卡驱动安装后，您可以通过计算机系统的“设备管理器”来确认板卡驱动是否正确安装。如果板卡驱动正确安装，您可以在“设备管理器”的设备列表中看到WD663X板卡选项，如图2.6所示。

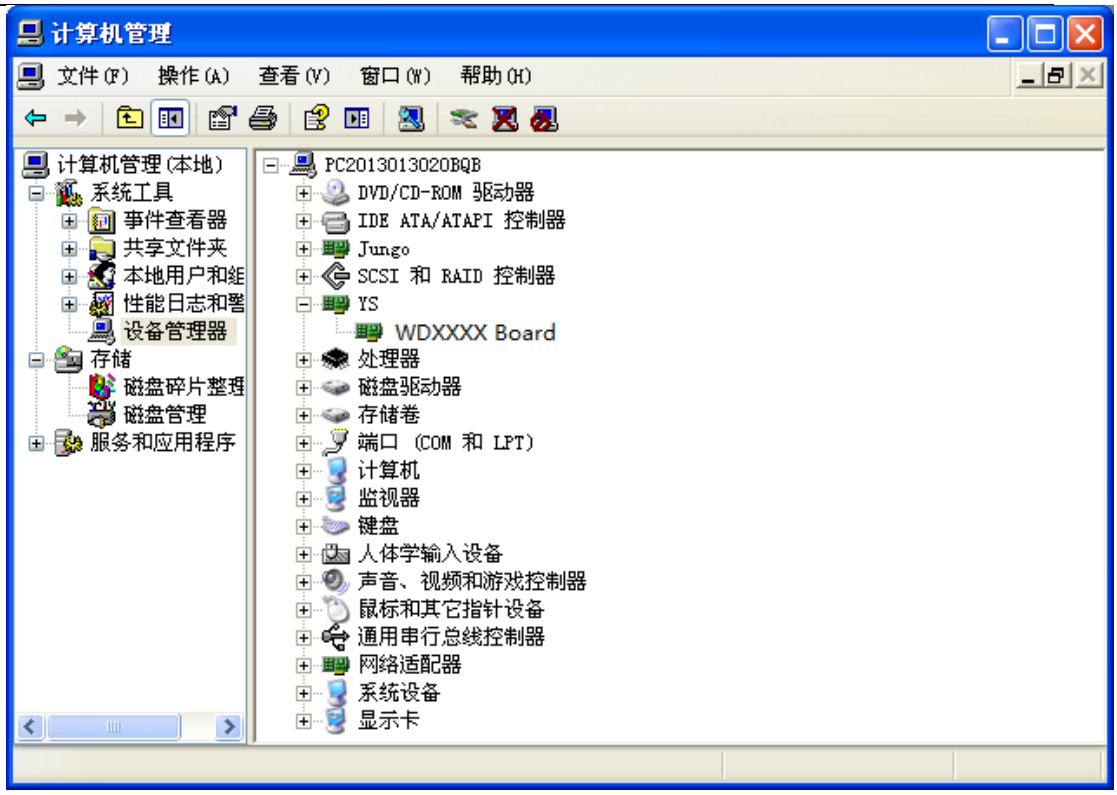

计算机管理
文件(F) 操作(A) 查看(V) 窗口(W) 帮助(H)
计算机管理(本地)
系统工具
事件查看器
共享文件夹
本地用户和组
性能日志和警
设备管理器
存储
磁盘碎片整理
磁盘管理
服务和应用程序
PC20130130200BQB
DVD/CD-ROM 驱动器
IDE ATA/ATAPI 控制器
Jungo
SCSI 和 RAID 控制器
YS
WDXXXX Board
处理器
磁盘驱动器
存储卷
端口 (COM 和 LPT)
计算机
监视器
键盘
人体学输入设备
声音、视频和游戏控制器
鼠标和其它指针设备
通用串行总线控制器
网络适配器
系统设备
显示卡

图 2.6

3. PNP 安装和卸载

3.1 软件安装

打开光盘在 Install 文件夹中点击 setup.exe（图 3.1）；进入安装界面后点击 next(图 3.2)；选择安装目录点击 next；完成安装。

图 3.1

图 3.2

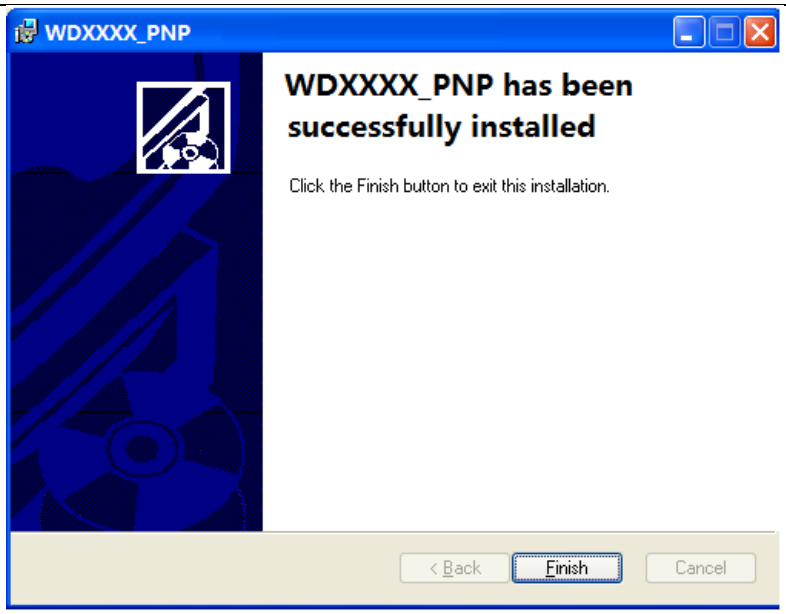


图 3.3

3.2 软件卸载

再次双击 setup.exe 进入卸载界面；依次点击 next，完成软件卸载(图 3.4)。

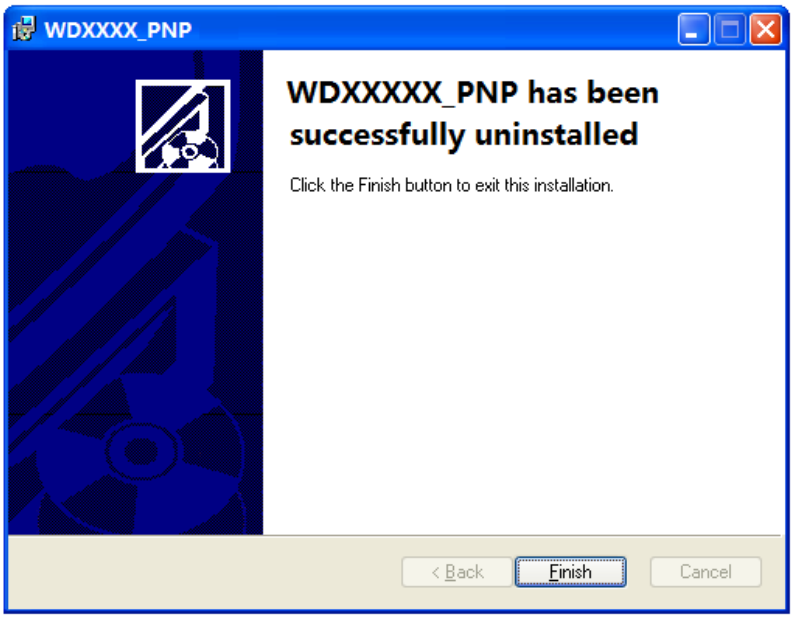


图 3.4

或者点击"计算机-〉控制面板-〉所有控制面板项-〉程序和功能”，右键点击 WD6761_PNP-〉卸载，完成软件卸载

4. PNP 界面操作

模块连接

运行 WD663X.exe 文件，弹出模块操作界面，如下图

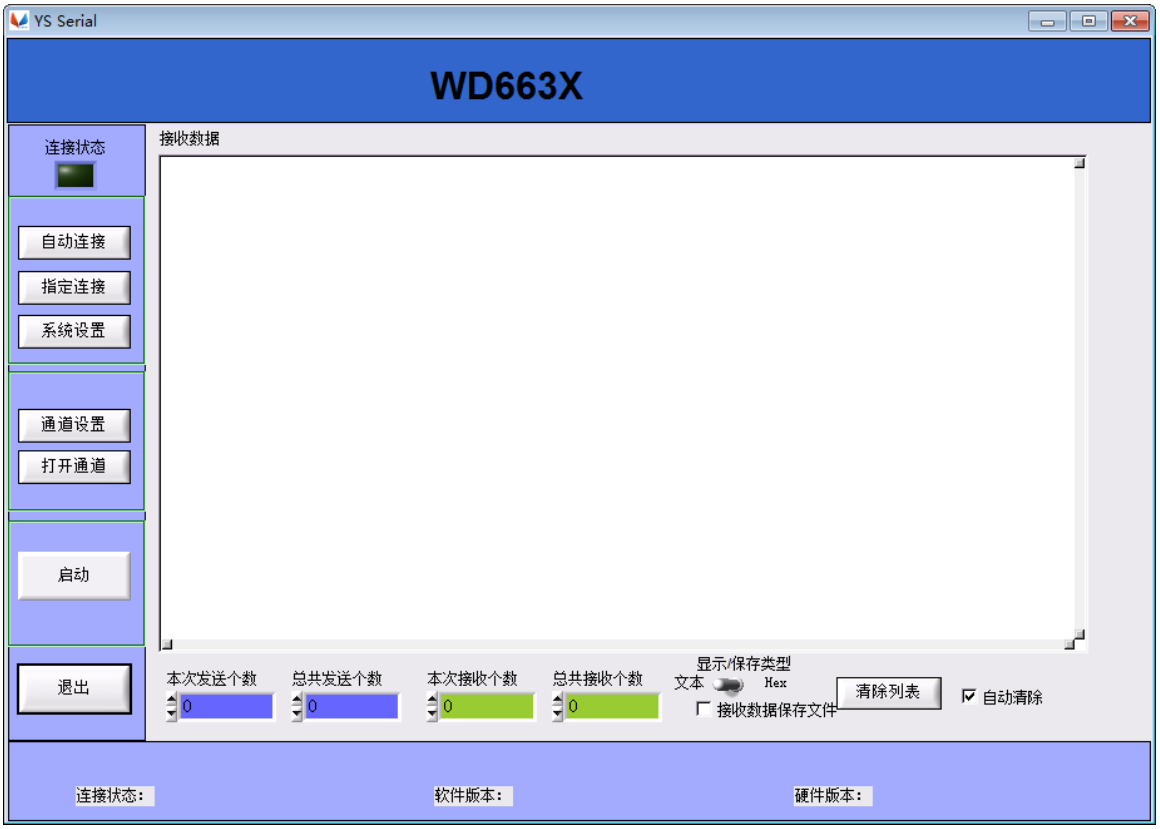


图 4.1 PNP 操作界面

模块链接区域如下图所示：

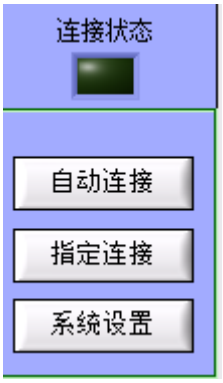


图 4.2 模块链接操作区域

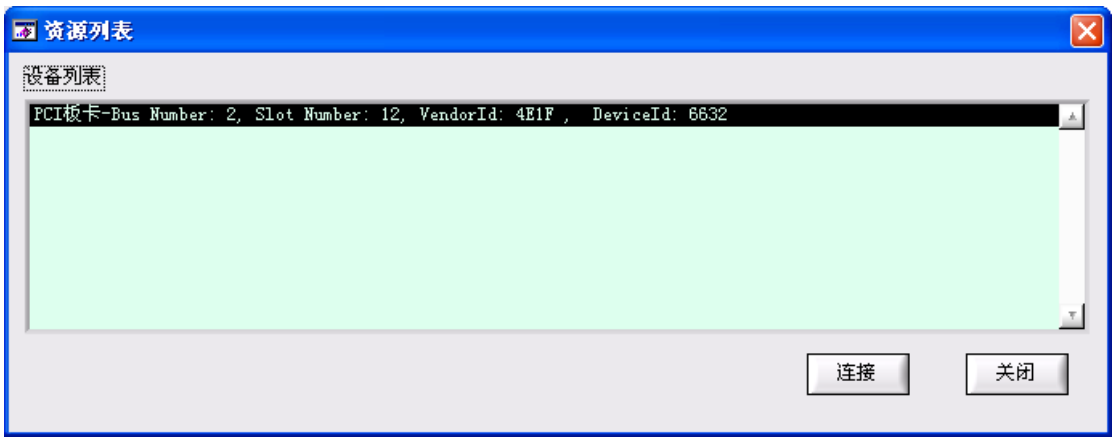


图 4.3 查询到的设备资源区域

自动连接：点击“自动连接”按钮软件将自动连接到系统找到的第一张 WD663X 模块；

指定连接：根据 WDSerial_DeviceFind_PCI（）函数查询得到的总线号和设备号连接指定的 WD663X 模块如图 4.3。总线号和设备号还可以通过计算机设备管理器查询。打开计算机设备管理器，在 WD663X 模块属性中有如下描述：“PCI Slot 1 (PCI 总线 3、设备 5、功能 0)”。此时，模块总线号为 3，设备号为 5。

退出：断开模块连接并退出软件。

连接状态栏：模块连接成功后，显示模块的相关信息，如连接状态、厂商 ID、板卡 ID 等。

如果模块连接成功，连接状态指示灯显示绿色。

如果模块连接失败，弹出提示框显示“连接板卡失败”字样，并且连接状态指示灯显示黑色。此时请检查硬件安装是否正确、驱动安装是否正确。检查完毕后，重启计算机运行该软件。

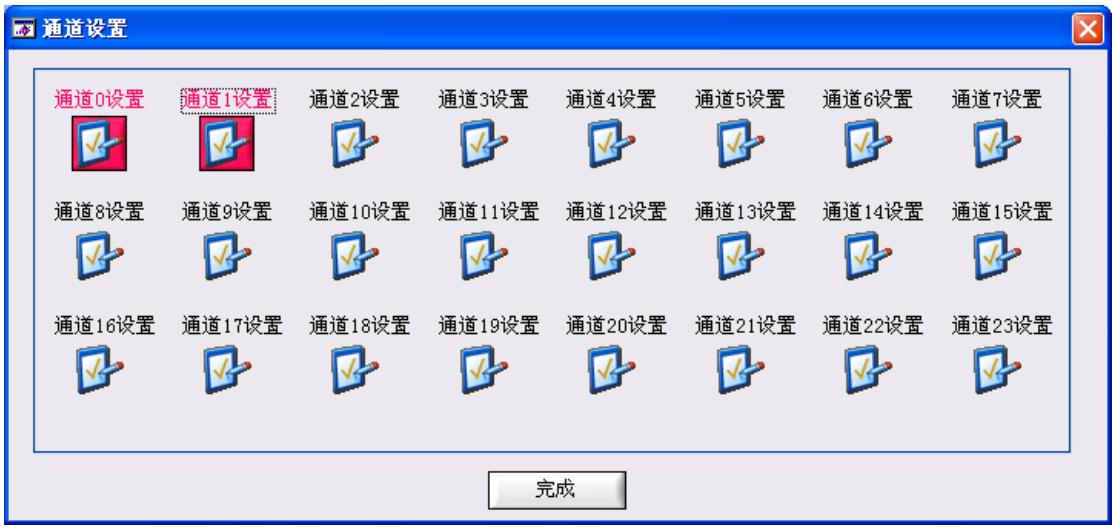
■ 系统设置：

设置串口发送的发送周期和接收方式（查询方式读取数据，中断方式读取数据）。点击“系统设置”按钮，弹出如下操作界面：

这里总共可以缓存设置 8 种发送周期。
中断接收数据方式需设置中断接收的数据个数。

■ 通道设置：

点击“通道设置”按钮，弹出选择通道界面，如下图：



WD663X 串口模块有 16 个串口通道，每个串口通道可设置不同的波特率，列如：

通道 0 设置波特率为 115200，
通道 1 可设置波特率为：9600，
通道 2 可设置波特率为：1200；

。。。。。。。

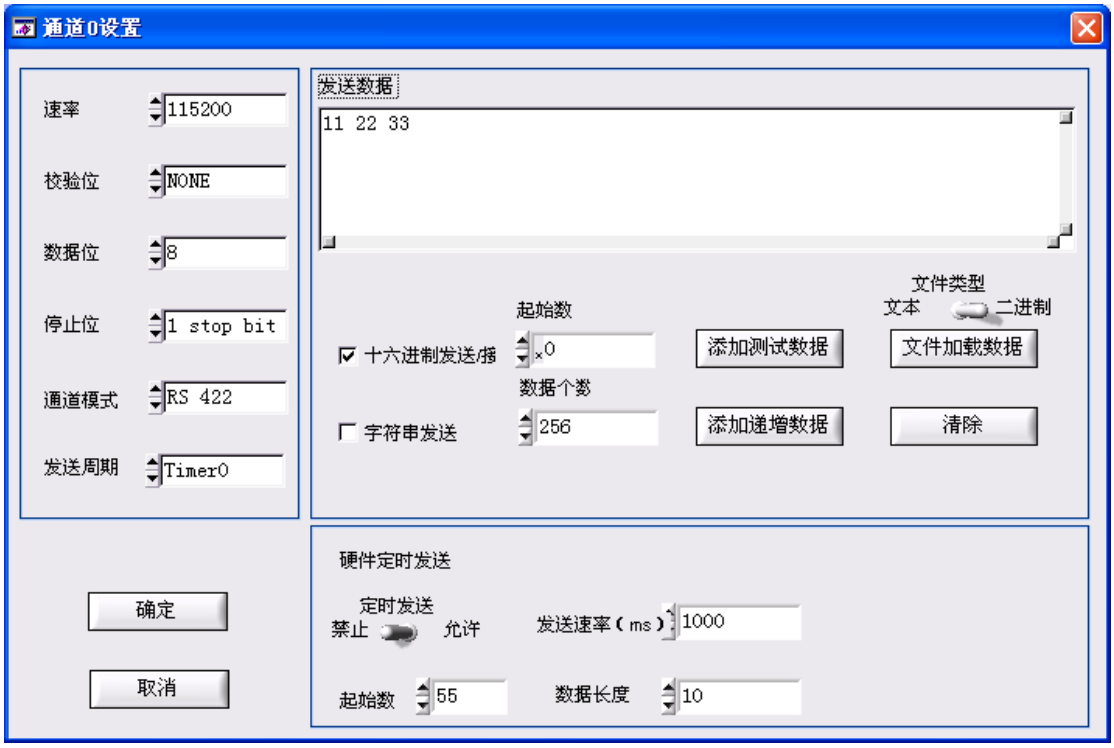
串口模式：WD663X 每个串口通道都支持 RS485/RS422/RS232 软件可选；
WD663X 支持最高 4Mbps，最低 75bps 波特率设置；
定时发送数据：WD663X 模块的 0~7 通道支持，8~16 不支持；

- 通道参数设置：

  点击相应的通道设置按钮，可进入对应的通道参数设置界面；

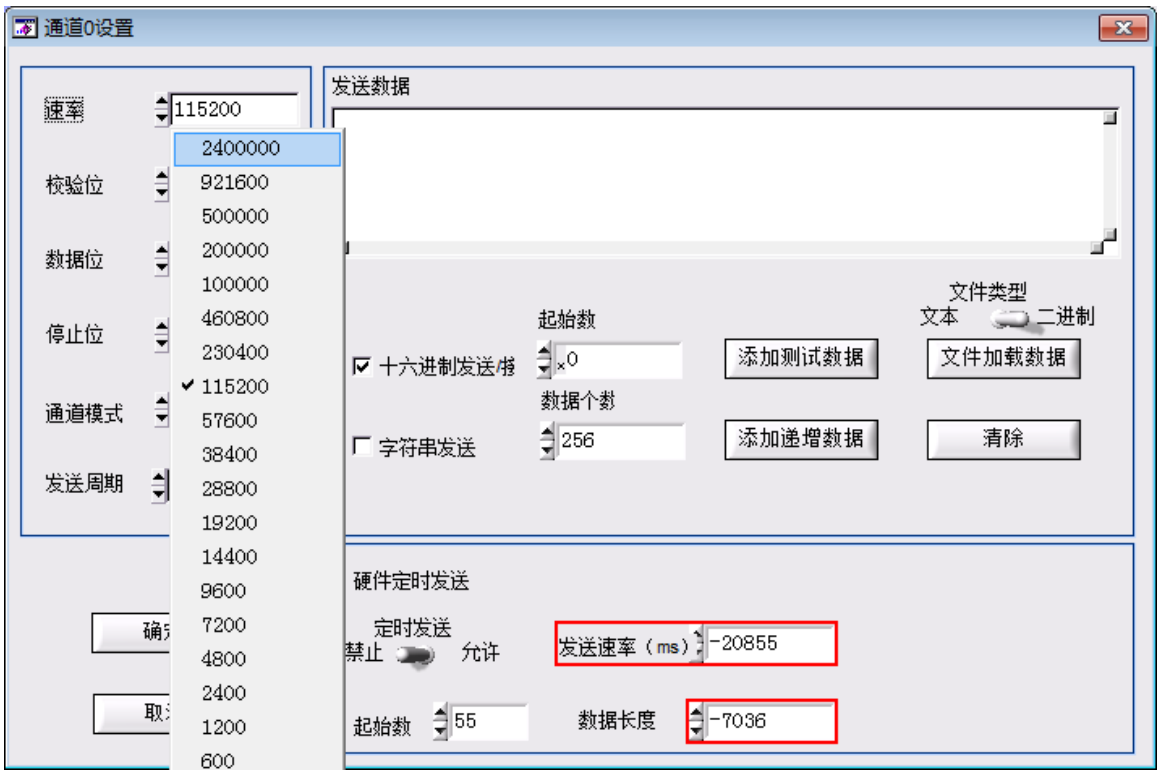
  示列：通道 0 参数设置；

  A、点击“通道 0 设置”按钮，弹出如下通道 0 参数设置界面：

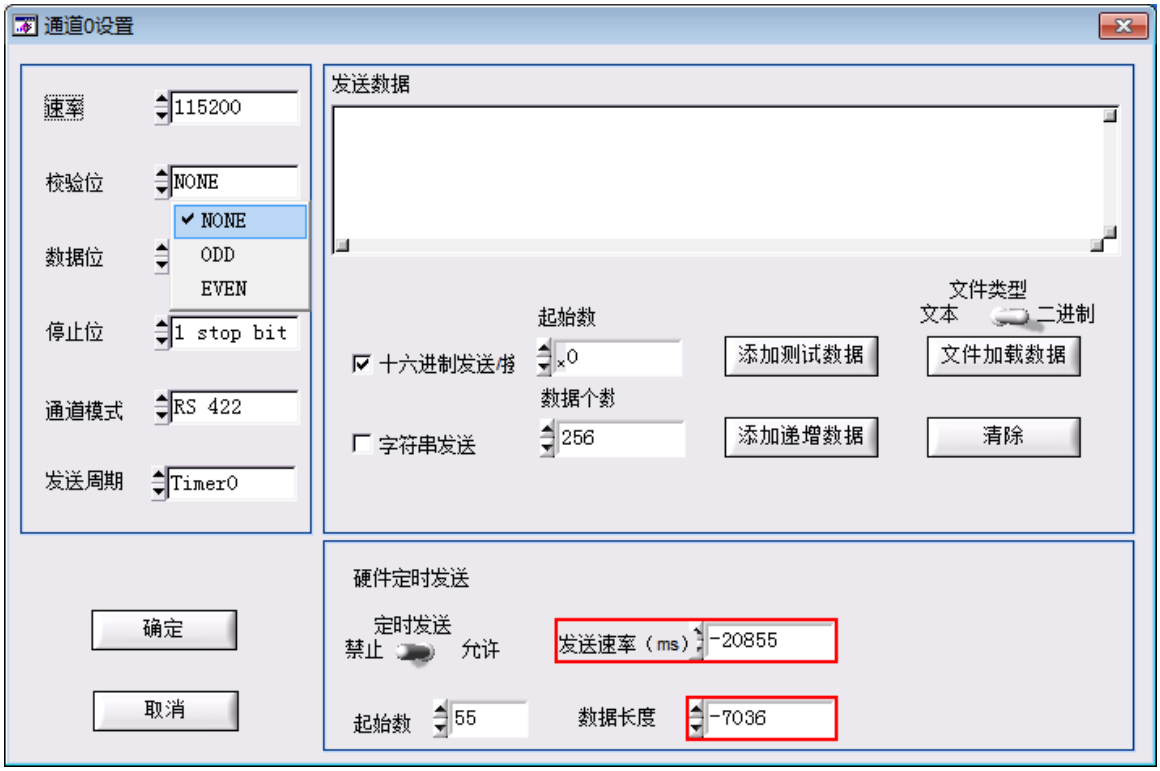


B、选择串口通道模式：WD663X 模块每个通道都支持 RS485/RS422/RS232 可选。

C、设置串口通信波特率：WD663X 支持最高 4Mbps，最低 75bps 波特率设置。这里通过波特率下拉列表框中选中 115200bps，如下图选择；



D、选择校验方式：WD663X 支持无校验、奇校验、偶校验三种发送校验方式。在校验位下拉列表框中选择“无校验”，如下图：



E、数据位：WD663X 支持数据位可设置，数据位可设置为 5、6、7 或 8 位。这里设置为 8 位。

F、停止位：WD663X 支持停止位可设置，停止位可设置为 1 或 2 位。这里设置为 1 位。

G、选择串口发送周期，这里选择在前面设置的 8 种发送周期中的 Timer0 为通道 0 的数据发送周期。

H、选择需要发送的数据的格式和个数，由两种格式可选 1) 十六进制发送 2）字符串发送；

I、在发送数据文本框中输入待发送数据。或者是选择定时发送数据(通道 0~7 有效)，

定时发送数据：

1）把“定时发送”拨码开关，拨码到允许端，

2）在“起始数”输入框中输入定时发送的数据的起始值，

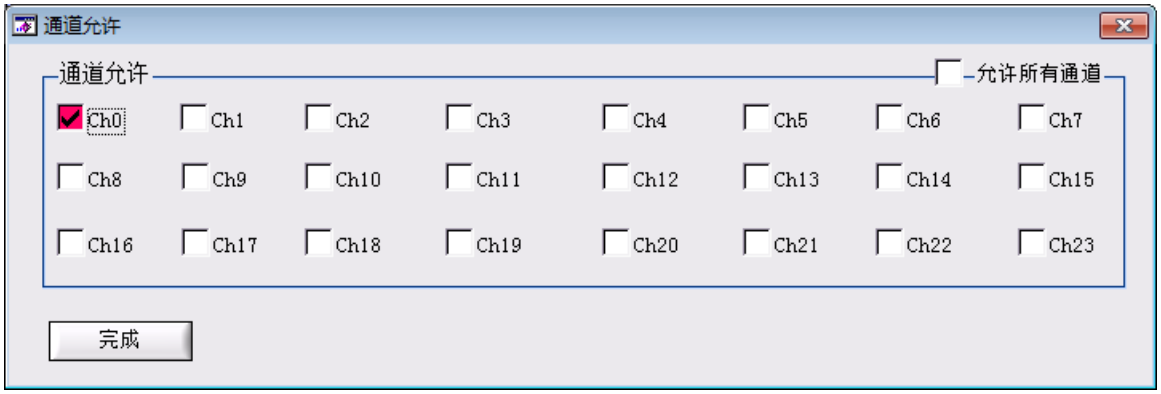
3）在发送速率框中输入定时发送的周期，

4）在数据长度框中输入定时发送的待发送数据个数，数据从起始数开始递增，增量为 1；

J、参数设置完成，点击“确定”按钮，向模块写入设置的参数。

K、回到通道设置界面，点击“完成”按钮，完成参数设置，回到主界面。

■ **打开通道设置：**点击“打开通道”按钮弹出如下界面：



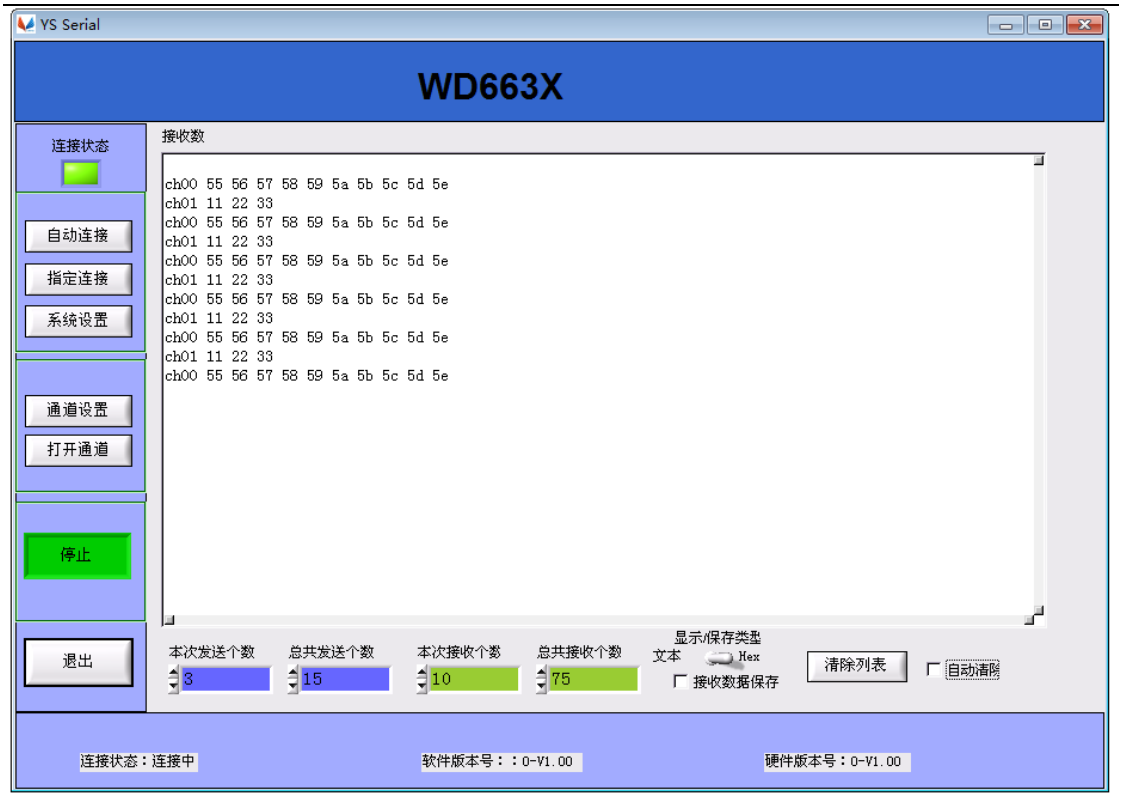
选中对应通道的复选框，即允许该通道输入/输出；

列：选择通道 0，即允许通道 0 输入/输出。未勾选的通道，禁止输入输出。

通道选择完成，点击“完成”按钮，完成允许/禁止通道输入/输出设置。

其他通道设置也按照上述顺序设置。

接好连接器针脚，点击启动按钮，串口通道能与其他串口进行通讯。如下图：



注：这里根据你输入的数据类型，选择把“显示/保存类型”拨码开关拨向对应数据类型端；文本：接收数据为字符串或文本，Hex：接收数据为 16 进制数据；

示列如下：WD663X 模块接收到其他串口设备发送的 16 进制数据，则把“显示/保存类型”拨码开关拨向 Hex 端。

WD663X 串口通讯设置步骤：

1、 设置串口接收数据方式，中断/查询；设置串口发送数据周期

2、 设置串口通道参数，如：波特率、校验、停止位、数据位、串口模式（RS 422/RS 485/RS232）、发送数据等

3、 打开对应串口通道，即允许/禁止对应串口输入/输出

4、 启动设备发送/接收

5. WD663X API Routines

《串口通讯模块软件手册(WD663X)》是基于最大 16 通道串口通讯模块 WD663X 函数库编写的，目的是为了让读者了解 API 函数库中的各类函数，便于用户在 API 函数库的基础上编程。其中各个函数的使用范例都是在 Labwindows/CVI2015 平台下编写的，且较为简略，详细使用可参考与板卡配套的示例程序。

WDSerial_AutoConnectFirst

函数名

函数功能

此函数自动连接到系统找到的第一张 WD663X 模块，并返回句柄以供其他函数使用。调用此函数同时对 WD663X 模块复位。调用此函数连接的模块与插在机箱中的位置无关。

连接函数只需要调用一次即可，在调用其他函数之前，需要使用此函数或 WDSerial_AutoConnectFirst 函数获得的模块句柄。

参数类型

函数原型

ViStatus _VI_FUNC WDSerial_AutoConnectFirst(ViUInt32* cardnum);

参数说明

参数输入/输出方向
in:输入 out:输出

cardnum

[out]：模块初始化句柄，成功获取后供其他函数调用。

返回值

VI_SUCCESS	(函数调用成功)
VI_ERROR_INV_OBJECT	（无效句柄）

使用例程

```
ViUInt32  cardnum;
hr = WDSerial _ AutoConnectFirst (&cardnum);
if((hr!=0)||(cardnum==0))  printf("模块连接失败");
```

6. 函数概要说明

本节对 API 函数功能进行概括性说明，使用户对 API 函数有一个总体的了解。

连接函数

WDSerial_DeviceFind_PCI	查找 PCI 总线类模块
WDSerial_AutoConnectFirst	自动连接第一张 WD663X 板卡，获得板卡句柄
WDSerial_Init	根据总线号和设备号连接板卡，获得板卡句柄
WDSerial_Reset	软件复位板卡
WDSerial_Close	关闭板卡，释放资源
WDSerial_GetRevision	获得板卡硬件版本号

串口功能函数

WDSerial_ReceiveBlockFIFO	读取指定通道接收到的数据
WDSerial_TransmitFIFOBlock	设置指定通道发送的数据
WDSerial_SetCh	设置通道参数（波特率等）
WDSerial_GetCh	获取通道参数（波特率等）
WDSerial_SetChMode	设置串口模式（RS232/RS422/RS485）
WDSerial_GetChMode	获得串口模式
WDSerial_StartStop	启动模块，开始数据收发
WDSerial_GetFiFoLen	获得指定通道接收 FIFO 中未读数据个数
WDSerial_ChannelEn	打开允许串口通道数据收发
WDSerial_StatusGetString	转换错误代码到错误信息
WDSerial_TxFIFOReset	清除通道发送 FIFO
WDSerial_RxFIFOReset	清除通道接收 FIFO
WDSerial_ConfigTxFrame	设置硬件定时发送状态
WDSerial_GetConfigTxFrame	读取硬件定时发送状态
WDSerial_UpdataFrameData	更新定时发送数据区

中断管理函数

WDSerial_RegisterFunction	注册中断回调函数
WDSerial_SetChInterupt	设置指定通道中断条件
WDSerial_EnableBoardInterrupt	允许板卡中断
WDSerial_DisableBoardInterrupt	禁止板卡中断

7. 函数详细说明

本节对每一个 API 函数进行详细说明，并列出简单使用范例。

WDSerial_AutoConnectFirst

函数功能

此函数自动连接找到的第一张模块，并返回句柄以供其他函数使用，只需要调用一次后其他函数就可以使用，不需要多次重复调用,除非调用了函数WDSerial_Close。如系统中有多张模块，与模块在系统的位置无关。

函数原型

ViStatus _VI_FUNC WDSerial_AutoConnectFirst(ViUInt32* cardnum);

参数说明

cardnum

[out] 模块初始化句柄，成功获取后供其他函数调用。

返回值

VI_SUCCESS (函数调用成功)

VI_ERROR_INV_OBJECT （无效句柄）

使用例程

```
ViUInt32 cardnum=0;
ViStatus hr;
hr = WDSerial_AutoConnectFirst(&cardnum);
if (hr == 0)
    printf("Device has been connected successfully");
```

WDSerial_DeviceFind_PCI

函数功能

此函数查找系统中的 PCI 总线类 WD663X 模块，并返回 PCI 模块厂家 ID、设备 ID、总线号和设备号。找到资源后可以调用函数 WDSerial_Init 初始化模块。

PCI 总线类模块包含 PCI、PCI-E、cPCI、PXI、cPCI-E、PXI-E 和 PC104 Plus 总线。

函数原型

```
ViStatus _VI_FUNC WDSerial_DeviceFind_PCI(
                                ViUInt16        MaxNum,
                                PCI_SOURCE*     PCI_SOURCE,
                                ViUInt16*       FindNum);
```

参数说明

MaxNum

[in] 查找模块的最大数。

PCI_SOURCE

[out] 查找到的 PCI 设备资源数组。PCI_SOURCE 结构体如下：

```
typedef struct pci_source
{
    ViUInt32 BusNumber; //总线号
    ViUInt32 SlotNumber;//槽号
    ViUInt32 VendorId;//厂商 ID
    ViUInt32 DeviceId;//模块 ID
}PCI_SOURCE;
```

FindNum

[out] 找到的模块数。

返回值

API_SUCCESS （函数调用成功）

VI_ERROR_RSRC_NFOUND (模块未找到)

使用列程

```
ViStatus hr;
ViUInt32 cardnum;
PCI_SOURCE    pPCI_SOURCE [10]；
ViUInt16 FindNum=0；
//查找 PCI 设备
hr = WDSerial_DeviceFind_PCI (10，pPCI_SOURCE，& FindNum);
if(hr!=0)    printf(“设备查找失败！”);
//连接设备
hr=WDSerial_Init(
                    pPCI_SOURCE [0].BusNumber,
                    pPCI_SOURCE [0].SlotNumber,
                    &cardnum);
```

WDSerial_Init

函数功能

此函数初始化指定 BusNumber 和 DeviceNumber 的模块，并返回句柄以供其他函数使用。适用于系统中存在单张或多张 WD663X 板卡。此函数只需要调用一次后其他函数就可以使用，不需要多次重复调用,除非调用了函数 WDSerial_Close。退出程序时请调用函数 WDSerial_Close 释放资源。

函数原型

```
ViStatus  _VI_FUNC  WDSerial_Init(
                                ViUInt16        BusNumber,
                                ViUInt16        SlotNumber,
                                ViUInt32        *cardnum);
```

参数说明

cardnum

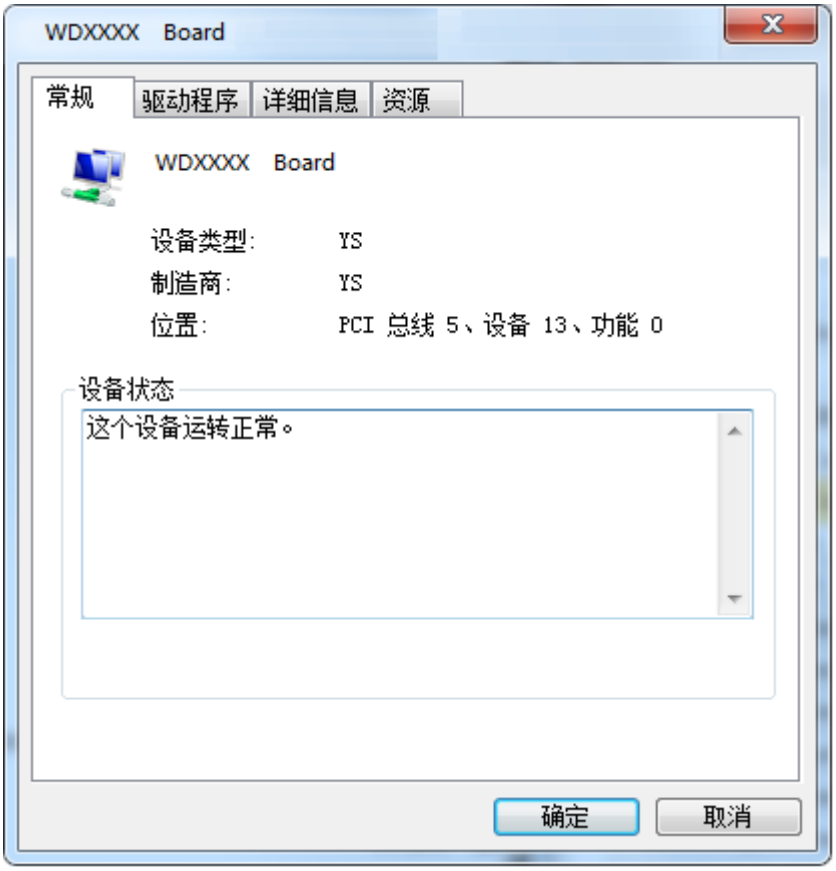
[out] 返回的模块初始化句柄，以供其它 API 函数使用。该函数须定义为全局变量。

BusNumber

[in] 模块总线地址，可在设备管理器中查询。

SlotNumber,

[in] 模块在系统中的设备号，可在设备管理器中查询。 总线号（BusNumber）和设备号（DeviceNumber）可通过计算机设备管理器查询。打开计算机设备管理器，在板卡属性中有如下描述：“PCI Slot 4 (PCI 总线 3、设备 4、功能 0)”，此时，总线号为 3，设备号为 4。如下图：



返回值

VI_SUCCESS (函数调用成功)

VI_ERROR_INV_OBJECT （无效句柄）

使用例程

```
ViStatus    hr;
ViUInt32 cardnum;
PCI_SOURCE    pPCI_SOURCE [10]；
ViUInt16 FindNum=0；
//查找 PCI 设备
hr = WDSerial_DeviceFind_PCI (10，pPCI_SOURCE，& FindNum);
if(hr!=0)    printf(“设备查找失败！”);
//连接设备
hr=WDSerial_Init(
                    pPCI_SOURCE [0].BusNumber,
                    pPCI_SOURCE [0].SlotNumber,
```

&cardnum);

WDSerial_Reset

函数功能

此函数软件复位模块。复位后发送 FIFO 的数据将被清空，模块发送和接收将被停止。各通道配置被设置为初始化状态。

函数原型

```
ViStatus _VI_FUNC    WDSerial_Reset(
                                    ViUInt32       cardnum);
```

参数说明

cardnum

[in] 模块句柄，由 WDSerial_Init 或 WDSerial_AutoConnectFirst 获得

返回值

VI_SUCCESS	(函数调用成功)
VI_ERROR_INV_OBJECT	（无效句柄）

使用例程

```
ViStatus hr=0;
ViUInt32       cardnum;
if (cardnum==0) return -1;
hr = WDSerial _Reset(cardnum);
if(hr != 0)
    MessagePopup(“err!”,”模块初始化失败！”);
```

WDSerial_Close

函数功能

本函数释放模块在初始化时所分配的程序资源，停止所有正在运行的线程。在应用程序退出之前必须调用此函数。

函数原型

ViStatus _VI_FUNC WDSerial_Close(

ViUInt32 cardnum);

参数说明

cardnum

[in] 模块句柄，由WDSerial_Init或WDSerial_AutoConnectFirst获得

返回值

VI_SUCCESS (函数调用成功)

VI_ERROR_INV_OBJECT （无效句柄）

使用例程

```
ViStatus    hr;
ViUInt32       cardnum;
 hr=WDSerial_Close(cardnum);
```

WDSerial_GetRevision

函数功能

返回 microcode 版本和 API 程序版本。

函数原型

```
ViStatus _VI_FUNC WDSerial_GetRevision(
                                    ViUInt32      cardnum,
                                    ViUInt32      *ucode_rev,
                                    ViUInt32      *api_rev);
```

参数说明

cardnum

[in] 模块句柄，由 WDSerial_Init 或 WDSerial_AutoConnectFirst 获得。

ucode_Rev

[out] 指向硬件的版本号。为 0x0100，表示 V1.00

api_Rev

[out] 指向 API 程序的版本号。为 0x0100，表示 V1.00

返回值

VI_SUCCESS	(函数调用成功)
VI_ERROR_INV_OBJECT	（无效句柄）

使用例程

```
ViStatus    hr;
ViUInt32       cardnum;
ViUInt32 api_rev,ucode_rev;
hr = WDSerial_GetRevision(cardnum, &ucode_rev, &api_rev);
```

WDSerial_StatusGetString

函数功能

从错误代码到错误信息的转换。调用模块其他函数后将返回函数状态代码，可以通过此函数将代码转换为信息描述。

函数原型

ViStatus _VI_FUNC WDSerial_StatusGetString(

char * message,

ViInt32 status);

参数说明

message

[out] 转换后的错误信息。

status

[in] 错误代码，为各函数的返回值。

错误代码	错误信息	错误代码	错误信息
0	函数返回成功	-12	校验失败
-1	其他错误	-13	存储空间不够
-2	无效板卡句柄号	-14	If no new data is available for this port
-3	无效通道号	-15	发送 buffer 忙
-4	无效长度值	-16	方式不匹配
-5	指针为空	-17	无效 VL 句柄
-6	没有找到板卡	-18	
-7	无效参数	-19	没有足够的空间创建 VL
-8	不匹配错误	-20	没有可用的端口
-9	发送超时	-21	连接失败
-10	接收超时	-22	不支持功能

-11	设备已关闭	-23	系统错

返回值

VI_SUCCESS (函数调用成功)

VI_ERROR_INV_OBJECT （无效句柄）

使用例程

```
ViStatus hr ;
char message [256];
//调用函数
hr = WDSerial _AutoConnectToFirst(&cardnum);
WDSerial _StatusGetString(hr, & message); //获取错误信息
MessagePopup(“错误信息”, pString); //显示错误信息
```

WDSerial_SetCh

函数功能

此函数用于对通道参数进行设置。包括速率、校验方式、数据字位数、停止位个数。最高速率为4Mbps，最低为75bps；校验方式为：奇效验（ODD），偶效验（EVEN），无效验（NONE）；数据位数：为8到5位；停止位：停止位为1-2位。

函数原型

```
ViStatus _VI_FUNC WDSerial_SetCh(
                                ViUInt32    cardnum,
                                ViUInt16    ch,
                                ViUInt16    rate,
                                ViUInt16    parity,
                                ViUInt16    bitcount,
                                ViUInt16    stopbit);
```

参数说明

cardnum

[in] 模块句柄，由WDSerial_Init或WDSerial_AutoConnectFirst获得。

ch

[in] 通道号。通道号0到16。

rate

[in] 通信速率设置：

设置值	波特率（bps）	设置值	波特率（bps）
0	500000	11	9600
1	200000	12	7200
2	100000	13	4800
3	460800	14	2400
4	230400	15	1200
5	115200	16	600

6	57600	17	300
7	38400	18	150
8	28800	19	75
9	19200	20	4000000
10	14400	21	921600
		23	614400

parity

[in] 设置接收或发送通道的校验方式，对应的校验方式如下：

0：NONE（无校验）；

2：ODD（奇校验）；

3：EVEN（偶校验）；

bitcount

[in] 数据位数，可设置值为：

0：8位；

1：7位

2：6位

3：5位

stopbit

[in] 停止位数，可设置值为：

1：1位停止位

2：2位停止位

返回值

VI_SUCCESS (函数调用成功)

VI_ERROR_INV_OBJECT （无效句柄）

使用范例

```
ViUInt16 ch=0;
```

//以上代码设置通道0波特率为115200，校验方式为无校验，BitCount为8位，停止位为1位。

```
hr = WDSerial_SetCh(cardnum, ch, 5, 0, 0, 1);
```

WDSerial_GetCh

函数功能

此函数得到指定通道的设置信息。包括速率、校验方式、数据字位数、停止位个数。最高速率为4Mbps，最低为75bps；校验方式为：奇效验（ODD），偶效验（EVEN），无效验（NONE）；数据位数：为8到5位；停止位: 停止位为1-2位。

函数原型

```
ViStatus _VI_FUNC WDSerial_SetCh(
                                ViUInt32    cardnum,
                                ViUInt16    ch,
                                ViUInt16    *rate,
                                ViUInt16    *parity,
                                ViUInt16    *bitcount,
                                ViUInt16    *stopbit)；
```

参数说明

cardnum

[in] 模块句柄，由WDSerial_Init或WDSerial_AutoConnectFirst获得。

ch

[in] 通道号。 通道号0到16。

rate

[out] 获得通信速率：

设置值	波特率（bps）	设置值	波特率（bps）
0	500000	11	9600
1	200000	12	7200
2	100000	13	4800
3	460800	14	2400
4	230400	15	1200
5	115200	16	600

6	57600	17	300
7	38400	18	150
8	28800	19	75
9	19200	20	4000000
10	14400	21	921600
		23	614400

parity

[out] 获得接收或发送通道的校验方式，对应的校验方式如下：

0：NONE（无校验）；

2：ODD（奇校验）；

3：EVEN（偶校验）；

bitcount

[out] 获得数据位数，值为：

0：8位；

1：7位

2：6位

3：5位

stopbit

[out] 停止位数，可设置值为：

1：1位停止位

2：2位停止位

返回值

VI_SUCCESS (函数调用成功)

VI_ERROR_INV_OBJECT （无效句柄）

使用例程

```
ViUInt16 ch=0, rate, parity, bitcount, stopbit;
hr = WDSerial_GetCh(cardnum, ch, &rate, &parity, &bitcount, &stopbit);
```

WDSerial_SetChMode

函数功能

此函数设置指定通道工作方式。通道模式为RS-422/RS-485-2W/RS-485-4W/RS-232。

函数原型

ViStatus _VI_FUNC WDSerial_SetChMode(

ViUInt32 cardnum,

ViUInt16 ch,

ViUInt32 type422);

参数说明

cardnum

[in] 模块句柄，由WDSerial_Init或WDSerial_AutoConnectFirst获得。

ch

[in] 通道号。通道号0到16。

type422

[In] 通道工作方式，码值对应关系如下：

0：RS-422模式；

1：RS-232模式；

2：RS-485模式（4 write）；

3：RS-485模式（2 write）

返回值

VI_SUCCESS (函数调用成功)

VI_ERROR_INV_OBJECT （无效句柄）

使用例程

```
ViUInt32 cardnum；
ViUInt16 ch=0；
ViUInt32 type422=1；
//设置通道0为RS-232模
```

hr = WDSerial_SetChMode(cardnum, 0, type422);式。

WDSerial_GetChMode

函数功能

此函数得到由函数WDSerial_SetChMode设定的模式。通道模式为RS-422/RS-485-2W/RS-485-4W/RS-232。只返回函数WDSerial_SetChMode设定的模式。

函数原型

```
ViStatus _VI_FUNC WDSerial_SetChMode(
                                ViUInt32    cardnum,
                                ViUInt16    ch,
                                ViUInt32    *type422);
```

参数说明

cardnum

[in] 模块句柄，由WDSerial_Init或WDSerial_AutoConnectFirst获得。

ch

[in] 通道号。通道号0到16。

type422

[out] 通道工作方式，码值对应关系如下：

0：RS-422模式；

1：RS-232模式；

2：RS-485模式（4 write）；

3：RS-485模式（2 write）。

返回值

VI_SUCCESS	(函数调用成功)
VI_ERROR_INV_OBJECT	（无效句柄）

使用例程

```
ViUInt32 type422;
hr = WDSerial_GetChMode(cardnum, 0, &type422);
```

WDSerial_StartStop

函数功能

开始/停止发送或接收，发送或接收在开始后才有效。在对模块设置完成后调用此函数开始接收和发送数据。

函数原型

```
ViStatus  _VI_FUNC  WDSerial_StartStop(
                                    ViUInt32     cardnum,
                                    ViUInt32     flag,
                                    ViUInt32     ExtTrg);
```

参数说明

cardnum

[in] 模块句柄，由WDSerial_Init或WDSerial_AutoConnectFirst获得。

flag

[in]0:停止，1:开始。

ExtTrg

[in] 保留，触发方式，仅启动时有效；0：内触发，1：外触发。

返回值

VI_SUCCESS	(函数调用成功)
VI_ERROR_INV_OBJECT	（无效句柄）

使用列程

```
ViStatus hr=0;
ViUInt32       cardnum;
int flag=1;
//启动串口发送、接收
hr = WDSerial_StartStop(cardnum,flag,0);
if(hr != 0)
    MessagePopup(“err!”,”模块初始化失败！”);
```

WDSerial_ChannelEn

函数功能

允许/禁止对应通道收发数据，通道允许后才能收发数据。

函数原型

```
ViStatus _VI_FUNC WDSerial_ChannelEn(
                                    ViUInt32    cardnum,
                                    ViUInt32    Ch,
                                    ViUInt32    Enable);
```

参数说明

cardnum

[in] 模块句柄，由WDSerial_Init或WDSerial_AutoConnectFirst获得。

ch

[in] 通道号。通道号0到16。

Enable

[in] 对应通道允许。1：允许，0：禁止。

返回值

VI_SUCCESS	(函数调用成功)
VI_ERROR_INV_OBJECT	（无效句柄）

使用范例

```
ViUInt32  Enable  =1;
hr  =  WDSerial_ChannelEn  (cardnum,  0,  Enable);
```

WDSerial_GetFiFoLen

函数功能

得到接收通道FIFO中的数据个数。每个通道接收和发送FIFO的大小为16Kbyte。

函数原型

```
ViStatus _VI_FUNC WDSerial_GetFiFoLen(
                                ViUInt32    cardnum,
                                ViUInt16    ch,
                                ViUInt32    *len);
```

参数说明

cardnum

[in] 模块句柄，由WDSerial_Init或WDSerial_AutoConnectFirst获得。

ch

[in] 通道号。通道号0到16。

Len

[out] 得到的FIFO中的数据个数。

返回值

VI_SUCCESS	(函数调用成功)
VI_ERROR_INV_OBJECT	（无效句柄）

使用范例

```
ViUInt32 len=0;
hr = WDSerial_GetFiFoLen(cardnum, 0, &len);
```

WDSerial_TransmitFIFOBlock

函数功能

此函数写入一组数据到发送FIFO。写入发送FIFO后数据将依次发送，直到发送完成。调用WDSerial_StartStop函数启动发送/接收。

函数原型

```
ViStatus _VI_FUNC WDSerial_TransmitFIFOBlock(
                                    ViUInt32    cardnum,
                                    ViUInt16    ch,
                                    ViUInt16    count,
                                    ViUInt8     *datablock);
```

参数说明

cardnum

[in] 模块句柄，由WDSerial_Init或WDSerial_AutoConnectFirst获得。

ch

[in] 通道号。通道号0到16。

count

[in] 发送的数据个数。

datablock

[in] 发送的数据块。

返回值

VI_SUCCESS	(函数调用成功)
VI_ERROR_INV_OBJECT	（无效句柄）

使用例程

```
ViUInt16    count=10;
ViUInt8     datablock[10];
for(i=0; i<count; i++)
    datablock[i] = 0x55+i; //初始化发送数据块
hr = WDSerial_TransmitFIFOBlock(cardnum, 0, count, datablock);
```

WDSerial_ReceiveBlockFIFO

函数功能

此函数从接收FIFO中读数据，microcode将收到的数据依次放入FIFO中，每个通道接收和发送FIFO的大小为16Kbyte，调用WDSerial_StartStop函数启动发送/接收。

函数原型

```
ViStatus _VI_FUNC WDSerial_ReceiveBlockFIFO(
                                    ViUInt32        cardnum,
                                    ViUInt16        ch,
                                    ViUInt16        maxcount,
                                    ViUInt8         *datablock,
                                    ViUInt32        *rxcount);
```

参数说明

cardnum

[in] 模块句柄，由WDSerial_Init或WDSerial_AutoConnectFirst获得。

ch

[in] 通道号。通道号0到16。

maxcount

[in] 可读的最大数据个数，最大为16383。

datablock

[out] 接收到的数据。数据指针,需要分配maxcount大小的数据缓冲区。

rxcount

[out] 读回的实际数据个数，此值小于等于maxcount。当接收FIFO中数据个数大于等于maxcount时，rxcount等于maxcount，当接收FIFO中的数据小于maxcount时，rxcount等于接收FIFO中的数据个数。

返回值

VI_SUCCESS (函数调用成功)

VI_ERROR_INV_OBJECT （无效句柄）

使用例程

```
ViUInt16      maxcount=50;
ViUInt8       datablock[50];
ViUInt32      rxcount=0;
ViStatus   hr;
ViUInt32      cardnum;
hr = WDSerial_ReceiveBlockFIFO(cardnum, 0,  maxcount,
                                    datablock, &rxcount);
if (rxcount==0) return;
for (i=0; i<rxcount; i++)
{
    sprintf(str  Buf, “data=%.2x”, datablock[i]);
     printf(strBuf); //打印的收到的数据
}
```

WDSerial_TxFIFOReset

函数功能

此函数清除通道发送FIFO中的数据。

函数原型

```
ViStatus  _VI_FUNC  WDSerial_TxFIFOReset(
                                    ViUInt32     cardnum  ,
                                    ViUInt16     ch);
```

参数说明

cardnum

[in] 模块句柄，由WDSerial_Init或WDSerial_AutoConnectFirst获得。

ch

[in] 通道号。通道号0到16。

返回值

VI_SUCCESS	(函数调用成功)
VI_ERROR_INV_OBJECT	（无效句柄）

使用例程

```
ViUInt16  ch=10;
//以上程序清除通道10的发送FIFO。
hr  =  WDSerial_TxFIFOReset  (cardnum,ch);
```

WDSerial_RxFIFOReset

函数功能

此函数清除通道接收FIFO中的数据。

函数原型

```
ViStatus  _VI_FUNC  WDSerial_RxFIFOReset(
                                    ViUInt32     cardnum  ,
                                    ViUInt16     ch  );
```

参数说明

cardnum

[in] 模块句柄，由WDSerial_Init或WDSerial_AutoConnectFirst获得。

ch

[in] 通道号。通道号0到16。

返回值

VI_SUCCESS	(函数调用成功)
VI_ERROR_INV_OBJECT	（无效句柄）

使用例程

```
ViUInt16  ch=10;
//以上程序清除通道10的接收FIFO
hr  =WDSerial_RxFIFOReset  (cardnum,ch);
```

WDSerial_ConfigTxFrame

函数功能

此函数设置硬件定时发送，模块通道0到7共8个通道支持硬件定时发送，其他通道不支持。

函数原型

```
ViStatus _VI_FUNC WDSerial_ConfigTxFrame(
                                ViUInt32    cardnum,
                                ViUInt16    Ch,
                                ViUInt16    EN,
                                ViUInt16    Frame_Len,
                                ViUInt16    Frame_Frq,
                                ViUInt16    BuffB);
```

参数说明

cardnum

[in] 模块句柄，由WDSerial_Init或WDSerial_AutoConnectFirst获得。

ch

[in] 通道号。通道号0到7。

EN

[in] 定时发送允许。

Frame_Len

[in] 定时发送数据长度，最大2047个数据。

Frame_Frq

[in] 定时发送数据周期，单位ms，范围1~65535。

BuffB

[in] 当前数据Buffer，定时发送至此后双Buffer数据存储。

0：BufferA；

1：BufferB。

返回值

VI_SUCCESS (函数调用成功)

VI_ERROR_INV_OBJECT （无效句柄）

使用例程

```
ViUInt16    Ch=0;
ViUInt16    EN=1;
ViUInt16    Frame_Len=10;
ViUInt16    Frame_Frq=100;
ViUInt16    BuffB=0;
hr=  WDSerial_ConfigTxFrame(
    cardnum,
    Ch,   // (i) base address of memory area
    EN,   // (i) base address of memory area
    Frame_Len,   // (i) base address of memory area
    Frame_Frq,
    BuffB);
```

WDSerial_GetConfigTxFrame

函数功能

此函数读取硬件定时发送状态，模块通道0到7共8个通道支持硬件定时发送，其他通道不支持。

函数原型

```
ViStatus _VI_FUNC WDSerial_GetConfigTxFrame(
                                ViUInt32    cardnum,
                                ViUInt16    Ch,
                                ViUInt16    *EN,
                                ViUInt16    *Frame_Len,
                                ViUInt16    *Frame_Frq,
                                ViUInt16    *BuffB);
```

参数说明

cardnum

[in] 模块句柄，由WDSerial_Init或WDSerial_AutoConnectFirst获得。

ch

[in] 通道号。通道号0到7。

EN

[out] 定时发送允许。

Frame_Len

[out] 定时发送数据长度，最大2047个数据。

Frame_Frq

[out] 定时发送数据周期，单位ms，范围1~65535。

BuffB

[out] 当前数据Buffer，定时发送至此后双Buffer数据存储。

0：BufferA；

1：BufferB。

返回值

VI_SUCCESS (函数调用成功)

VI_ERROR_INV_OBJECT （无效句柄）

使用例程

```
ViUInt16    Ch=0;
ViUInt16    EN=0;
ViUInt16    Frame_Len=0;
ViUInt16    Frame_Frq=0;
ViUInt16    BuffB=0;
hr= WDSerial_GetConfigTxFrame (
    cardnum,
    Ch,  // (i) base address of memory area
    &EN,  // (i) base address of memory area
    &Frame_Len,  // (i) base address of memory area
    &Frame_Frq,
    &BuffB);
```

WDSerial_UpdataFrameData

函数功能

此函数更新定时发送数据区，定时发送数据为双缓冲区，可以实现数据的乒乓发送。

函数原型

```
ViStatus _VI_FUNC WDSerial_UpdataFrameData(
                                ViUInt32    cardnum,
                                ViUInt16    Ch,
                                ViUInt16    BuffB,
                                ViUInt8     *pData,
                                ViUInt16    Len);
```

参数说明

cardnum

[in] 模块句柄，由WDSerial_Init或WDSerial_AutoConnectFirst获得。

ch

[in] 通道号。通道号0到7。

BuffB

[in] 更新的数据区。

pData

[in] 更新的数据。

Len

[in] 数据长度。

返回值

VI_SUCCESS	(函数调用成功)
VI_ERROR_INV_OBJECT	（无效句柄）

使用例程

```
ViUInt16    Ch=0;
ViUInt16    EN=0;
```

```
ViUInt16     BuffB=0;
ViUInt8      pData [10]={0x11, 0x22, 0x33, 0x44, 0x55, 0x66, 0x77, 0x88,
0x99, 0xAA};
ViUInt16     Len=10;
hr= WDSerial_UpdataFrameData (
    cardnum,
    Ch,   // (i) base address of memory area
    BuffB,   // (i) base address of memory area
    pData,   // (i) base address of memory area
    Len);
```

WDSerial_SetChInterupt

函数功能

此函数对指定通道进行中断设置。中断类型包含FIFO中接收数据个数中断和收到特定字符中断。

函数原型

```
ViStatus _VI_FUNC WDSerial_SetChInterupt(
                                ViUInt32    cardnum,
                                ViUInt16    ch,
                                ViUInt16    masklength_int,
                                ViUInt16    maskchar_int,
                                ViUInt16    intlength,
                                ViUInt8     intchar);
```

参数说明

cardnum

[in] 模块句柄，由WDSerial_Init或WDSerial_AutoConnectFirst获得。

ch

[in] 通道号。通道号0到16。

masklength_int

[in] 是否允许设置的数据长度产生中断,1：允许； 0：禁止。

maskchar_int

[in] 保留，是否允许收到特殊字符产生中断，1：允许 0：禁止。

intlength

[in] 产生中断的数据长度可设1~4096，当masklength_int为1时有效。

intchar

[in] 保留，产生中断的特殊字符，当maskchar_int为1时有效。

返回值

VI_SUCCESS	(函数调用成功)
VI_ERROR_INV_OBJECT	（无效句柄）

使用例程

ViUInt16　masklength_int=1，intlength=10;

//以上程序设置了指定通道收到的数据大于等于10时，产生中断。

hr = WDSerial_SetChInterupt(cardnum,ch，masklength_int,0,intlength,0);

WDSerial_RegisterFunction

函数功能

安装中断服务程序。在此模块中有两类中断，FIFO中接收数据个数中断和特定字符中断。在设置的相关中断产生时，中断服务程序被调用。

函数原型

```
ViStatus _VI_FUNC WDSerial_RegisterFunction(
                                    ViUInt32        cardnum,
                                    API_INT_FIFO_RS *sIntFIFO1,
                                    ViUInt32        flag);
```

参数说明

cardnum

[in] 模块句柄，由WDSerial_Init或WDSerial_AutoConnectFirst获得。

sIntFIFO1

[in]详见Data Structures。

flag

[in] 可以设置的值有，1：注册中断回调函数，0：注销中断回调函数

返回值

VI_SUCCESS (函数调用成功)

VI_ERROR_INV_OBJECT （无效句柄）

使用例程

```
static API_INT_FIFO_RS sIntFIFO1; //声明中断回调函数
ViInt32 _stdcall emo_Data_watch_function(ViUInt32 ardnum,struct API_INT
_FIFO_RS *sIntFIFO);
sIntFIFO1.function = demo_rt_watch_function;
sIntFIFO1.iPriority = THREAD_PRIORITY_ABOVE_NORM;
sIntFIFO1.dwMilliseconds = 0;
sIntFIFO1.iNotification = 0; // Dont care about startup or shutdown
sIntFIFO1.mask_index=0xFFFF;
```

```
WDSerial_SetChInterupt(cardnum,0,1,0,10,0x5a);
WDSerial_RegisterFunction(cardnum,  &sIntFIFO1,1);
```

WDSerial_EnableBoardInterrupt

函数功能

此函数允许模块中断产生，中断条件由WDSerial_SetChInterupt设置。

函数原型

ViStatus _VI_FUNC WDSerial_EnableBoardInterrupt(

ViUInt32 cardnum);

参数说明

cardnum

[in] 模块句柄，由WDSerial_Init或WDSerial_AutoConnectFirst获得。

返回值

VI_SUCCESS	(函数调用成功)
VI_ERROR_INV_OBJECT	（无效句柄）

使用例程

```
hr= WDSerial_EnableBoardInterrupt(cardnum);
```

WDSerial_DisableBoardInterrupt

函数功能

此函数禁止模块中断产生。

函数原型

ViStatus _VI_FUNC WDSerial_DisableBoardInterrupt(

ViUInt32 cardnum);

参数说明

cardnum

[in] 模块句柄，由WDSerial_Init或WDSerial_AutoConnectFirst获得。

返回值

VI_SUCCESS (函数调用成功)

VI_ERROR_INV_OBJECT （无效句柄）

使用例程

hr= WDSerial_ DisableBoardInterrupt(cardnum);

附录 Data Structures

这一节描述了在API中使用到的数据结构，描述了每个结构的元素和使用方法。使用到的数据结构有：

API_INT_FIFO_RS

```
typedef struct api_int_fifo_rs
{
    ViInt32 (_stdcall *function)(ViUInt32 cardnum, struct API_INT_FIFO_RS *pFIFO);
    int iPriority; // THREAD_PRIORITY_* as defined for Windows function
    // SetThreadPriority() or zero for normal priority.
    ViUInt32 dwMilliseconds;// Thread time-out interval in milliseconds or INFINITE
    // Mask to request startup and shutdown notification:
    ViInt32 iNotification; // CALL_STARTUP if function to be called at creation
    // of thread, CALL_SHUTDOWN if function is to be
    // called upon destruction of thread. “OR” both
    // together to enable notification on both events.
    // See “bForceShutdown” and “bForceStartup” below.
    // User variables; not referenced by the API:
    int nUser[8]; // Spare variables for use by the user.
    void *pUser[8]; // Spare variables for use by the user.
    // Event filter structure. A one “1' enables the specified event; when
    // detected the API will place it in the FIFO and call the user function.
    // Top Level Notification Event Mask:
    ViInt32 FilterType; // One or more EVENT_ definitions, “or'ed” together.
    // Event filter mask array.
    // rt tr sa
    ViUInt32 cardnum; // Card number associated with this thread. (RO) $
   ViInt32 head_index; // Index of element being added to queue (0->63) (RO) $
    ViInt32 tail_index; // Index of element to be removed from queue (RW) $
    ViInt32 mask_index; // Mask for wrapping head and tail pointers (RO) $
    int bForceShutdown;// 1 - Thread is being shutdown, -1 complete (RO) $
    int bForceStartup; // 1 - Thread is being started, 0 complete (RO) $
    int nPtrIndex; // Index into API pointer table (RO) $
    ViUInt32 numEvents; // Total number of events, including overflows (RW) $
```

```
    ViUInt32 queue_oflow; // Count incremented by API when FIFO overflows (RW) $
    void* hEvent; // Handle to event object (RO) $
    //HANDLE hkEvent; // Kernel mode handle to event object (RO) $
    void* hThread; // Handle to thread (RO) $
    ViUInt32 lThreadId; // ID of new user interrupt thread. (RO) $

    struct BT_FIFO // FIFO structure: events for user to process. $
    {
        ViUInt16 chnum;
        // unless mode code (then indicates mode code number)
        ViInt32 wordcount; // Word count of message; 0-31; 0 indicates 32 words
        ViInt32 reserved2; // Reserved for API.
    }
    fifo[MAX_FIFO_LEN]; // FIFO has exactly 64 entries.
}
API_INT_FIFO_RS;
```

ViInt32 (_stdcall *function)(ViUInt32 cardnum, struct API_INT_FIFO_RS *pFIFO);

注册的回调函数。

int iPriority; 创建的线程优先级，固定为

THREAD_PRIORITY_ABOVE_NORMAL。

ViUInt32 dwMilliseconds;// 线程等待事件超时错，如设置为INFINITE时为永不超时。

ViInt32 iNotification; // CALL_STARTUP if function to be called at creation

int nUser[8]; 用户数据存储

void *pUser[8]; 用户指针数据存储

ViInt32 FilterType; // 保留.

ViUInt32 cardnum; //板卡句柄

ViInt32 head_index; //中断信息FIFO中的头指针。

ViInt32 tail_index; // 中断信息FIFO中的尾指针。

ViInt32 mask_index; //中断屏蔽,bit0~bit15表示ch0~ch15中断允许或禁止。1：允许 0：禁止

fifo[MAX_FIFO_LEN]; // 中断信息FIFO，此FIFO中记录了16个通道中是否有未读数据的信息，上述的head_index和tail_index分别为此FIFO的头指针和尾指针。

该结构中chnum为接收通道号，wordcount为此通道接收到的数据个数。如下面的程序，表示了查找此FIFO中接收通道的数据信息：

```
tail = sIntFIFO->tail_index; //获得FIFO的尾指针
while ( tail != sIntFIFO->head_index ) //当尾指针不等于头指针，表示有通道接收到了数据
{
     chnum    = sIntFIFO->fifo[tail].chnum; //接收到数据的通道号
    wordcount = sIntFIFO->fifo[tail].wordcount; //接收到的数据个数
    //读取接受数据
    WDSerial_ReceiveBlockFIFO( cardnum, chnum, wordcount, datablock,
&rxcount);

    tail++;                              // Next entry
    tail&= sIntFIFO ->mask_index
    sIntFIFO->tail_index = tail;         // Save the index
}
```

附录 程序例程

板卡操作步骤：

连接板卡->获取板卡信息->设置读取数据方式->设置通道参数->获取通道参数->开始/停止模块功能—>关闭板卡。

```
/*************************************************************/
#include "WD663X.h"

ViUInt32 cardnum=0; //板卡句柄
static  API_INT_FIFO_RS  sIntFIFO1;
ViInt32  _stdcall  demo_Data_watch_function(ViUInt32  cardnum,struct
API_INT_FIFO_RS *sIntFIFO);

void main()
{
    ViUInt16 count=10; datablock[10];
    int ch=0;
     rc = WDSerial_AutoConnectFirst(&cardnum);//自动连接板卡
     //rc = WDSerial_Init(Busnum,deviceNum,&cardnum);//  多张板卡时

     /*中断管理函数，如果不需要使用中断方式读数据，则不用调用
WDSerial_SetChInterupt
    和 RegisterFunction 这两个函数*/
     /*rc  =  WDSerial_SetChInterupt(cardnum,
                    ch,//  通道
                    masklength_int,  //是否允许设置的数据长度产生中断
                    maskchar_int,  //保留，是否允许收到特殊字符产生中断
                    intlength,//产生中断的数据长度
                    intchar);//产生中断的特殊字符
    //注册、注销中断
    sIntFIFO1.function = demo_Data_watch_function;
    sIntFIFO1.iPriority = THREAD_PRIORITY_ABOVE_NORMAL;
    sIntFIFO1.dwMilliseconds = 0;
    sIntFIFO1.iNotification = 0; // Dont care about startup or shutdown
    sIntFIFO1.mask_index=0xFFFF;
    WDSerial_RegisterFunction(
                    cardnum,
                    &sIntFIFO1,  //API_INT_FIFO_RS  详见Data  Structures
                    1);//注册、注销标志*/
    WDSerial_EnableBoardInterrupt(cardnum);
    //设置通道参数
```

```
    rc = WDSerial_SetCh(cardnum,
                        ch,//通道
                        rate,//波特率
                        parity,//校验位
                        bitcount,//数据位
                        stopbit);//停止位

    //发送数据
    rc  =  WDSerial_TransmitFIFOBlock(cardnum,
                        ch,  //  通道
                        count,  //数据个数
                        datablock);//  数据
      WDSerial_StartStop(cardnum, 1,0);//开始、结束标志

      Monitor(); //查询方式读数据，如果采用中断方式，则不用调此函数

}
```

读数据方式

查询方式

```
Monitor()
{
     while
     {
          for (ch=0;ch<16;ch++)    //查找 16 个通道
          {
              WDSerial_ReceiveBlockFIFO(
                          cardnum,
                          ch,
                          maxcount,//最大读取个数,
                          datablock,//读回的数据
                          &rxcount);//读回数据的个数
            if (rxcount > 0)//  如果有数据，则解析数据
            {
                 for (i=0;i<rxcount;i++)
                 {
                     //打印数据
                       .....................................
                 }
            }
        }
       ProcessSystemEvents(); //CVI  库函数
      }
```

```
}
```

中断方式

```
//中断回调函数
ViInt32 _stdcall demo_Data_watch_function(ViUInt32 cardnum,struct API_INT_FIFO_RS *sIntFIFO)
{
    ViUInt16    chnum;
    ViInt32     tail=0, wordcount;
    ViUInt8     datablock[4096];
    ViUInt32    rxcount=0,i=0;

    tail = sIntFIFO->tail_index; //尾地址
    while ( tail != sIntFIFO->head_index )
    {
        chnum    = sIntFIFO->fifo[tail].chnum;
        wordcount = sIntFIFO->fifo[tail].wordcount;
        memset(datablock,0,sizeof(datablock));
        //读取接收数据
        WDSerial_ReceiveBlockFIFO(
                        cardnum,
                        ch,
                        maxcount,//最大读取个数,
                        datablock,//读回的数据
                        &rxcount);//读回数据的个数

        for (i=0;i<rxcount;i++)
        {
            .....//打印数据
        }
        tail++;                         // Next entry
        sIntFIFO->tail_index = tail;    // Save the index

    }
     return 0;
}
```

关闭板卡

```
//函数退出时必须调用此函数

void ProgramQuit()
{
    //如果注册了中断函数，此必须调用此函数注销中断
```

```
    /*
    WDSerial_DisableBoardInterrupt(cardnum);
    WDSerial_RegisterFunction(
                    cardnum,
                    &sIntFIFO1, //API_INT_FIFO_RS 详见Data Structures
                    0);//注册、注销标志*/

     WDSerial_Close(cardnum);
}
```