第 10 章　上位机程序开发

在 USB 设备开发中，上位机程序是用于与用户进行接口的。上位机程序通过 USB 设备驱动程序和外部的 USB 硬件进行通信，USB 固件程序执行所用的硬件操作。一般来说，根据选择开发平台的不同，可以使用 Visual C++、Visual C#和 LabVIEW 等开发上位机程序。

本章首先介绍了 Visual C++中控制 USB 设备的相关函数，接着介绍了 Visual C#中读写 USB 设备的主意函数，最后介绍了在 LabVIEW 中如何读写 USB 设备。本章内容包括：

- Visual C++读写 USB 设备；
- Visual C#读写 USB 设备；
- LabVIEW 读写 USB 设备。

10.1　Visual C++读写 USB 设备

在 USB 设备开发过程中，上位机程序可以采用广泛应用的 Visual C++来实现。对于 Cypress 公司的 EZ-USB 系列芯片，其提供了全面的 CY3684 开发包。在该开发包中，可以使用 CYIOCTL 控制函数类和 CyAPI 控制函数类来实现 Visual C++环境下对 USB 设备的读写。

10.1.1　CYIOCTL 控制函数类

CYIOCTL 控制函数类为 Cypress 公司的 EZ-USB FX2LP 系列 USB 接口芯片，提供了简单的控制接口。在使用 Cypress 公司提供的驱动程序基础上，只需在主机 Visual C++程序中加入头文件 cyioctl.h，然后便可以调用相应的控制函数。

为了能够使用这些函数，主机程序必须首先获得 USB 设备的控制句柄。可以通过以下的代码在程序中获得连接到主机的 USB 设备句柄。

```
CCyUSBDevice *USBDevice = new CCyUSBDevice();          //USB 设备
HANDLE hDevice = USBDevice->DeviceHandle();            //打开设备句柄
```

其中，hDevice 即为获得的 USB 设备句柄。在退出程序的时候，需要释放该 USB 设备句柄，使用如下的语句即可：

```
delete USBDevice;
```

在主程序获得 USB 设备的控制句柄后，便可以调用 CYIOCTL 控制函数类提供的接口控制函数，下面分别进行介绍。

1．中止 I/O 端点的请求接口 IOCTL_ADAPT_ABORT_PIPE

中止 I/O 端点的请求接口 IOCTL_ADAPT_ABORT_PIPE 用于中止 I/O 端点的请求，其使用示例代码如下：

```
DWORD dwBytes = 0;
UCHAR Address = 0x82;                                  //地址
DeviceIoControl(hDevice, IOCTL_ADAPT_ABORT_PIPE,       //DeviceIoControl 函数
                &Address, sizeof (UCHAR),
```

```
                    NULL,  0,
                    &dwBytes, NULL);
```

这里在 DeviceIoControl 函数中，参数 hDevice 表示当前 USB 设备的句柄，参数 IOCTL_ADAPT_ABORT_PIPE 表示使用该接口进行通信，参数 Address 为通信的端点号及传输方向。

2．断开 USB 设备接口 IOCTL_ADAPT_CYCLE_PORT

断开 USB 设备接口 IOCTL_ADAPT_CYCLE_PORT 用于将 EZ-USB 设备从 USB 总线上断开，并进行重连接。其使用代码示例如下：

```
DWORD dwBytes = 0;
DeviceIoControl(hDevice, IOCTL_ADAPT_CYCLE_PORT,      //DeviceIoControl 函数
                    NULL,  0,
                    NULL,  0,
                    &dwBytes, NULL);
```

其中，参数 hDevice 表示当前 USB 设备的句柄，参数 IOCTL_ADAPT_CYCLE_PORT 表示使用该接口进行通信。

3．获得设备地址接口 IOCTL_ADAPT_GET_ADDRESS

获得设备地址接口 IOCTL_ADAPT_GET_ADDRESS 用于重新获得 EZ-USB 设备的地址，其使用示例代码如下：

```
DWORD dwBytes = 0;
UCHAR DevAddr;
    DeviceIoControl(hDevice, IOCTL_ADAPT_GET_ADDRESS, //DeviceIoControl 函数
                    &DevAddr, sizeof (UCHAR),
                    &DevAddr, sizeof (UCHAR),
                    &dwBytes, NULL);
```

其中，参数 hDevice 表示当前 USB 设备的句柄，参数 IOCTL_ADAPT_GET_ADDRESS 表示使用该接口进行通信，DevAddr 为返回的 USB 设备地址。

4．获取替换接口 IOCTL_ADAPT_GET_ALT_INTERFACE_SETTING

获取替换接口 IOCTL_ADAPT_GET_ALT_INTERFACE_SETTING 用于获得当前 EZ-USB 设备的可替换接口设置，其使用示例代码如下：

```
DWORD dwBytes = 0;
UCHAR intfc = 0;
UCHAR alt;
DeviceIoControl(hDevice, IOCTL_ADAPT_GET_ALT_INTERFACE_SETTING, //DeviceIoControl 函数
                    &intfc, sizeof (alt),
                    &alt, sizeof (alt),
                    &dwBytes, NULL);
```

其中，参数 hDevice 表示当前 USB 设备的句柄，参数 IOCTL_ADAPT_GET_ALT_INTERFACE_SETTING 表示使用该接口进行通信。

5．获取字符串接口 IOCTL_ADAPT_GET_DEVICE_NAME

获取字符串接口 IOCTL_ADAPT_GET_DEVICE_NAME 用于获得连接的 EZ-USB 设备的产品描述字符串，其使用示例代码如下：

```
DWORD dwBytes = 0;
ULONG len = 256;
UCHAR *buf = new UCHAR[len];
DeviceIoControl(hDevice, IOCTL_ADAPT_GET_DEVICE_NAME, //DeviceIoControl 函数
```

```
                    buf, len,
                    buf, len,
                    &dwBytes, NULL);
    delete[] buf;
```

其中，参数 hDevice 表示当前 USB 设备的句柄，参数 buf 为返回的字符串，参数 IOCTL_ADAPT_GET_DEVICE_NAME 表示使用该接口进行通信。

6．获取电源接口 IOCTL_ADAPT_GET_DEVICE_POWER_STATE

获取电源接口 IOCTL_ADAPT_GET_DEVICE_POWER_STATE 用于获得 EZ-USB 设备的电源状态，其使用示例代码如下：

```
DWORD dwBytes = 0;
UCHAR pwrState;
DeviceIoControl(hDevice, IOCTL_ADAPT_GET_DEVICE_POWER_STATE, //DeviceIoControl 函数
                    &pwrState,   sizeof (pwrState),
                    &pwrState, sizeof (pwrState),
                    &dwBytes, NULL);
```

其中，参数 hDevice 表示当前 USB 设备的句柄，参数 pwrState 为返回的字符串，参数 IOCTL_ADAPT_GET_DEVICE_POWER_STATE 表示使用该接口进行通信。

7．获取版本接口 IOCTL_ADAPT_GET_DRIVER_VERSION

获取版本接口 IOCTL_ADAPT_GET_DRIVER_VERSION 用于获得 EZ-USB 驱动的版本，其使用示例代码如下：

```
DWORD dwBytes = 0;
ULONG ver;
DeviceIoControl(hDevice, IOCTL_ADAPT_GET_DRIVER_VERSION, //DeviceIoControl 函数
                    &ver, sizeof (ver),
                    &ver, sizeof (ver),
                    &dwBytes, NULL);
```

其中，参数 hDevice 表示当前 USB 设备的句柄，参数 ver 为返回的版本号，参数 IOCTL_ADAPT_GET_DRIVER_VERSION 表示使用该接口进行通信。

8．获取替换名称接口 IOCTL_ADAPT_GET_FRIENDLY_NAME

获取替换名称接口 IOCTL_ADAPT_GET_FRIENDLY_NAME 用于获得 EZ-USB 设备的替换名称，其使用示例代码如下：

```
DWORD dwBytes = 0;
PUCHAR FriendlyName = new UCHAR[256];
DeviceIoControl(hDevice, IOCTL_ADAPT_GET_FRIENDLY_NAME, //DeviceIoControl 函数
                    FriendlyName, 256,
                    FriendlyName, 256,
                    &dwBytes, NULL);
delete[] FriendlyName;
```

其中，参数 hDevice 表示当前 USB 设备的句柄，参数 FriendlyName 为返回的字符串，参数 IOCTL_ADAPT_GET_FRIENDLY_NAME 表示使用该接口进行通信。

9．获取端点数接口 IOCTL_ADAPT_GET_NUMBER_ENDPOINTS

获取端点数接口 IOCTL_ADAPT_GET_NUMBER_ENDPOINTS 用于获得 EZ-USB 的端点数，其使用的示例代码如下：

```
DWORD dwBytes = 0;
```

```
UCHAR endPts;
DeviceIoControl(hDevice, IOCTL_ADAPT_GET_NUMBER_ENDPOINTS, //DeviceIoControl 函数
                        NULL, 0,
                        &endPts, sizeof (endPts),
                        &dwBytes, NULL);
```

其中，参数 hDevice 表示当前 USB 设备的句柄，参数 endPts 为返回的端点个数，参数 IOCTL_ADAPT_GET_NUMBER_ENDPOINTS 表示使用该接口进行通信。

10．获取传输大小接口 IOCTL_ADAPT_GET_TRANSFER_SIZE

获取传输大小接口 IOCTL_ADAPT_GET_TRANSFER_SIZE 用于获得 EZ-USB 的传输大小，其使用的示例代码如下：

```
DWORD BytesXfered;
SET_TRANSFER_SIZE_INFO SetTransferInfo;
SetTransferInfo.EndpointAddress = Address;
DeviceIoControl(hDevice,   IOCTL_ADAPT_GET_TRANSFER_SIZE, //DeviceIoControl 函数
                        &SetTransferInfo,   sizeof (SET_TRANSFER_SIZE_INFO),
                        &SetTransferInfo,   sizeof (SET_TRANSFER_SIZE_INFO),
                        &BytesXfered,   NULL);
LONG transferSz = SetTransferInfo.TransferSize;
```

其中，参数 hDevice 表示当前 USB 设备的句柄，使用了 SET_TRANSFER_SIZE_INFO 结构，参数 IOCTL_ADAPT_GET_TRANSFER_SIZE 表示使用该接口进行通信。

11．获取 USBDI 接口 IOCTL_ADAPT_GET_USBDI_VERSION

获取 USBDI 接口 IOCTL_ADAPT_GET_USBDI_VERSION 用于获得 USBDI 的版本号，其使用的示例代码如下：

```
DWORD dwBytes = 0;
ULONG ver;
DeviceIoControl(hDevice, IOCTL_ADAPT_GET_USBDI_VERSION, //DeviceIoControl 函数
                        &ver, sizeof (ver),
                        &ver, sizeof (ver),
                        &dwBytes, NULL);
```

其中，参数 hDevice 表示当前 USB 设备的句柄，参数 ver 表示返回的 USBDI 版本号，参数 IOCTL_ADAPT_GET_USBDI_VERSION 表示使用该接口进行通信。

12．复位设备接口 IOCTL_ADAPT_RESET_PARENT_PORT

复位设备接口 IOCTL_ADAPT_RESET_PARENT_PORT 用于复位 EZ-USB 设备，并清除错误标记，其使用代码示例如下：

```
DWORD dwBytes;
DeviceIoControl(hDevice, IOCTL_ADAPT_RESET_PARENT_PORT, //DeviceIoControl 函数
                          NULL, 0,
                          NULL, 0,
                          &dwBytes, NULL);
```

其中，参数 hDevice 表示当前 USB 设备的句柄，参数 IOCTL_ADAPT_RESET_PARENT_PORT 表示使用该接口进行通信。

13．复位端点接口 IOCTL_ADAPT_RESET_PIPE

复位端点接口 IOCTL_ADAPT_RESET_PIPE 用于复位 EZ-USB 设备的端点，并清除错误标记，其使用代码示例如下：

```
DWORD dwBytes;
UCHAR Address = 0x82;
DeviceIoControl(hDevice, IOCTL_ADAPT_RESET_PIPE, //DeviceIoControl 函数
                        &Address, sizeof (Address)
                        NULL, 0,
                        &dwBytes, NULL);
```

其中，参数 hDevice 表示当前 USB 设备的句柄，参数 IOCTL_ADAPT_RESET_PIPE 表示使用该接口进行通信。

14．设置替换接口 IOCTL_ADAPT_SELECT_INTERFACE

设置替换接口 IOCTL_ADAPT_SELECT_INTERFACE 用于选择设置 EZ-USB 可替换接口的设置，其使用示例代码如下：

```
DWORD dwBytes = 0;
UCHAR alt = 2;
DeviceIoControl (hDevice, IOCTL_ADAPT_SELECT_INTERFACE, //DeviceIoControl 函数
                        &alt, sizeof (alt),
                        &alt, sizeof (alt),
                        &dwBytes, NULL);
```

其中，参数 hDevice 表示当前 USB 设备的句柄，参数 IOCTL_ADAPT_SELECT_INTERFACE 表示使用该接口进行通信。

15．发送控制请求接口 IOCTL_ADAPT_SEND_EP0_CONTROL_TRANSFER

发送控制请求接口 IOCTL_ADAPT_SEND_EP0_CONTROL_TRANSFER 通过 EZ-USB 默认的端点 0 向设备发送控制请求，其使用示例代码如下：

```
union   {
       struct    {
         UCHAR Recipient:5;
         UCHAR   Type:2;
         UCHAR Direction:1;
       } bmRequest;
       UCHAR bmReq;
    };

    bmRequest.Recipient = 0;                                    //设备
    bmRequest.Type          = 2;                                //供应商
    bmRequest.Direction = 1;                                    //输入命令(从设备到主机)
    int iXmitBufSize = sizeof(SINGLE_TRANSFER) + bufLen;    //长度
    UCHAR *pXmitBuf = new UCHAR[iXmitBufSize];              //申请内存
    ZeroMemory(pXmitBuf, iXmitBufSize);
    PSINGLE_TRANSFER pTransfer = (PSINGLE_TRANSFER)pXmitBuf;
    pTransfer->SetupPacket.bmRequest = bmReq;
    pTransfer->SetupPacket.bRequest = ReqCode;
    pTransfer->SetupPacket.wValue = Value;
    pTransfer->SetupPacket.wIndex = Index;
    pTransfer->SetupPacket.wLength = bufLen;
    pTransfer->SetupPacket.ulTimeOut = TimeOut / 1000;
    pTransfer->WaitForever = false;
    pTransfer->ucEndpointAddress = 0x00;                        //控制管道
    pTransfer->IsoPacketLength = 0;
```

```
    pTransfer->BufferOffset = sizeof (SINGLE_TRANSFER);
    pTransfer->BufferLength = bufLen;
    DWORD dwReturnBytes;
    DeviceIoControl (hDevice, IOCTL_ADAPT_SEND_EP0_CONTROL_TRANSFER,
                             pXmitBuf,   iXmitBufSize,
                             pXmitBuf,   iXmitBufSize,
                             &dwReturnBytes,   NULL);
    UCHAR *ptr = pXmitBuf + sizeof (SINGLE_TRANSFER);    //复制数据到 buf 中
    memcpy(buf, ptr, dwReturnBytes);
```

16．数据传输接口 IOCTL_ADAPT_SEND_NON_EP0_TRANSFER

数据传输接口 IOCTL_ADAPT_SEND_NON_EP0_TRANSFER 用于进行 EZ-USB 的块、中断、同步等数据传输，其使用示例代码如下：

```
PUCHAR CCyBulkEndPoint::BeginDataXfer(PCHAR buf, LONG bufLen, OVERLAPPED *ov)
{
    if (hDevice == INVALID_HANDLE_VALUE)
        return NULL;
    int iXmitBufSize = sizeof (SINGLE_TRANSFER) + bufLen;
    PUCHAR pXmitBuf = new UCHAR[iXmitBufSize];
    ZeroMemory(pXmitBuf, iXmitBufSize);
    PSINGLE_TRANSFER pTransfer = (PSINGLE_TRANSFER)pXmitBuf;
    pTransfer->WaitForever = false;
    pTransfer->ucEndpointAddress = Address;
    pTransfer->IsoPacketLength = 0;
    pTransfer->BufferOffset = sizeof (SINGLE_TRANSFER);
    pTransfer->BufferLength = bufLen;
    UCHAR *ptr = (PUCHAR) pTransfer + pTransfer->BufferOffset;   //复制 buf 中的内容到 pXmitBuf 中
    memcpy(ptr, buf, bufLen);
    DWORD dwReturnBytes;
    DeviceIoControl(hDevice, IOCTL_ADAPT_SEND_NON_EP0_TRANSFER,
                                pXmitBuf, iXmitBufSize,
                                pXmitBuf, iXmitBufSize,
                                &dwReturnBytes, ov);
    return   pXmitBuf;
}
```

17．设置电源接口 IOCTL_ADAPT_SET_DEVICE_POWER_STATE

设置电源接口 IOCTL_ADAPT_SET_DEVICE_POWER_STATE 用于设置 EZ-USB 设备的电源状态，其使用示例代码如下：

```
DWORD dwBytes = 0;
UCHAR pwrState = 0;
DeviceIoControl(hDevice, IOCTL_ADAPT_SET_DEVICE_POWER_STATE, //DeviceIoControl 函数
                         &pwrState, sizeof (pwrState),
                         &pwrState, sizeof (pwrState),
                         &dwBytes, NULL);
```

其中，参数 hDevice 表示当前 USB 设备的句柄，参数 pwrState 表示电源状态码，参数 IOCTL_ADAPT_SET_DEVICE_POWER_STATE 表示使用该接口进行通信。

18．设置传输字节接口 IOCTL_ADAPT_SET_TRANSFER_SIZE

设置传输字节接口 IOCTL_ADAPT_SET_TRANSFER_SIZE 用于设置 EZ-USB 的传输字节，其使用

示例代码如下：

```
DWORD BytesXfered;
SET_TRANSFER_SIZE_INFO    SetTransferInfo;
SetTransferInfo.EndpointAddress    = Address;                    //地址
SetTransferInfo.TransferSize = 0x2000;                           //8 KB 传输字节
DeviceIoControl(hDevice,    IOCTL_ADAPT_SET_TRANSFER_SIZE, //DeviceIoControl 函数
                          &SetTransferInfo,    sizeof (SET_TRANSFER_SIZE_INFO),
                          &SetTransferInfo,    sizeof (SET_TRANSFER_SIZE_INFO),
                          &BytesXfered,    NULL);
```

10.1.2　CyAPI 控制函数类

Cypress 的 cyioctl 接口控制函数只是提供了基本的 USB 控制及传输操作，一般用于测试 USB 设备的连接。如果需要进行更加详细的控制操作，则需用使用 CyAPI 控制函数类。

CyAPI 控制函数类提供了更为精细的控制接口来读写 EZ-USB FX2LP 系列 USB 接口芯片。在使用 Cypress 公司提供的驱动程序基础上，只需在 Visual C++主机程序的加入头文件 CyAPI.h 和库文件 CyAPI.lib 即可调用相应的控制函数。

CyAPI 控制函数类主要包括 8 个控制类，包括块传输端点控制类 CCyBulkEndPoint、控制传输端点类 CCyControlEndPoint、中断传输端点控制类 CCyInterruptEndPoint、同步传输端点控制类 CCyIsocEndPoint、设备控制类 CCyUSBDevice、配置信息类 CCyUSBConfig、端点控制类 CCyUSBEndPoint 和接口控制类 CCyUSBInterface。下面分别进行介绍。

1．块传输端点控制类 CCyBulkEndPoint

块传输端点控制类 CCyBulkEndPoint 主要用于块传输端点。在该类中最常用的函数是 BeginDataXfer，其函数原型为：

```
PUCHAR CCyBulkEndPoint::BeginDataXfer(PCHAR buf, LONG len, OVERLAPPED *ov)
```

BeginDataXfer 函数是一个非常好用的异步 IO 传输方法。这个函数建立一个数据传输通道，初始化传输，并且立刻返回，不用等待传输完毕。其使用代码示例如下：

```
CCyUSBDevice    *USBDevice = new    CCyUSBDevice(Handle);

    OVERLAPPED outOvLap, inOvLap;
    outOvLap.hEvent= CreateEvent(NULL, false, false, "CYUSB_OUT");
    inOvLap.hEvent= CreateEvent(NULL, false, false, "CYUSB_IN");

    char    inBuf[128];
    ZeroMemory(inBuf,    128);

    char buffer[128];
    LONG    length = 128;

    //初始化数据输入输出
    UCHAR    *inContext = USBDevice->BulkInEndPt->BeginDataXfer(inBuf, length, &inOvLap);
    UCHAR    *outContext = USBDevice->BulkOutEndPt->BeginDataXfer(buffer, length, &outOvLap);
    //等待传输介绍
    USBDevice->BulkOutEndPt->WaitForXfer(&outOvLap,100);
    USBDevice->BulkInEndPt->WaitForXfer(&inOvLap,100);
     //结束传输
```

```
USBDevice->BulkOutEndPt->FinishDataXfer(buffer, length, &outOvLap,outContext);
USBDevice->BulkInEndPt->FinishDataXfer(inBuf, length, &inOvLap,inContext);
 //关闭输入输出
CloseHandle(outOvLap.hEvent);
CloseHandle(inOvLap.hEvent);
```

2．控制传输端点类 CCyControlEndPoint

在 CCyControlEndPoint 类中，同样定义了 BeginDataXfer 函数，用于异步 IO 操作。这里的函数原型如下：

```
PUCHAR CCyControlEndPoint::BeginDataXfer (PCHAR buf, LONG len, OVERLAPPED *ov)
```

该函数同样建立一个数据传输通道，初始化传输，并且立刻返回，不用等待传输完毕。在这里主要用于控制端点的数据传输。例如：

```
CCyUSBDevice    *USBDevice = new    CCyUSBDevice(Handle);

    //    Just for typing efficiency
    CCyControlEndPoint    *ept = USBDevice->ControlEndPt;

    OVERLAPPED OvLap;
    OvLap.hEvent    = CreateEvent(NULL, false, false, "CYUSB_CTL");

    char buffer[128];
    LONG    length     = 128;                                          //长度
    ept->Target          = TGT_DEVICE;
    ept->ReqType     = REQ_VENDOR;
    ept->Direction     = DIR_TO_DEVICE;
    ept->ReqCode     = 0x05;                                           //请求码
    ept->Value         = 1;
    ept->Index         = 0;

    PUCHAR Context = ept->BeginDataXfer(buffer, length, &OvLap);
    ept->WaitForXfer(&OvLap,100);
    ept->FinishDataXfer(buffer, length, &OvLap,Context);

    CloseHandle(OvLap.hEvent);
```

在程序中，首先声明控制端点 ept，并对 ept 进行了参数设置。共有 5 个参数，规定的端点 0 传输的方向，数据字节数等。下面分别介绍：

- Target：控制传输中的一个重要的参数，其可取值为 TGT_DEVICE、TGT_INTFC、TGT_ENDPT 和 TGT_OTHER。
- ReqType：请求的类型，其可取值为 REQ_STD、REQ_CLASS 和 REQ_VENDOR。
- Direction：控制控制端点数据传输的方向，其可取值为 DIR_TO_DEVICE 和 DIR_FROM_DEVICE 两个。
- ReqCode：请求的代码，在 USB 芯片的固件程序中，对该代码有专门的处理程序。控制端点的通信，主要便是通过其来实现的。
- Value：控制传输的重要参数，其值依赖于 ReqCode 参数。
- Index：控制传输的重要参数，其值依赖于 ReqCode 参数。

本例中，对 ept 初始化表示数据从 USB 设备发送到主机，数据长度为 128 个字节。通过 BeginDataXfer 函数将该命令发送 USB 设备。USB 芯片得到这个请求后，在相应的请求代码程序处理中将 128 个数据

发送给主机，主机将接收到的数据保存在 buffer 中。

除此以外，该类中还定义了 Write 和 Read 函数，其函数原型为：

```
bool CCyControlEndPoint::Write(PCHAR buf, LONG &len)
bool CCyControlEndPoint::Read( PCHAR buf, LONG &len)
```

Write 函数是一个简化版本的传输函数，将函数的传输方向固定为 DIR_TO_DEVICE，即主机向 USB 设备发送数据。Write 函数的使用代码示例如下：

```
CCyUSBDevice    *USBDevice = new CCyUSBDevice(Handle);

//   Just for typing efficiency
CCyControlEndPoint    *ept = USBDevice->ControlEndPt;          //控制端点

ept->Target     = TGT_DEVICE;
ept->ReqType    = REQ_VENDOR;
ept->ReqCode    = 0x07;                                        //请求码
ept->Value      = 1;
ept->Index      = 0;

char    buf[512];
ZeroMemory(buf,512);
LONG bytesToSend = 128;

ept->Write(buf,    bytesToSend);                               //Write 函数
```

Read 函数是一个简化版本的传输函数，将函数的传输方向固定为 DIR_FROM_DEVICE，即主机向 USB 设备发送数据。Read 函数的使用代码示例如下：

```
CCyUSBDevice    *USBDevice = new    CCyUSBDevice(Handle);

//   Just for typing efficiency
CCyControlEndPoint    *ept = USBDevice->ControlEndPt;          //控制端点

ept->Target    = TGT_DEVICE;
ept->ReqType = REQ_VENDOR;
ept->ReqCode = 0x07;
ept->Value     = 1;
ept->Index     = 0;

char    buf[512];
LONG bytesToRead = 64;

ept->Read(buf, bytesToRead);                                   //Read 函数
```

3．中断传输端点控制类 CCyInterruptEndPoint

中断传输端点控制类 CCyInterruptEndPoint 为 CCyUSBEndPoint 类的一个子类。中断传输端点控制类 CCyInterruptEndPoint 对中断端点的定义和使用代码，示例如下：

```
CCyInterruptEndPoint *IntInEpt = NULL;
  CCyInterruptEndPoint    *IntOutEpt = NULL;
  CCyUSBDevice *USBDevice = new CCyUSBDevice(Handle);

  int eptCount = USBDevice->EndPointCount();
  for (int i=1; i<eptCount; i++) {                             //端点信息
```

```
        bool    bIn    = USBDevice->EndPoints[i]->Address    & 0x80;
        bool    bInt = (USBDevice->EndPoints[i]->Attributes    == 3);

        if (bInt    && bIn) IntInEpt = (CCyInterruptEndPoint *) USBDevice->EndPoints[i];
        if (bInt && !bIn) IntOutEpt = (CCyInterruptEndPoint *) USBDevice->EndPoints[i];
    }
```

在 CCyInterruptEndPoint 类中，同样定义了 BeginDataXfer 函数，其用法和前面的类似，这里不再具体介绍。BeginDataXfer 函数原型为：

```
PUCHAR CCyInterruptEndPoint::BeginDataXfer(PCHAR buf, LONG len, OVERLAPPED *ov)
```

4．同步传输端点控制类 CCyIsocEndPoint

同步传输端点控制类 CCyIsocEndPoint 同样为 CCyUSBEndPoint 类的一个子类。同步传输端点控制类 CCyIsocEndPoint 对同步端点的定义和使用代码，示例如下：

```
    CCyIsocEndPoint *IsocInEpt = NULL;
    CCyIsocEndPoint    *IsocOutEpt = NULL;

    CCyUSBDevice *USBDevice    = new    CCyUSBDevice(Handle);
    int    eptCount = USBDevice->EndPointCount();

    for    (int    i=1; i<eptCount;    i++) {                                    //端点信息
        bool    bIn = USBDevice->EndPoints[i]->Address    & 0x80;
        bool    bInt = (USBDevice->EndPoints[i]->Attributes    == 1);
        if (bInt    && bIn) IsocInEpt = (CCyIsocEndPoint *) USBDevice->EndPoints[i];
        if    (bInt && !bIn) IsocOutEpt = (CCyIsocEndPoint *) USBDevice->EndPoints[i];
    }
```

在 CCyIsocEndPoint 类中定义了两个主要的函数：BeginDataXfer 和 CreatePktInfos。下面分别介绍。

❑ BeginDataXfer 函数

BeginDataXfer 函数同样用于执行异步 IO 操作，其函数原型为：

```
PUCHAR CCyIsocEndPoint::BeginDataXfer (PCHAR buf, LONG len, OVERLAPPED *ov)
```

BeginDataXfer 函数使用示例代码如下：

```
    CCyUSBDevice *USBDevice = new CCyUSBDevice(hWnd);
    CCyIsocEndPoint *IsoIn = USBDevice->IsocInEndPt;

if (IsoIn)
{

        int    pkts = 16;
        LONG bufSize = IsoIn->MaxPktSize * pkts;

        PUCHAR              context;
        OVERLAPPED        inOvLap;
        PUCHAR              buffer          = new UCHAR[bufSize];
        CCyIsoPktInfo *isoPktInfos = new CCyIsoPktInfo[pkts];

        IsoIn->SetXferSize(bufSize);

        inOvLap.hEvent = CreateEvent(NULL, false, false, NULL);

        // 开始数据传输
```

```
    context = IsoIn->BeginDataXfer(buffer, bufSize, &inOvLap);

    // 等待传输完毕
    if (!IsoIn->WaitForXfer(&inOvLap, 1500))
   {
       IsoIn->Abort();
       // Wait for the stalled command to complete
       WaitForSingleObject(inOvLap.hEvent,INFINITE);
    }

    int complete = 0;
    int partial = 0;

    // 必须调用 FinishDataXfer 函数清除内存
    if (IsoIn->FinishDataXfer(buffer, bufSize, &inOvLap, context, isoPktInfos))
   {
       for (int i=0; i< pkts; i++)
          if (isoPktInfos[i].Status)
             partial++;
          else
             complete++;
    }
      else
        partial++;

    delete buffer;                                          //删除 buffer
    delete [] isoPktInfos;
}
```

❑ CreatePktInfos 函数

CreatePktInfos 函数用于创建 CCyIsoPktInfo 实体，用于同步数据传输中，其函数原型为：

```
CCyIsoPktInfo* CCyIsocEndPoint::CreatePktInfos(LONG bufLen, int &packets)
```

CreatePktInfos 函数的使用示例代码如下：

```
  CCyUSBDevice *USBDevice = new CCyUSBDevice();
  CCyIsocEndPoint *IsoIn = USBDevice->IsocInEndPt;

  if (IsoIn) {

        LONG     bufSize = 4096;
    PUCHAR buffer = new UCHAR[bufSize];

    CCyIsoPktInfo *isoPktInfos;
    int                pkts;

    // 申请 IsoPktInfo 实体，并指出多少个申请到
        isoPktInfos = IsoIn->CreatePktInfos(bufSize, pkts);

        if (IsoIn->XferData(buffer, bufSize, isoPktInfos)) {

      LONG recvdBytes = 0;
```

```
        for (int i=0; i<pkts; i++)
            if (isoPktInfos[i].Status == 0)
                recvdBytes += isoPktInfos[i].Length;

    }

     delete [] buffer;                                          //删除 buffer
    delete [] isoPktInfos;
}
```

5．设备控制类 CCyUSBDevice

设备控制类 CCyUSBDevice 是一个原始的库入口类指针，其中定义了很多 USB 设备的各种操作，使用前必须首先获得 USB 设备的句柄。CCyUSBDevice 类中的成员函数，如表 10.1 所示。

表 10.1　CCyUSBDevice类成员函数

函数名	声明原形	功能
AltIntfc函数	UCHAR CCyUSBDevice::AltIntfc(void)	返回当前设备的可替换接口设置。
AltIntfcCount函数	UCHAR CCyUSBDevice::AltIntfcCount(void)	返回设备的可替换接口数。
BcdDevice	USHORT CCyUSBDevice::BcdUSB	USB设备描述符中的BcdUSB值。
BcdUSB	USHORT CCyUSBDevice::BcdUSB	USB设备描述符中的BcdUSB值。
BulkInEndPt	CCyBulkEndPoint* CCyUSBDevice::BulkInEndPt	指向第一个块输入端点数据的指针。
BulkOutEndPt	CCyBulkEndPoint* CCyUSBDevice::BulkOutEndPt	指向第一个块输出端点数据的指针。
Close函数	void CCyUSBDevice:: Close(void)	关断主机程序和USB驱动的接口。
Config函数	UCHAR CCyUSBDevice::Config(void)	返回USB设备的当前设置值。
ConfigAttrib	UCHAR CCyUSBDevice::ConfigAttrib	USB设备当前配置描述符的ConfigAttrib值。
ConfigCount函数	UCHAR CCyUSBDevice::ConfigCount(void)	返回USB设备的配置数。
ConfigValue	UCHAR CCyUSBDevice::ConfigValue	USB设备当前配置描述符的bConfigurationValue值。
ControlEndPt	CCyControlEndPoint* CCyUSBDevice::ControlEndPt	指向控制端点0。
DevClass	UCHAR CCyUSBDevice::DevClass	USB描述符的bDeviceClass值。
DeviceCount函数	UCHAR CCyUSBDevice::DeviceCount(void)	返回连接到USB总线上的USB设备个数。
DeviceHandle函数	HANDLE CCyUSBDevice::DeviceHandle(void)	返回USB设备的句柄。
DeviceName	char CCyUSBDevice::DeviceName[USB_STRING_MAXLEN]	字符数组，表示设备描述符的iProduct域。
DevProtocol	UCHAR CCyUSBDevice::DevProtocol	USB设备描述符bDeviceProtocol域的值。
DevSubClass	UCHAR CCyUSBDevice::DevSubClass	USB设备描述符bDeviceSubClass域的值。
DriverGUID函数	GUID CCyUSBDevice::DriverGUID(void)	返回USB驱动的GUID值。
DriverVersion	ULONG CCyUSBDevice::DriverVersion	驱动的版本。
EndPointCount函数	UCHAR CCyUSBDevice::EndPointCount(void)	返回当前接口的端点数加1，即包括端点0。
EndPointOf函数	CCyUSBEndPoint* CCyUSBDevice::EndPointOf(UCHAR addr)	返回端点的指针。
EndPoints	CCyUSBEndPoint* CCyUSBDevice::EndPoints	端点的列表。
FriendlyName	char CCyUSBDevice::FriendlyName[USB_STRING_MAXLEN]	驱动文件中的FriendlyName字段值。

GetDeviceDescriptor函数	void CCyUSBDevice::GetDeviceDescriptor(PUSB_DEVICE_DESCRIPTOR descr)	获得当前设备的设备描述符。
GetConfigDescriptor函数	void CCyUSBDevice::GetConfigDescriptor(PUSB_CONFIGURATION_DESCRIPTOR descr)	获得当前设备的配置描述符。
GetIntfcDescriptor函数	void CCyUSBDevice::GetIntfcDescriptor(PUSB_INTERFACE_DESCRIPTOR descr)	获得当前选择的接口描述符。
GetUSBConfig函数	CCyUSBConfig CCyUSBDevice::GetUSBConfig(int index)	返回配置描述符中的内容。
Interface函数	UCHAR CCyUSBDevice::Interface(void)	返回当前选择的设备接口号。
InterruptInEndPt	CCyInterruptEndPoint* CCyUSBDevice::InterruptInEndPt	指向当前接口的中断IN端点。
InterruptOutEndPt	CCyInterruptEndPoint* CCyUSBDevice::InterruptOutEndPt	指向当前接口的中断OUT端点。
IntfcClass	UCHAR CCyUSBDevice::IntfcClass	当前接口描述符的bInterfaceClass域的值。
IntfcCount函数	UCHAR CCyUSBDevice::IntfcCount(void)	返回USB设备的配置描述符bNumInterfaces域的值。
IntfcProtocol	UCHAR CCyUSBDevice::IntfcProtocol	表示bInterfaceProtocol值。
IntfcSubClass	UCHAR CCyUSBDevice::IntfcSubClass	当前接口描述符IntfcSubClass域的值。
IsocInEndPt	CCyIsocEndPoint* CCyUSBDevice::IsocInEndPt	指向当前接口的同步IN端点。
IsocOutEndPt	CCyIsocEndPoint* CCyUSBDevice::IsocOutEndPt	指向当前接口的同步OUT端点。
IsOpen函数	bool CCyUSBDevice::IsOpen(void)	检查是否获得USB设备的句柄并启动该设备。
Manufacturer	wchar_t CCyUSBDevice::Manufacturer[USB_STRING_MAXLEN]	设备描述符iManufacturer的值。
MaxPacketSize	UCHAR CCyUSBDevice::MaxPacketSize	设备描述符bMaxPacketSize0域的值。
MaxPower	UCHAR CCyUSBDevice::MaxPower	当前配置描述符MaxPower域的值。
Open函数	bool CCyUSBDevice::Open(UCHAR dev)	打开USB设备。
Product	wchar_t CCyUSBDevice::Product[USB_STRING_MAXLEN]	设备描述符iProduct域的值。
ProductID	USHORT CCyUSBDevice::ProductID	设备描述符idProduct域的值。
ReConnect函数	bool CCyUSBDevice::ReConnect(void)	断开USB设备，并进行重连接。
Reset函数	bool CCyUSBDevice::Reset(void)	复位USB设备。
SerialNumber	wchar_t CCyUSBDevice::SerialNumber[USB_STRING_MAXLEN]	设备描述符iSerialNumber域的值。
SetConfig函数	void CCyUSBDevice::SetConfig(UCHAR cfg)	设置当前设备的配置。
SetAltIntfc函数	bool CCyUSBDevice::SetAltIntfc(UCHAR alt)	设置设备接口。
StrLangID	USHORT CCyUSBDevice::StrLangID	设备第一个字符串描述符的bString域的值。
USBAddress	UCHAR CCyUSBDevice::USBAddress	当前打开的USB设备的总线地址。
USBDIVersion	ULONG CCyUSBDevice::USBDIVersion	USB主机驱动器的版本，其值为BCD码的形式。
VendorID	USHORT CCyUSBDevice::VendorID	设备描述符idVendor域的值。

下面介绍常用的成员函数及其在程序设计中的应用。

- BulkInEndPt

BulkInEndPt 为指向第一个块输入端点数据的指针的，其使用示例代码如下：

```
CCyUSBDevice *USBDevice = new    CCyUSBDevice(Handle);     //USB 设备

CCyBulkEndPoint *BulkIn2 = NULL;
```

```
int eptCount = USBDevice->EndPointCount();

for (int i=1; i<eptCount;   i++) {                                    //IN 端点信息
      bool bIn = USBDevice->EndPoints[i]->bIn;
      bool bBulk =   (USBDevice->EndPoints[i]->Attributes == 2);

      if (bBulk   && bIn)      BulkIn2 = (CCyBulkEndPoint *) USBDevice->EndPoints[i];
      if (BulkIn2   == USBDevice->BulkInEndPt)      BulkIn2 = NULL;
 }
```

❑ BulkOutEndPt

BulkOutEndPt 为指向第一个块输出端点数据的指针的，其使用示例代码如下：

```
CCyUSBDevice   *USBDevice = new   CCyUSBDevice(Handle);  //USB 设备

CCyBulkEndPoint *BulkOut2 = NULL;

int   eptCount = USBDevice->EndPointCount();

for   (int   i=1; i<eptCount;   i++) {                                //OUT 端点信息
      bool bIn = USBDevice->EndPoints[i]->Address   & 0x80;
      bool bBulk = (USBDevice->EndPoints[i]->Attributes == 2);

      if (bBulk   && !bIn) BulkOut2 = (CCyBulkEndPoint *) USBDevice->EndPoints[i];
      if (BulkOut2 == USBDevice->BulkOutEndPt) BulkOut2 = NULL;
 }
```

❑ ControlEndPt

ControlEndPt 指向控制端点 0，其使用示例代码如下所示：

```
CCyUSBDevice *USBDevice = new CCyUSBDevice(Handle);

//   Just for typing efficiency
CCyControlEndPoint *ept = USBDevice->ControlEndPt;          //控制端点

ept->Target      = TGT_DEVICE;
ept->ReqType     = REQ_VENDOR;
ept->Direction = DIR_TO_DEVICE;
ept->ReqCode     = 0x05;                                    //请求码
ept->Value       = 1;
ept->Index       = 0;

char   buf[512];
ZeroMemory(buf, 512);
LONG buflen =   512;

ept->XferData(buf,   buflen);                               //传输
```

❑ DeviceCount 函数

DeviceCount 函数用于返回连接到 USB 总线上的 USB 设备个数，其使用示例代码如下所示：

```
USBDevice = new   CCyUSBDevice(Handle);

int devices = USBDevice->DeviceCount( );                    //统计设备个数
```

```
int vID, pID;

int   d = 0;
do    {
      USBDevice->Open(d);                                    //自动打开 USB 设备
      vID = USBDevice->VendorID;
      pID = USBDevice->ProductID
      d++;
}     while ((d < devices ) && (vID != 0x0547) &&   (pID != 0x1002));
```

❑ EndPointCount 函数

EndPointCount 函数返回当前接口的端点数加 1，即包括端点 0。其使用示例代码如下：

```
CCyBulkEndPoint *BulkInEpt = NULL;
CCyBulkEndPoint *BulkOutEpt = NULL;

CCyUSBDevice   *USBDevice = new   CCyUSBDevice(Handle);
int   eptCount = USBDevice->EndPointCount();

// 跳过控制端点 0
for   (int   i=1; i<eptCount;   i++)
          {
      bool bIn = USBDevice->EndPoints[i]->Address   & 0x80;
      bool bBulk = (USBDevice->EndPoints[i]->Attributes == 2);

      if (bBulk && bIn) B ulkInEpt   = (CCyBulkEndPoint *) USBDevice->EndPoints[i];
      if (bBulk && !bIn) BulkOutEpt = (CCyBulkEndPoint *) USBDevice->EndPoints[i];
}
```

❑ EndPointOf 函数

EndPointOf 函数用于返回端点的指针，其使用示例代码如下：

```
UCHAR eptAddr = 0x82;
CCyUSBDevice *USBDevice = new     CCyUSBDevice(Handle);
CCyUSBEndPoint *EndPt =   USBDevice->EndPointOf(eptAddr);
if (EndPt)   EndPt->Reset(   );
```

❑ EndPoints

EndPoints 是端点的列表，其使用示例代码如下：

```
CCyUSBDevice   *USBDevice = new   CCyUSBDevice(Handle);

int   epCnt = USBDevice->EndPointCount();

bool bBulk, bIn;
int   blkInCnt = 0;

for   (int   e=0;   e<epCnt; e++) {
   bBulk = USBDevice->EndPoints[e]->Attributes   == 2;
   bIn     = USBDevice->EndPoints[e]->Address & 0x80;

   if (bBulk && bIn) blkInCnt++;
}
```

❑ GetUSBConfig 函数

GetUSBConfig 函数用于返回配置描述符中的内容，其使用示例代码如下：

```
    CCyUSBDevice    *USBDevice = new    CCyUSBDevice(Handle);

    String s;

    for    (int    c=0; c<USBDevice->ConfigCount();    c++) {
        CCyUSBConfig cfg = USBDevice->GetUSBConfig(c);
      //输出配置信息
        s.sprintf("bLength: 0x%x",cfg.bLength); EZOutputMemo->Lines->Add(s);
        s.sprintf("bDescriptorType: %d",cfg.bDescriptorType); EZOutputMemo->Lines->Add(s);
        s.sprintf("wTotalLength:                %d                (0x%x)",cfg.wTotalLength,cfg.wTotalLength);
EZOutputMemo->Lines->Add(s);
        s.sprintf("bNumInterfaces: %d",cfg.bNumInterfaces); EZOutputMemo->Lines->Add(s);
        s.sprintf("bConfigurationValue: %d",cfg.bConfigurationValue); EZOutputMemo->Lines->Add(s);
        s.sprintf("iConfiguration: %d",cfg.iConfiguration); EZOutputMemo->Lines->Add(s);
        s.sprintf("bmAttributes: 0x%x",cfg.bmAttributes); EZOutputMemo->Lines->Add(s);
        s.sprintf("MaxPower: %d",cfg.MaxPower); EZOutputMemo->Lines->Add(s);
        EZOutputMemo->Lines->Add("**********************************");

        for     (int    i=0; i<cfg.AltInterfaces; i++) {
          CCyUSBInterface    *ifc = cfg.Interfaces[i];
          EZOutputMemo->Lines->Add("Interface    Descriptor:" + String(i+1));
          EZOutputMemo->Lines->Add("--------------------------------");
          s.sprintf("bLength: 0x%x",ifc->bLength); EZOutputMemo->Lines->Add(s);
          s.sprintf("bDescriptorType: %d",ifc->bDescriptorType); EZOutputMemo->Lines->Add(s);
          s.sprintf("bInterfaceNumber: %d",ifc->bInterfaceNumber); EZOutputMemo->Lines->Add(s);
          s.sprintf("bAlternateSetting: %d",ifc->bAlternateSetting); EZOutputMemo->Lines->Add(s);
          s.sprintf("bNumEndpoints: %d",ifc->bNumEndpoints); EZOutputMemo->Lines->Add(s);
          s.sprintf("bInterfaceClass: %d",ifc->bInterfaceClass); EZOutputMemo->Lines->Add(s);

          for     (int    e=0; e<ifc->bNumEndpoints; e++) {
            CCyUSBEndPoint    *ept = ifc->EndPoints[e+1];
            EZOutputMemo->Lines->Add("EndPoint    Descriptor: " + String(e+1));
            EZOutputMemo->Lines->Add("--------------------------------");
            s.sprintf("bLength: 0x%x",ept->DscLen); EZOutputMemo->Lines->Add(s);
            s.sprintf("bDescriptorType: %d",ept->DscType); EZOutputMemo->Lines->Add(s);
            s.sprintf("bEndpointAddress: 0x%x",ept->Address); EZOutputMemo->Lines->Add(s);
            s.sprintf("bmAttributes: 0x%x",ept->Attributes); EZOutputMemo->Lines->Add(s);
            s.sprintf("wMaxPacketSize: %d",ept->MaxPktSize); EZOutputMemo->Lines->Add(s);
            s.sprintf("bInterval: %d",ept->Interval); EZOutputMemo->Lines->Add(s);
            EZOutputMemo->Lines->Add("**********************************");
          }
        }
    }
```

- InterruptInEndPt

InterruptInEndPt 用于指向当前接口的中断 IN 端点，其使用示例代码如下：

```
  CCyUSBDevice    *USBDevice = new    CCyUSBDevice(Handle);

  CCyInterruptEndPoint *IntIn2 = NULL;
```

```
int   eptCount = USBDevice->EndPointCount();

for   (int   i=1; i<eptCount;   i++) {                                    //中断 IN 端点信息
      bool   bIn   = USBDevice->EndPoints[i]->Address   & 0x80;
      bool   bInt = (USBDevice->EndPoints[i]->Attributes == 3);

      if (bInt && bIn)   IntIn2 = (CCyInterruptEndPoint *) USBDevice->EndPoints[i];
      if (IntIn2 == USBDevice->InterruptInEndPt) IntIn2 = NULL;
   }
```

❑ InterruptOutEndPt

InterruptOutEndPt 用于指向当前接口的中断 OUT 端点，其使用示例代码如下：

```
CCyUSBDevice *USBDevice = new CCyUSBDevice(Handle);

CCyInterruptEndPoint *IntOut2 = NULL;
int eptCount = USBDevice->EndPointCount();

for   (int   i=1; i<eptCount;   i++)                                      //中断 OUT 端点信息
{
      bool   bIn = USBDevice->EndPoints[i]->Address   & 0x80;
      bool   bInt =   (USBDevice->EndPoints[i]->Attributes == 3);

      if (bInt   && !bIn) IntOut2 = (CCyInterruptEndPoint *) USBDevice->EndPoints[i];
      if (IntOut2 == USBDevice->InterruptOutEndPt) IntOut2 = NULL;
   }
```

❑ IsocInEndPt

IsocInEndPt 用于指向当前接口的同步 IN 端点，其使用示例代码如下：

```
CCyUSBDevice *USBDevice = new CCyUSBDevice(Handle);

CCyIsocEndPoint *IsocIn2 = NULL;

int eptCount = USBDevice->EndPointCount();

for   (int   i=1; i<eptCount;   i++) {                                    //同步 IN 端点信息
      bool   bIn = USBDevice->EndPoints[i]->Address   & 0x80;
      bool   bIsoc = (USBDevice->EndPoints[i]->Attributes == 1);

      if (bIsoc && bIn) IsocIn2 = (CCyIsocEndPoint *) USBDevice->EndPoints[i];
      if (IsocIn2 == USBDevice->IsocInEndPt) IsocIn2 = NULL;
   }
```

❑ IsocOutEndPt

IsocOutEndPt 用于指向当前接口的同步 OUT 端点，其使用示例代码如下：

```
CCyUSBDevice *USBDevice = new CCyUSBDevice(Handle);

CCyIsocEndPoint *IsocOut2 = NULL;
int eptCount = USBDevice->EndPointCount();

for   (int   i=1; i<eptCount;   i++) {                                    //同步 OUT 端点信息
      bool bIn = USBDevice->EndPoints[i]->Address   & 0x80;
      bool bIsoc = (USBDevice->EndPoints[i]->Attributes == 1);
```

```
        if (bIsoc && !bIn) IsocOut2 = (CCyIsocEndPoint *) USBDevice->EndPoints[i];
        if (IsocOut2 == USBDevice->IsocOutEndPt) IsocOut2 = NULL;
    }
```

❑ Open 函数

Open 用于打开 USB 设备，其使用示例代码如下：

```
CCyUSBDevice *USBDevice =    new CCyUSBDevice(Handle);

 if (USBDevice->DeviceCount()   && !USBDevice->Open(0)) {      //打开设备 0
        USBDevice->Reset();
        USBDevice->Open(0);
 }

 if (! USBDevice->IsOpen()) return false;
```

❑ ProductID 和 VendorID

ProductID 用于表示设备描述符 idProduct 域的值，VendorID 用于表示设备描述符 idVendor 域的值。ProductID 和 VendorID 的使用示例代码如下所示：

```
    USBDevice   = new   CCyUSBDevice(Handle);   // Create an instance of CCyUSBDevice
    int     devices = USBDevice->DeviceCount();
    int     vID, pID;
    int   d =   0;

    do {
        USBDevice->Open(d);                    // 打开 USB 设备
        vID = USBDevice->VendorID;
        pID   = USBDevice->ProductID;
        d++;
    } while ((d < devices ) && (vID != 0x0547) && (pID != 0x1002));
```

6．配置信息类 CCyUSBConfig

配置信息类 CCyUSBConfig 用于表示 USB 设备的配置信息。其定义了很多配置值。配置信息类 CCyUSBConfig 中的配置信息，如表 10.2 所示。

表 10.2　配置信息类CCyUSBConfig的配置信息

配置名	声明原型	功能
AltInterfaces	CCyUSBConfig::AltInterfaces	配置的接口总数。
bConfigurationValue	UCHAR CCyUSBConfig::bConfigurationValue	所选择配置描述符bConfigurationValue域的值。
bDescriptorType	UCHAR CCyUSBConfig::bDescriptorType	配置描述符bDescriptorType域的值。
bLength	UCHAR CCyUSBConfig::bLength	当前选择配置描述符的bLength域的值。
bmAttributes	UCHAR CCyUSBConfig::bmAttributes	当前选择配置描述符的bmAttributes域的值。
bNumInterfaces	UCHAR CCyUSBConfig::bNumInterfaces	当前选择配置描述符bNumInterfaces域的值。
iConfiguration	UCHAR CCyUSBConfig::iConfiguration	当前选择配置描述符iConfiguration域的值。
CCyUSBConfig	CCyUSBConfig::CCyUSBConfig(CCyUSBConfig& cfg)	CCyUSBConfig的类结构。
Interfaces	CCyUSBInterface* CCyUSBConfig::Interfaces[MAX_INTERFACES]	表示各个接口描述符。
wTotalLength	USHORT CCyUSBConfig::wTotalLength	所选配置描述符wTotalLength域的值。

这里最主要的是 Interfaces，其用于表示各个接口描述符。Interfaces 的使用示例代码如下：

```
    CCyUSBDevice    *USBDevice = new    CCyUSBDevice(Handle);

    String s;

    for    (int    c=0; c<USBDevice->ConfigCount();    c++) {
        CCyUSBConfig cfg = USBDevice->GetUSBConfig(c);
//输出接口信息
        s.sprintf("bLength: 0x%x",cfg.bLength); EZOutputMemo->Lines->Add(s);
        s.sprintf("bDescriptorType: %d",cfg.bDescriptorType); EZOutputMemo->Lines->Add(s);
        s.sprintf("wTotalLength:                %d                (0x%x)",cfg.wTotalLength,cfg.wTotalLength);
EZOutputMemo->Lines->Add(s);
        s.sprintf("bNumInterfaces: %d",cfg.bNumInterfaces); EZOutputMemo->Lines->Add(s);
        s.sprintf("bConfigurationValue: %d",cfg.bConfigurationValue); EZOutputMemo->Lines->Add(s);
        s.sprintf("iConfiguration: %d",cfg.iConfiguration); EZOutputMemo->Lines->Add(s);
        s.sprintf("bmAttributes: 0x%x",cfg.bmAttributes); EZOutputMemo->Lines->Add(s);
        s.sprintf("MaxPower: %d",cfg.MaxPower); EZOutputMemo->Lines->Add(s);
        EZOutputMemo->Lines->Add("**********************************");

        for     (int    i=0; i<cfg.AltInterfaces; i++) {
          CCyUSBInterface    *ifc = cfg.Interfaces[i];
          EZOutputMemo->Lines->Add("Interface    Descriptor:" + String(i+1));
          EZOutputMemo->Lines->Add("--------------------------------");
          s.sprintf("bLength: 0x%x",ifc->bLength); EZOutputMemo->Lines->Add(s);
          s.sprintf("bDescriptorType: %d",ifc->bDescriptorType); EZOutputMemo->Lines->Add(s);
          s.sprintf("bInterfaceNumber: %d",ifc->bInterfaceNumber); EZOutputMemo->Lines->Add(s);
          s.sprintf("bAlternateSetting: %d",ifc->bAlternateSetting); EZOutputMemo->Lines->Add(s);
          s.sprintf("bNumEndpoints: %d",ifc->bNumEndpoints); EZOutputMemo->Lines->Add(s);
          s.sprintf("bInterfaceClass: %d",ifc->bInterfaceClass); EZOutputMemo->Lines->Add(s);

          for     (int    e=0; e<ifc->bNumEndpoints; e++) {
            CCyUSBEndPoint    *ept = ifc->EndPoints[e+1];
            EZOutputMemo->Lines->Add("EndPoint    Descriptor: " + String(e+1));
            EZOutputMemo->Lines->Add("--------------------------------");
            s.sprintf("bLength: 0x%x",ept->DscLen); EZOutputMemo->Lines->Add(s);
            s.sprintf("bDescriptorType: %d",ept->DscType); EZOutputMemo->Lines->Add(s);
            s.sprintf("bEndpointAddress: 0x%x",ept->Address); EZOutputMemo->Lines->Add(s);
            s.sprintf("bmAttributes: 0x%x",ept->Attributes); EZOutputMemo->Lines->Add(s);
            s.sprintf("wMaxPacketSize: %d",ept->MaxPktSize); EZOutputMemo->Lines->Add(s);
            s.sprintf("bInterval: %d",ept->Interval); EZOutputMemo->Lines->Add(s);
            EZOutputMemo->Lines->Add("**********************************");
          }
        }
    }
```

7．端点控制类 CCyUSBEndPoint

端点控制类 CCyUSBEndPoint 包含了对 USB 端点的各种描述符及操作函数。端点控制类 CCyUSBEndPoint 中的函数，如表 10.3 所示。

表 10.3　端点控制类CCyUSBEndPoint的函数

函数名	声明原型	功能
Abort函数	void CCyUSBEndPoint::Abort(void)	中止IO传输端点。
Address	UCHAR CCyUSBEndPoint::Address	设备返回的端点描述符bEndpointAddress域的值。
Attributes	UCHAR CCyUSBEndPoint::Attributes	端点描述符bmAttributes域的值。
BeginDataXfer函数	virtual PUCHAR CCyUSBEndPoint::BeginDataXfer(PCHAR buf, LONG len, OVERLAPPED *ov) = 0	用于通过端点进行异步数据传输。
bIn	bool CCyUSBEndPoint::bIn	表示该端点是否为IN端点。
CCyUSBEndPoint	CCyUSBEndPoint::CCyUSBEndPoint(CCyUSBEndPoint& ept)	CCyUSBEndPoint类的构造。
DscLen	UCHAR CCyUSBEndPoint::DscLen	USB端点描述符bLength域的值。
DscType	UCHAR CCyUSBEndPoint::DscType	USB端点描述符bDescriptorType域的值。
GetXferSize函数	ULONG CCyUSBEndPoint::GetXferSize(void)	返回当前端点的传输字节数
FinishDataXfer函数	bool CCyUSBEndPoint::FinishDataXfer(PCHAR buf, LONG &len, OVERLAPPED *ov, PUCHAR pXmitBuf, CCyIsoPktInfo* pktInfos = NULL)	用于异步数据传输中，将数据保存或其他操作。
hDevice	HANDLE CCyUSBEndPoint::hDevice	表示获得的USB句柄。
Interval	UCHAR CCyUSBEndPoint::Interval	端点描述符bInterval域中的值。
MaxPktSize	UCHAR CCyUSBEndPoint::MaxPktSize	表示端点描述符wMaxPacketSize域中的值
Reset函数	bool CCyUSBEndPoint::Reset(void)	复位端点，并清除错误标记。
SetXferSize函数	void CCyUSBEndPoint::SetXferSize(ULONG xfer)	用于设置传输大小。
TimeOut	ULONG CCyUSBEndPoint::TimeOut	用于表示传输等待的时间。
WaitForXfer函数	bool CCyUSBEndPoint::WaitForXfer(OVERLAPPED *ov, ULONG tOut)	用于等待异步通信结束。
XferData函数	bool CCyUSBEndPoint::XferData(PCHAR buf, LONG &len, PUCHAR pktInfos = NULL)	用于进行同步IO数据传输。

下面介绍程序设计中常用的函数及其用法。

❑ Abort 函数

Abort 函数用于中止 IO 传输端点，其使用示例代码如下：

```
CCyUSBDevice   *USBDevice = new CCyUSBDevice(Handle);

USBDevice->ControlEndPt->Abort();                                    //停止端点
```

❑ Address

Address 用于表示设备返回的端点描述符 bEndpointAddress 域的值，其高位如果为（0x8_）则表示 IN 端点，如果高位为（0x0_）则表示 OUT 端点。Address 的使用代码，示例如下：

```
CCyUSBDevice   *USBDevice = new   CCyUSBDevice(Handle);

CCyBulkEndPoint *BulkIn2 = NULL;

int eptCount = USBDevice->EndPointCount();

for   (int   i=1; i<eptCount;   i++) {
     bool   bIn = USBDevice->EndPoints[i]->Address & 0x80;       //端点地址
     bool   bBulk = (USBDevice->EndPoints[i]->Attributes == 2);

     if (bBulk   && bIn) BulkIn2 = (CCyBulkEndPoint *) USBDevice->EndPoints[i];
     if (BulkIn2 == BulkInEndPt) BulkIn2 = NULL;
 }
```

❑ Attributes

Attributes 用于表示端点描述符 bmAttributes 域的值，其共有 4 个可选值：0 表示控制端点、1 表示同步端点、2 表示块端点、3 中断端点。Attributes 的使用代码，示例如下：

```
CCyUSBDevice   *USBDevice = new   CCyUSBDevice(Handle);

CCyBulkEndPoint *BulkIn2 = NULL;

int   eptCount = USBDevice->EndPointCount();

for   (int   i=1; i<eptCount;   i++) {
      bool   bIn = USBDevice->EndPoints[i]->bIn;
      bool   bBulk = (USBDevice->EndPoints[i]->Attributes == 2);

      if (bBulk && bIn) BulkIn2 = (CCyBulkEndPoint *) USBDevice->EndPoints[i];
      if (BulkIn2 == BulkInEndPt) BulkIn2 = NULL;
 }
```

❑ BeginDataXfer 函数、WaitForXfer 函数和 FinishDataXfer 函数

BeginDataXfer 函数用于通过端点进行异步数据传输，WaitForXfer 函数用于等待异步通信结束，FinishDataXfer 函数用于异步数据传输中将数据保存或其他操作。这三个函数经常在一起使用，其使用示例代码如下：

```
CCyUSBDevice   *USBDevice = new   CCyUSBDevice(Handle);

OVERLAPPED outOvLap, inOvLap;
outOvLap.hEvent   = CreateEvent(NULL, false, false, "CYUSB_OUT");
inOvLap.hEvent    = CreateEvent(NULL, false, false, "CYUSB_IN");

char   inBuf[128];
ZeroMemory(inBuf,   128);

char buffer[128];
LONG   length = 128;

//开始传输
UCHAR   *inContext = USBDevice->BulkInEndPt->BeginDataXfer(inBuf, length, &inOvLap);
UCHAR   *outContext = USBDevice->BulkOutEndPt->BeginDataXfer(buffer, length, &outOvLap);

//等待
USBDevice->BulkOutEndPt->WaitForXfer(&outOvLap,100);
USBDevice->BulkInEndPt->WaitForXfer(&inOvLap,100);

//传输结束
USBDevice->BulkOutEndPt->FinishDataXfer(buffer, length, &outOvLap,outContext);
USBDevice->BulkInEndPt->FinishDataXfer(inBuf, length, &inOvLap,inContext);

CloseHandle(outOvLap.hEvent);
CloseHandle(inOvLap.hEvent);
```

❑ bIn

bIn 用于表示该端点是否为 IN 端点，其使用代码，示例如下：

```
CCyUSBDevice   *USBDevice = new   CCyUSBDevice(Handle);

CCyBulkEndPoint *BulkIn2 = NULL;

int   eptCount = USBDevice->EndPointCount();

for   (int   i=1; i<eptCount;   i++) {
      bool In = USBDevice->EndPoints[i]->bIn;                          //是否为 IN 端点
      bool bBulk = (USBDevice->EndPoints[i]->Attributes == 2);

      if (bBulk   && In) BulkIn2 = (CCyBulkEndPoint *) USBDevice->EndPoints[i];
      if (BulkIn2   == USBDevice->BulkInEndPt) BulkIn2 = NULL;
  }
```

❑ TimeOut

TimeOut 用于表示传输等待的时间，其使用代码，示例如下：

```
  char buffer[128];
  LONG length = 128;

  CCyUSBDevice *USBDevice = new CCyUSBDevice(Handle);

  USBDevice->BulkOutEndPt->TimeOut = 1000;                  //timeout 时间设置为 1s
  USBDevice->BulkOutEndPt->XferData(buf, length);
```

❑ XferData 函数

XferData 函数用于进行同步 IO 数据传输，其使用代码，示例如下：

```
CCyUSBDevice *USBDevice = new CCyUSBDevice(Handle);

char buf[12] = "hello   world";
LONG length = 11;

if (USBDevice->BulkOutEndPt)
   USBDevice->BulkOutEndPt->XferData(buf, length);          //同步传输
```

8．接口控制类 CCyUSBInterface

接口控制类 CCyUSBInterface 表示了 USB 设备的接口，其中定义很多函数及变量。接口控制类 CCyUSBInterface 中的函数，如表 10.4 所示。

表 10.4　接口控制类CCyUSBInterface中的函数

函数名	声明原型	功能
bAlternateSetting	UCHAR CCyUSBInterface::bAlternateSetting	当前所选的接口bAlternateSetting域的值。
bAltSettings	UCHAR CCyUSBInterface::bAltSettings	表示接口的个数。
bDescriptorType	UCHAR CCyUSBInterface::bDescriptorType	USB描述符bDescriptorType域的值。
CCyUSBInterface	CCyUSBInterface::CCyUSBInterface:(CCyUSBInterface& intfc)	CCyUSBInterface类的构造
bInterfaceClass	UCHAR CCyUSBInterface::bInterfaceClass	当前接口描述符bInterfaceClass域的值
bInterfaceNumber	UCHAR CCyUSBInterface::bInterfaceNumber	当前接口描述符bInterfaceNumber域的值。
bInterfaceProtocol	UCHAR CCyUSBInterface::bInterfaceProtocol	当前接口描述符bInterfaceProtocol域的值。
bInterfaceSubClass	UCHAR CCyUSBInterface:: bInterfaceSubClass	当前接口描述符bInterfaceSubClass域的值。
bLength	UCHAR CCyUSBInterface:: bLength	当前接口描述符bLength域的值
bNumEndpoints	UCHAR CCyUSBInterface:: bNumEndpoints	当前接口描述符bNumEndpoints域的值。
EndPoints	CCyUSBEndPoint*	表示接口的端点参数。

	CCyUSBInterfac::EndPoints[MAX_ENDPTS]	
iInterface	UCHAR CCyUSBInterface::iInterface	当前所选接口描述符iInterface域的值。

其中最常用的是 EndPoints，其用于表示接口的端点参数。EndPoints 的使用代码，示例如下：

```
    CCyUSBDevice    *USBDevice = new    CCyUSBDevice(Handle);

    String s;

    for    (int    c=0; c<USBDevice->ConfigCount();    c++) {
        CCyUSBConfig cfg = USBDevice->GetUSBConfig(c);
//接口信息
        s.sprintf("bLength: 0x%x",cfg.bLength); EZOutputMemo->Lines->Add(s);
        s.sprintf("bDescriptorType: %d",cfg.bDescriptorType); EZOutputMemo->Lines->Add(s);
        s.sprintf("wTotalLength:                %d                (0x%x)",cfg.wTotalLength,cfg.wTotalLength);
EZOutputMemo->Lines->Add(s);
        s.sprintf("bNumInterfaces: %d",cfg.bNumInterfaces); EZOutputMemo->Lines->Add(s);
        s.sprintf("bConfigurationValue: %d",cfg.bConfigurationValue); EZOutputMemo->Lines->Add(s);
        s.sprintf("iConfiguration: %d",cfg.iConfiguration); EZOutputMemo->Lines->Add(s);
        s.sprintf("bmAttributes: 0x%x",cfg.bmAttributes); EZOutputMemo->Lines->Add(s);
        s.sprintf("MaxPower: %d",cfg.MaxPower); EZOutputMemo->Lines->Add(s);
        EZOutputMemo->Lines->Add("**********************************");

        for     (int    i=0; i<cfg.AltInterfaces; i++) {
          CCyUSBInterface    *ifc = cfg.Interfaces[i];
          EZOutputMemo->Lines->Add("Interface    Descriptor:" + String(i+1));
          EZOutputMemo->Lines->Add("--------------------------------");
          s.sprintf("bLength: 0x%x",ifc->bLength); EZOutputMemo->Lines->Add(s);
          s.sprintf("bDescriptorType: %d",ifc->bDescriptorType); EZOutputMemo->Lines->Add(s);
          s.sprintf("bInterfaceNumber: %d",ifc->bInterfaceNumber); EZOutputMemo->Lines->Add(s);
          s.sprintf("bAlternateSetting: %d",ifc->bAlternateSetting); EZOutputMemo->Lines->Add(s);
          s.sprintf("bNumEndpoints: %d",ifc->bNumEndpoints); EZOutputMemo->Lines->Add(s);
          s.sprintf("bInterfaceClass: %d",ifc->bInterfaceClass); EZOutputMemo->Lines->Add(s);

          for     (int    e=0; e<ifc->bNumEndpoints; e++) {
            CCyUSBEndPoint    *ept = ifc->EndPoints[e+1];
            EZOutputMemo->Lines->Add("EndPoint    Descriptor: " + String(e+1));
            EZOutputMemo->Lines->Add("--------------------------------");
            s.sprintf("bLength: 0x%x",ept->DscLen); EZOutputMemo->Lines->Add(s);
            s.sprintf("bDescriptorType: %d",ept->DscType); EZOutputMemo->Lines->Add(s);
            s.sprintf("bEndpointAddress: 0x%x",ept->Address); EZOutputMemo->Lines->Add(s);
            s.sprintf("bmAttributes: 0x%x",ept->Attributes); EZOutputMemo->Lines->Add(s);
            s.sprintf("wMaxPacketSize: %d",ept->MaxPktSize); EZOutputMemo->Lines->Add(s);
            s.sprintf("bInterval: %d",ept->Interval); EZOutputMemo->Lines->Add(s);
            EZOutputMemo->Lines->Add("**********************************");
          }
        }
    }
```

10.2 Visual C#读写 USB 设备

SuiteUSB.NET 2.0 开发包提供的 EZ-USB 驱动，可供在 Visual Studio 2005 或者更新的平台下进行访问。因此，利用 SuiteUSB.NET 2.0 开发包可以在 Visual C#语言中对 USB 设备进行读写。SuiteUSB.NET 2.0 开发包使用 CyUSB.dll 来读写 USB 设备。SuiteUSB.NET 2.0 开发包是对 CY3684 开发包的扩展，其中提供了更多的功能。下面主要介绍一些主要的新功能函数，其余部分和 CY3684 开发包类似。

10.2.1 CyHidDevice 类

CyHidDevice类提供了USB人机接口设备(HID)的控制，包括USB鼠标和USB键盘等。CyHidDevice类中的函数，如表 10.5 所示。

10.5 CyHidDevice类函数

函数名	声明原型	功能
GetFeature函数	bool GetFeature (int rptID)	表示HID设备的HID特性。
GetInput函数	bool GetInput (int rptID)	读取HID设备控制值。
ReadInput函数	bool ReadInput ( )	读取HID设备控制。
SetFeature函数	bool SetFeature(CyUSB.CyHidValue hidVal, uint val)	设置HID设备的特性。
SetOutput函数	System.Boolean SetOutput (int rptID)	提供了HID设备只写接口的控制。
ToString函数	override string ToString()	返回XML格式的HID设备描述符。
WriteOutput函数	bool WriteOutput ( )	提供了HID设备只写接口的控制。
Capabilities函数	CyUSB.HIDP_CAPS Capabilities { get; }	返回HIDP_CAPS信息。
Inputs	CyUSB.CyHidReport Inputs { get; }	HID设备HIDP_CAPS的所有输入控制。
Outputs	CyUSB.CyHidReport Outputs { get; }	HID设备HIDP_CAPS的所有输处控制。
RwAccessible函数	bool RwAccessible { get; }	是否可以打开一个可供读写的设备。
Usage	ushort Usage { get; }	HIDP_CAPS中的16b Usage字段。
UsagePage	ushort UsagePage { get; }	HIDP_CAPS中的16b UsagePage字段。
Version	ushort Version { get; }	HID设备的VersionNumber字段。

下面介绍常用的控制函数及其在 C#语言中的应用。

1．GetFeature 函数

GetFeature 函数表示 HID 设备的 HID 特性。GetFeature 函数在 C#语言中的使用示例代码如下：

```
USBDeviceList HidDevices = new USBDeviceList(CyConst.DEVICES_HID);    //设备列表
CyHidDevice hidAnalyzer = HidDevices[0] as CyHidDevice;

if ((hidAnalyzer != null) && hidAnalyzer.GetFeature(0))                    //GetFeature 函数
    for (int x=1; x < hidAnalyzer.Features.RptByteLen; x++)
        RawData[Vx++] = hidAnalyzer.Features.DataBuf[x];
```

2．GetInput 函数

GetInput 函数读取 HID 设备控制值。GetInput 函数在 C#语言中的使用示例代码如下：

```
USBDeviceList HidDevices = new USBDeviceList(CyConst.DEVICES_HID);    //设备列表
CyHidDevice hidDev = HidDevices[0] as CyHidDevice;

string s = "";
```

```
if (hidDev.GetInput(hidDev.Inputs.ID))                          //读取 HID 控制值
for (int i=1; i < hidDev.Inputs.RptByteLen; i++)
s += hidDev.Inputs.DataBuf[i].ToString("X2") + " ";
```

3．ReadInput 函数

ReadInput 函数用于读取 HID 设备控制。ReadInput 函数在 C#语言中的使用示例代码如下：

```
USBDeviceList HidDevices = new USBDeviceList(CyConst.DEVICES_HID);
CyHidDevice hidDev = HidDevices[0] as CyHidDevice;

hidDev.ReadInput();                                             // ReadInput 函数
```

4．SetFeature 函数

SetFeature 函数用于设置 HID 设备的特性。SetFeature 函数在 C#语言中的使用示例代码如下：

```
USBDeviceList HidDevices = new USBDeviceList(CyConst.DEVICES_HID);    //设备类别
CyHidDevice hidDev = HidDevices[0] as CyHidDevice;

CyHidValue hVal = hidDev.Features.Values[0];
if (hVal != null)
    hidDev.SetFeature(hVal, 0x20);                              //设置 HID 设备特性
```

5．SetOutput 函数

SetOutput 函数提供了 HID 设备只写接口的控制。在使用时，必须先将数据保存在 DataBuf 中。SetOutput 函数在 C#语言中的使用示例代码如下：

```
USBDeviceList HidDevices = new USBDeviceList(CyConst.DEVICES_HID);    //设备列表
CyHidDevice hidDev = HidDevices[0] as CyHidDevice;

hidDev.Outputs.DataBuf[0] = hidDev.Outputs.ID;
hidDev.Outputs.DataBuf[1] = 0x01;
hidDev.Outputs.DataBuf[2] = 0x02;

hidDev.SetOutput(hidDev.Outputs.ID);                            // SetOutput 函数
```

6．ToString 函数

ToString 函数用于返回 XML 格式的 HID 设备描述符。ToString 函数在 C#语言中的使用示例代码如下：

```
USBDeviceList HidDevices = new USBDeviceList(CyConst.DEVICES_HID);
CyHidDevice hidDev = HidDevices[0] as CyHidDevice;

string text = hidDev.ToString();
```

ToString 函数的典型返回信息，首先是设备描述符、VID、PID、类协议以及键值，示例如下：

```
<HID_DEVICE>
    FriendlyName=""
    Manufacturer="Logitech"                                     //制造商
    Product="USB-PS/2 Optical Mouse"
    SerialNumber="?"
    VendorID="0x046D"                                           //VID 值
    ProductID="0xC03D"                                          //PID 值
    Class="0x00"
    SubClass="0x00"
```

```
Protocol="0x00"
BcdUSB="0x0000"
Usage="0x0002"
UsagePage="0x0001"
Version="0x2000"                                        //版本号
<INPUT>
    RptByteLen="5"
    Buttons="1"
    Values="3"
    <BUTTON>                                            //BUTTON 信息
        Usage="0x0001"
        UsagePage="0x0009"
        UsageMax="0x0003"
        BitField="0x0002"
        LinkCollection="0x0001"
        LinkUsage="0x0001"
        LinkUsagePage="0x0001"
        IsAlias="False"
        IsRange="True"
        IsStringRange="False"
        IsDesignatorRange="False"
        IsAbsolute="True"
        StringIndex="0"
        StringMax="0"
        DesignatorIndex="0"
        DesignatorMax="0"
        DataIndex="0"
        DataIndexMax="2"
    </BUTTON>
```

其中，定义了 1 个按键 3 个值。下面便开始具体的 HID 键值的信息，第一个 VALUE 信息示例如下：

```
    <VALUE>                                             //VALUE 信息 1
        Usage="0x0038"
        UsagePage="0x0001"
        UsageMax="0x0038"
        BitField="0x0006"
        LinkCollection="0x0001"
        LinkUsage="0x0001"
        LinkUsagePage="0x0001"
        IsAlias="False"
        IsRange="False"
        IsStringRange="False"
        IsDesignatorRange="False"
        IsAbsolute="False"
        HasNull="False"
        StringIndex="0"
        StringMax="0"
        DesignatorIndex="0"
        DesignatorMax="0"
        DataIndex="3"
        DataIndexMax="3"
```

```
            BitField="0x0006"
            LinkCollection="0x0001"
            LinkUsage="0x0001"
            LinkUsagePage="0x0001"
            BitSize="8"
            ReportCount="1"
            Units="0"
            UnitsExp="0"
            LogicalMin="-127"
            LogicalMax="127"
            PhysicalMin="0"
            PhysicalMax="0"
        </VALUE>
```

第二个 HID 键值的 VALUE 信息示例如下：

```
        <VALUE>                                                    //VALUE 信息 2
            Usage="0x0031"
            UsagePage="0x0001"
            UsageMax="0x0031"
            BitField="0x0006"
            LinkCollection="0x0001"
            LinkUsage="0x0001"
            LinkUsagePage="0x0001"
            IsAlias="False"
            IsRange="False"
            IsStringRange="False"
            IsDesignatorRange="False"
            IsAbsolute="False"
            HasNull="False"
            StringIndex="0"
            StringMax="0"
            DesignatorIndex="0"
            DesignatorMax="0"
            DataIndex="4"
            DataIndexMax="4"
            BitField="0x0006"
            LinkCollection="0x0001"
            LinkUsage="0x0001"
            LinkUsagePage="0x0001"
            BitSize="8"
            ReportCount="1"
            Units="0"
            UnitsExp="0"
            LogicalMin="-127"
            LogicalMax="127"
            PhysicalMin="0"
            PhysicalMax="0"
        </VALUE>
```

第三个 HID 键值的 VALUE 信息示例如下：

```
        <VALUE>                                                    //VALUE 信息 3
            Usage="0x0030"
```

```
            UsagePage="0x0001"
            UsageMax="0x0030"
            BitField="0x0006"
            LinkCollection="0x0001"
            LinkUsage="0x0001"
            LinkUsagePage="0x0001"
            IsAlias="False"
            IsRange="False"
            IsStringRange="False"
            IsDesignatorRange="False"
            IsAbsolute="False"
            HasNull="False"
            StringIndex="0"
            StringMax="0"
            DesignatorIndex="0"
            DesignatorMax="0"
            DataIndex="5"
            DataIndexMax="5"
            BitField="0x0006"
            LinkCollection="0x0001"
            LinkUsage="0x0001"
            LinkUsagePage="0x0001"
            BitSize="8"
            ReportCount="1"
            Units="0"
            UnitsExp="0"
            LogicalMin="-127"
            LogicalMax="127"
            PhysicalMin="0"
            PhysicalMax="0"
        </VALUE>
    </INPUT>
</HID_DEVICE>
```

7．WriteOutput 函数

WriteOutput 函数提供了 HID 设备只写接口的控制，在使用时必须先将数据保存在 DataBuf 中。WriteOutput 函数在 C#语言中的使用示例代码如下：

```
USBDeviceList HidDevices = new USBDeviceList(CyConst.DEVICES_HID);
CyHidDevice hidDev = HidDevices[0] as CyHidDevice;

hidDev.Outputs.DataBuf[1] = 0x04;                                    //数据
hidDev.Outputs.DataBuf[2] = 0x06;                                    //数据

hidDev.WriteOutput();                                                //WriteOutput 函数
```

10.2.2 CyIsocEndPoint 类

同步传输端点控制类 CCyIsocEndPoint 为 CCyUSBEndPoint 类的一个子类。同步传输端点控制类 CCyIsocEndPoint 提供了对同步端点的定义和函数支持，CyIsocEndPoint 类函数列表，如表 10.6 所示。

表 10.6　CyIsocEndPoint类函数列表

函数名	声明原型	功能
BeginDataXfer	bool BeginDataXfer (ref byte[] singleXfer , ref byte[] buffer , ref int len , ref byte[] ov)	用于通过端点进行异步数据传输。
FinishDataXfer	bool FinishDataXfer (ref byte[] singleXfer , ref byte[] buffer , ref int len , ref byte[] ov , ref CyAPI.ISO_PKT_INFO[] pktInfo)	用于异步数据传输中，将数据保存或其他操作。
GetPktBlockSize	int GetPktBlockSize (int len)	返回同步数据传输的包大小。
GetPktCount	int GetPktCount (int len)	返回同步数据传输包数量。
XferData	bool XferData (ref byte[] buf , ref int len , ref CyUSB.ISO_PKT_INFO[] pktInfos)	同步数据传输。
XferData	bool XferData (ref byte[] buf , ref int len)	同步数据传输。

下面介绍常用的控制函数及其在 C#语言中的应用。

1．BeginDataXfer 函数、FinishDataXfer 函数和 GetPktBlockSize 函数

BeginDataXfer 函数、FinishDataXfer 函数和 GetPktBlockSize 函数用于异步数据传输。这几个函数在C#语言中的使用示例代码如下：

```
public unsafe void ListenThread()                                                    //线程
{
if (MyDevice == null) return;

 CyIsocEndPoint InEndpt = MyDevice.IsocInEndPt;                                      //端点

byte i = 0;

int BufSz = InEndpt.MaxPktSize * Convert.ToUInt16(PpxBox.Text);
int QueueSz = Convert.ToUInt16(QueueBox.Text);

InEndpt.XferSize = BufSz;

//设置缓冲区

 byte[][] cmdBufs              = new byte[QueueSz][];
 byte[][] xferBufs             = new byte[QueueSz][];
 byte[][] ovLaps               = new byte[QueueSz][];
 ISO_PKT_INFO[][] pktInfos     = new ISO_PKT_INFO[QueueSz][];

 for (i=0; i<QueueSz; i++)
 {
 cmdBufs[i]     = new byte[CyConst.SINGLE_XFER_LEN + InEndpt.GetPktBlockSize(BufSz)];
 pktInfos[i]    = new ISO_PKT_INFO[InEndpt.GetPktCount(BufSz)];
 xferBufs[i]    = new byte[BufSz];
 ovLaps[i]      = new byte[20];

 fixed( byte *tmp0 = ovLaps[i])
 {
 OVERLAPPED *ovLapStatus = (OVERLAPPED*) tmp0;
 ovLapStatus->hEvent = (uint) PInvoke.CreateEvent(0, 0, 0, 0);
 }
}
```

```
//预加载
int len = BufSz;

for (i=0; i<QueueSz; i++)
    InEndpt.BeginDataXfer(ref cmdBufs[i], ref xferBufs[i], ref len, ref ovLaps[i]);        // BeginDataXfer 函数

i = 0;
int Successes = 0;
int Failures = 0;

XferBytes = 0;
t1 = DateTime.Now;

for (;StartBtn.Text.Equals("Stop");)                                                        //停止
{
fixed( byte *tmp0 = ovLaps[i])
{
OVERLAPPED *ovLapStatus = (OVERLAPPED*) tmp0;
if (! InEndpt.WaitForXfer(ovLapStatus->hEvent,500))                                         // WaitForXfer 函数
{
InEndpt.Abort();
PInvoke.WaitForSingleObject(ovLapStatus->hEvent,500);
}
}
// FinishDataXfer 函数
if (InEndpt.FinishDataXfer(ref cmdBufs[i], ref xferBufs[i], ref len, ref ovLaps[i], ref pktInfos[i]))
{
XferBytes += len;
Successes++;
// Add code to examine each ISO_PKT_INFO here
}
else
  Failures++;

//重新提交
len = BufSz;
InEndpt.BeginDataXfer(ref cmdBufs[i], ref xferBufs[i], ref len, ref ovLaps[i]);

i++;

if (i == QueueSz)
{
i = 0;
t2 = DateTime.Now;

elapsed = t2-t1;
xferRate = (long)(XferBytes / elapsed.TotalMilliseconds) ;
xferRate = xferRate / (int)100 * (int)100;
```

```
 if (xferRate > ProgressBar.Maximum)
   ProgressBar.Maximum = (int)(xferRate * 1.25);

 ProgressBar.Value = (int) xferRate;
 ThroughputLabel.Text = ProgressBar.Value.ToString();

 SuccessBox.Text = Successes.ToString();
 FailuresBox.Text = Failures.ToString();

 Thread.Sleep(0);
 }
}
}
```

2．GetPktCount 函数

GetPktCount 函数返回同步数据传输包数量。GetPktCount 函数在 C#语言中的使用示例代码如下：

```
USBDeviceList    usbDevices    = new USBDeviceList(CyConst.DEVICES_CYUSB);
CyUSBDevice        MyDevice    = usbDevices[0x04B4,0x1003] as CyUSBDevice;   //USB 设备

if (MyDevice != null)
if (MyDevice.IsocInEndPt != null)
{
int len = MyDevice.IsocInEndPt.MaxPktSize * 8;

byte [] buf = new byte[len];

ISO_PKT_INFO[] pkInfos = new ISO_PKT_INFO[MyDevice.IsocInEndPt.GetPktCount(len)]; //GetPktCount 函数

MyDevice.IsocInEndPt.XferSize = len;

MyDevice.IsocInEndPt.XferData(ref buf, ref leng, ref pkInfos);                    //同步传输
}
```

3．XferData 函数

XferData 函数用于同步数据传输。XferData 函数有两种声明格式，一个返回 ISO_PKT_INFO 结构的数组，另一个则不返回。

在 C#中使用返回 ISO_PKT_INFO 结构数组的 XferData 函数的示例代码如下：

```
USBDeviceList    usbDevices    = new USBDeviceList(CyConst.DEVICES_CYUSB);
CyUSBDevice      MyDevice      = usbDevices[0x04B4,0x1003] as CyUSBDevice;   //USB 设备

if (MyDevice != null)
if (MyDevice.IsocInEndPt != null)
{
int len = MyDevice.IsocInEndPt.MaxPktSize * 8;

byte [] buf = new byte[len];

ISO_PKT_INFO[] pkInfos = new ISO_PKT_INFO[MyDevice.IsocInEndPt.GetPktCount(len)];

MyDevice.IsocInEndPt.XferSize = len;                                              //长度
```

```
MyDevice.IsocInEndPt.XferData(ref buf, ref leng, ref pkInfos);                    //XferData 函数
}
```

在 C#中使用不返回 ISO_PKT_INFO 结构数组的 XferData 函数的示例代码如下：

```
USBDeviceList    usbDevices    = new USBDeviceList(CyConst.DEVICES_CYUSB);
CyUSBDevice      MyDevice      = usbDevices[0x04B4,0x1003] as CyUSBDevice;   //USB 设备

if (MyDevice != null)
if (MyDevice.IsocInEndPt != null)
{
int len = MyDevice.IsocInEndPt.MaxPktSize * 8;

byte [] buf = new byte[len];

MyDevice.IsocInEndPt.XferSize = len;                                          //长度

MyDevice.IsocInEndPt.XferData(ref buf, ref len);                              //XferData 函数
}
```

10.2.3 CyUSBStorDevice 类

CyUSBStorDevice 类表示 USB 大容量存储设备类，其使用了微软大容量存储设备驱动 usbstor.sys。因此，使用 SuiteUSB.NET 2.0 开发包可以在 C#中使用系统标准的驱动程序，如表 10.7 所示。

表 10.7　CyUSBStorDevice类函数列表

函数	声明原型	功能
SendScsiCmd函数	bool SendScsiCmd(byte cmd, byte op, byte lun, byte dirIn, int bank, int lba, int bytes, byte[] data)	向设备发送命令。
ToString函数	string ToString( )()	返回XML格式的存储设备信息。
BlockSize	int BlockSize { get; }	表示存储设备的数据块大小。
TimeOut	uint TimeOut { set; get; }	表示SCSI_PASS_THROUGH结构中的TimeOutValue

在 C#语言中，可以通过如下语句创建一个由 usbstor.sys 支持的设备列表。

```
USBDeviceList devList = new USBDeviceList(CyConst.DEVICES_MSC);

 if (devList.Count == 0) return;

  CyUSBStorDevice StorDevice = usbDevices[0] as CyUSBStorDevice;

  string SerNum = StorDevice.SerialNumber;
```

下面介绍常用的控制函数及其在 C#语言中的应用。

1．SendScsiCmd 函数

SendScsiCmd 函数向设备发送命令，其使用 CDB10 命令结构。SendScsiCmd 函数在 C#语言中的使用示例代码如下：

```
USBDeviceList devList = new USBDeviceList(CyConst.DEVICES_MSC);   //创建设备列表
if (devList.Count == 0) return;
```

```
CyUSBStorDevice StorDevice = usbDevices[0] as CyUSBStorDevice;

public const byte CMD_READ     = 0x28;                                    //参数信息
byte opCode                = 0;
byte lun                   = 0;
byte dirIn                 = 1;
int bank                   = 0;
int lba                  = 0;
int xferSz                 = 512;
byte [] data               = new byte[xferSz];
//调用 SendScsiCmd 函数
StorDevice.SendScsiCmd(CMD_READ, opCode, lun, dirIn, bank, lba, xferSz, data);
```

2．ToString 函数

ToString 函数返回 XML 格式的存储设备信息。ToString 函数在 C#语言中的使用示例代码如下：

```
USBDeviceList devList = new USBDeviceList(CyConst.DEVICES_MSC); //USB 设备列表

CyUSBStorDevice StorDevice = usbDevices[0] as CyUSBStorDevice;

DescText.Text = StorDevice.ToString();                                    //ToString 函数
```

常见的典型返回信息示例如下：

```
<MSC_DEVICE>
    FriendlyName="Generic USB CF Reader USB Device"
    Manufacturer="Compatible USB storage device"                          //制造商
    Product="USB Reader"
    SerialNumber="2004888"
    VendorID="0x058F"                                                     //VID 值
    ProductID="0x9360"                                                    //PID 值
    Class="0x08"                                                          //类
    SubClass="0x06"                                                       //子类
    Protocol="0x50"
    BcdUSB="0x0100"
</MSC_DEVICE>
```

5．TimeOut

TimeOut 表示了 SCSI_PASS_THROUGH 结构中的 TimeOutValue 值，其在 C#语言中的使用示例代码如下：

```
USBDeviceList devList = new USBDeviceList(CyConst.DEVICES_MSC); //USB 设备列表
if (devList.Count == 0) return;

CyUSBStorDevice StorDevice = usbDevices[0] as CyUSBStorDevice;

StorDevice.Timeout = 10;                                                  //设置 10s
```

10.3　LabVIEW 读写 USB 设备

LabVIEW 开发平台提供了极其简单方便的硬件接口，特别是对复杂的 USB 接口协议。如果采用 NI-VISA 开发了 USB 设备的硬件驱动程序，则可以在 LabVIEW 中直接对该 USB 设备进行操作。需要

注意的是，如果采用 Cypress 公司的 EZ-USB 系列芯片，则需要首先安装固件下载驱动程序，然后在 NI-VISA 中安装 LabVIEW 平台下的驱动程序。这样，固件下载驱动程序负责下载 USB 设备的功能固件程序，NI-VISA 驱动程序负责与 LabVIEW 程序进行通信。

10.3.1 USB 设备测试

当使用 NI-VISA 安装完 USB 设备的驱动程序后，便可以在 LabVIEW 中进行读写操作。NI-VISA 中还提供了一个简单有效的工具 VISA Interactive Control，供用户测试 USB 设备的功能。这个小工具提供了多种硬件接口的测试。下面便介绍对于 USB 设备的测试，其步骤如下：

（1）首先连接 USB 设备至计算机。

（2）打开 VISA Interactive Control 软件，程序将自动检测 LabVIEW 平台下可用的硬件接口资源，并将其显示出来，如图 10.1 所示。

（3）在 VISA I/O 中寻找 USB 设备并选中该设备，则在下面的“Resource to Open”文本框中显示该 USB 设备的资源名称“USB0::0x1234::0x1111::NI-VISA-0::RAW”。在 LabVIEW 程序中，便是通过这个资源名称打开 USB 设备并进行操作的。

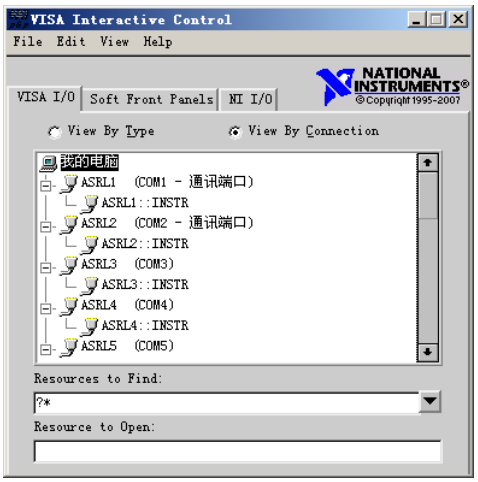

图 10.1 VISA Interactive Control 软件

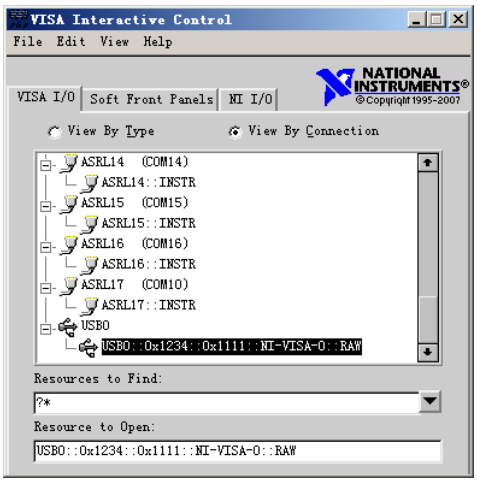

图 10.2 USB 设备

（4）双击该 USB 设备，弹出 USB 设备测试程序，如图 10.3 所示。该测试程序分为三个主要的部分：Template、Basic I/O 和 Interface I/O。

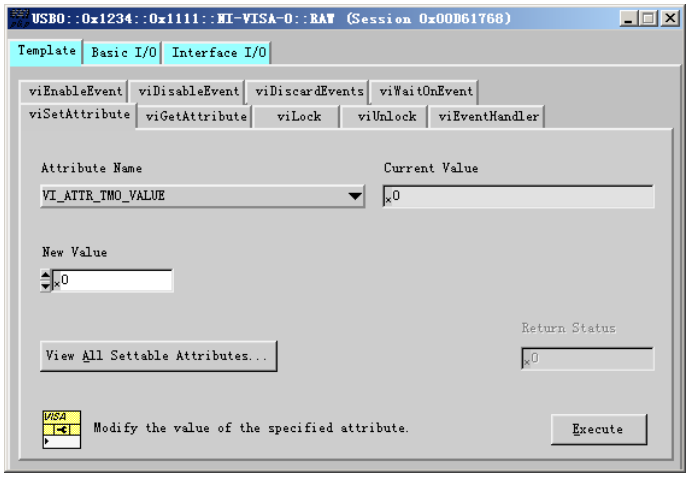

图 10.3　USB 设备测试程序

（5）在 Template 中，可以查询并设置该 USB 设备的属性等信息。例如，打开 viGetAttribute 页面，选择 Attribute Name 为 VI_ATTR_RSRC_MANF_NAME。

（6）单击“Execute”按钮，便显示制造商名称 National Instruments，如图 10.4 所示。

（7）在 Basic I/O 中，可以进行 USB 数据传输的测试。在 viWrite 页面中单击“Execute”按钮，便可以将“Buffer”文本框中的数据发送到外部 USB 设备，如图 10.5 所示。

（8）打开 viRead 页面，单击“Execute”按钮，便可以从 USB 设备读取数据并显示在“Buffer”中，如图 10.6 所示。

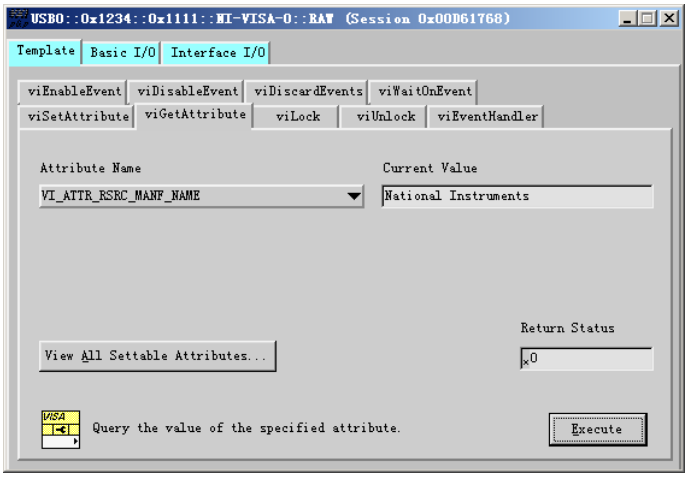

图 10.4　显示制造商名称

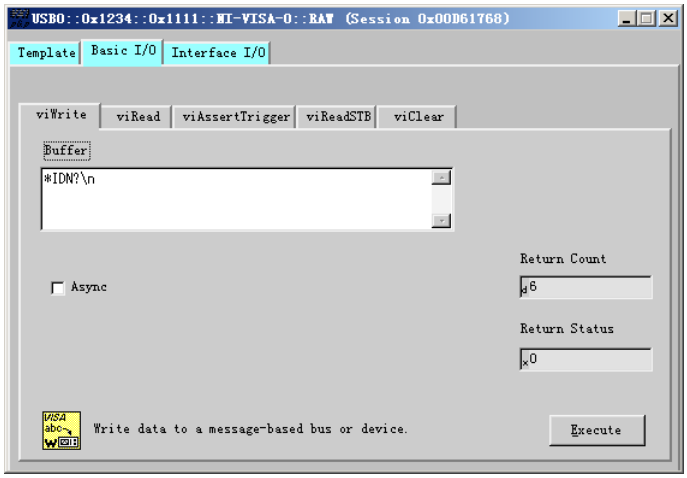

图 10.5　向 USB 设备写数据

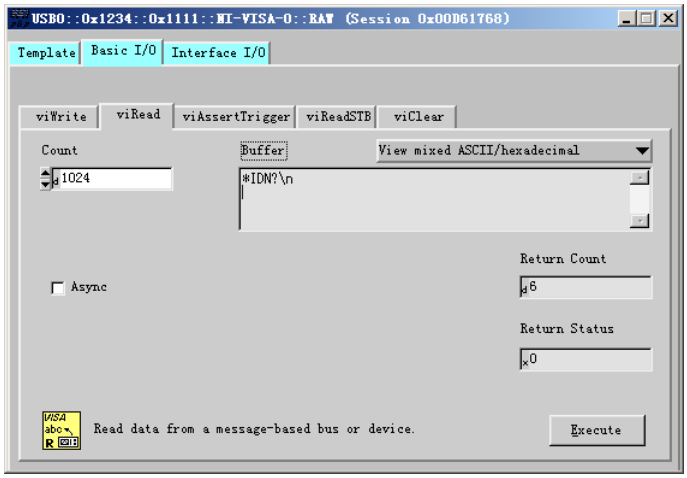

图 10.6　从 USB 设备读数据

（9）在 Interface I/O 中，可以各种标准及自定义控制请求的测试。例如，在 viUsbControlOut 中输入 Buffer 为“B2 01”，bmRequestType 为 0x40，bRequest 为 0xB2。然后单击“Execute”按钮，便可以执行自定义请求 0xB2，如图 10.7 所示。

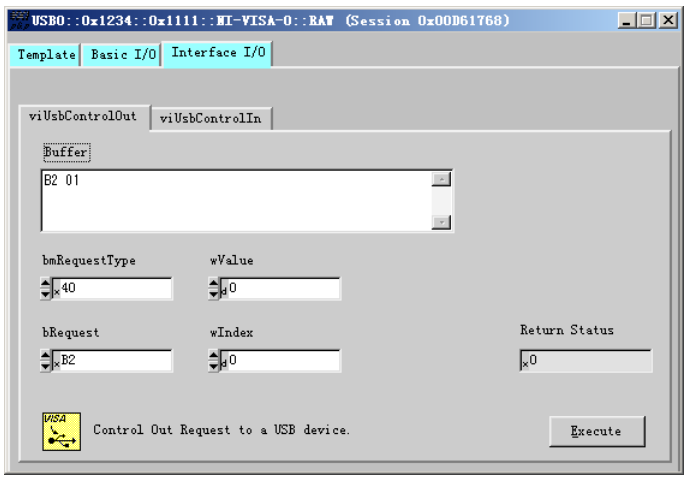

图 10.7　自定义请求 0xB2

（10）在 viUsbControlIn 中，输入 wLength 为 2，bmRequestType 为 0xC0，bRequest 为 0xB3。然后单击“Execute”按钮，便可以执行自定义请求 0xB3，如图 10.8 所示。

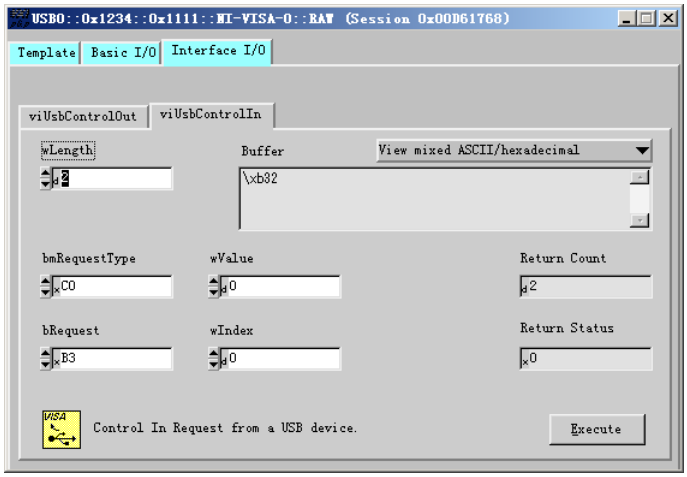

图 10.8　自定义请求 0xB3

10.3.2　VISA 控制函数

当使用 NI-VISA 安装完 USB 设备的驱动程序后，便可以在 LabVIEW 中使用相关的 VISA 控制函数进行读写操作。下面便介绍在 USB 程序设计中用到的 VISA 控制函数。

1．VISA 打开

“VISA 打开”用于打开指定的 VISA 资源名称，如图 10.9 所示。其各个主要连接端口的含义如下：

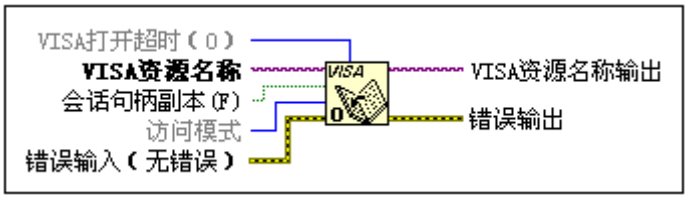

图 10.9　VISA 打开

- “VISA 打开超时”指定“VISA 打开”在返回错误前等待的最大超时值，以毫秒为单位。
- “VISA 资源名称”指定要打开的资源。该控件也可指定会话句柄和类。
- “会话句柄副本”：如会话句柄副本的值为 TRUE，且当前存在一个对资源开放的会话句柄，将为资源打开另一个会话句柄。如会话句柄副本设置为 FALSE，且存在一个对资源开放的会话句柄，将使用打开的会话句柄。VISA 会话句柄是 VISA 使用的唯一逻辑标识符，用于与资源进行通信。VISA 会话句柄由 VISA 资源名称控件保持，用户不能看见该控件。
- “访问模式”指定如何访问设备。如果该值为 0，则不以排它锁定或加载注册信息的方式打开会话句柄；如果该值为 1，则打开会话时获取排它锁定，若无法获得锁定，则关闭该会话并返回错误；如果该值为 4，则通过外部配置工具指定的值配置属性。
- “VISA 资源名称输出”是 VISA 会话句柄打开的资源及其类。

2．VISA 关闭

“VISA 关闭”用于关闭 VISA 资源名称指定的设备会话句柄或事件对象，如图 10.10 所示。打开 VISA 会话句柄并完成操作后，应关闭该会话句柄。“VISA 资源名称”指定要打开的资源。该控件也可指定会话句柄和类。

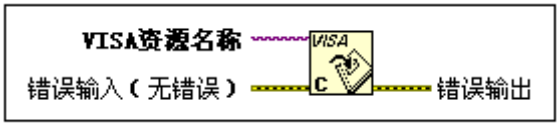

图 10.10　VISA 关闭

3．VISA USB 控制输出

“VISA USB 控制输出”对设备执行 USB 控制管道传输，如图 10.11 所示。该函数采用控制传输设置阶段的数据有效载荷作为参数。如传输要求数据阶段，该函数将发送可选数据缓冲区，即写入缓冲区。其各个主要连接端口的含义如下：

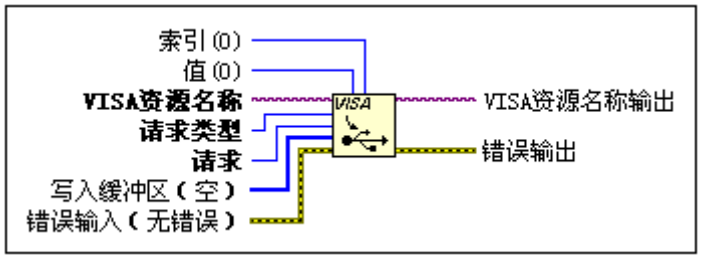

图 10.11　VISA USB 控制输出

- “索引”传递参数至设备。输入的值取决于请求输入的值。索引通常用于请求中以指定一个结束点或一个接口。
- “值”传递参数至设备。输入的值取决于请求输入的值。
- “VISA 资源名称”指定要打开的资源。该控件也可指定会话句柄和类。
- “请求类型”是发送至设备的请求的数值表示。参数是标识特定请求的特性的位映射。指定方向的位必须被设置为 0（主机至设备）。
- “请求”指定特定的请求。可以输入的请求取决于输入请求类型的值。
- “写入缓冲区”包含要写入设备的数据。
- “VISA 资源名称输出”是由 VISA 函数返回的 VISA 资源名称的副本。

4．VISA USB 控制输入

“VISA USB 控制输入”执行来自 USB 设备的 USB 控制管道传输，如图 10.12 所示。该函数采用控制传输设置阶段的数据有效载荷作为参数。如传输要求数据阶段，该函数将读取可选数据缓冲区，即读取缓冲区。其各个主要连接端口的含义如下：

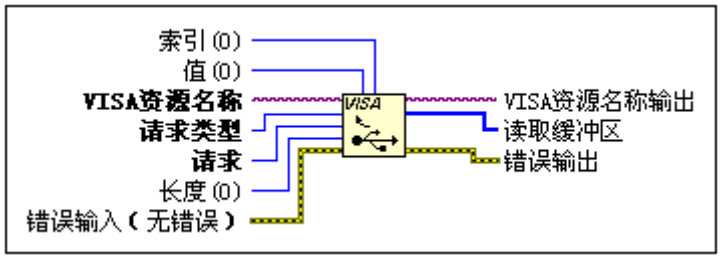

图 10.12　VISA USB 控制输入

- “索引”传递参数至设备。输入的值取决于请求输入的值。索引通常用于请求中以指定一个结束点或一个接口。

- “值”传递参数至设备。输入的值取决于请求输入的值。
- “VISA 资源名称”指定要打开的 USB 资源。
- “请求类型”是发送至设备的请求的数值表示。参数是标识特定请求的特性的位映射。指定方向的位必须被设置为 1（设备至主机）。
- “请求”指定特定的请求。可以输入的请求取决于输入请求类型的值。
- “长度”指定在控制传输的第二个阶段传输数据的长度。方向为设备至主机。
- “VISA 资源名称输出”是由 VISA 函数返回的 VISA 资源名称的副本。
- “读取缓冲区”包含从设备读取的数据。

5．VISA 写入

“VISA 写入”将写入缓冲区的数据写入 VISA 资源名称指定的设备或接口中，如图 10.13 所示。其各个主要连接端口的含义如下：

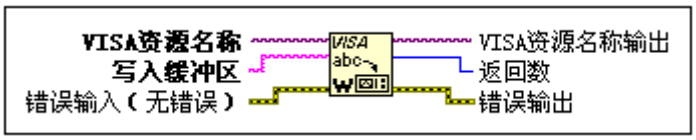

图 10.13　VISA 写入

- “VISA 资源名称”指定要打开的资源。
- “写入缓冲区”包含要写入设备的数据。
- “VISA 资源名称输出”是由 VISA 函数返回的 VISA 资源名称的副本。
- “返回数”包含实际写入的字节数。

6．VISA 读取

“VISA 读取”从 VISA 资源名称所指定的设备或接口中读取指定数量的字节，并将数据返回至读取缓冲区，如图 10.14 所示。其各个主要连接端口的含义如下：

- “VISA 资源名称”指定要打开的资源。该控件也可指定会话句柄和类。
- “字节总数”是要读取的字节数量。
- “VISA 资源名称输出”是由 VISA 函数返回的 VISA 资源名称的副本。
- “读取缓冲区”包含从设备读取的数据。
- “返回数”包含实际读取的字节数。

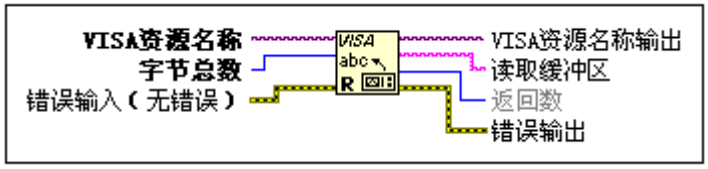

图 10.14　VISA 读取

在 LabVIEW 程序中，使用“VISA 打开”函数打开指定的 USB 设备，然后便可以使用各种 USB 控制函数来实现请求和数据的传输。当操作完成后，使用“VISA 关闭”函数关闭指定的 USB 设备即可。

10.4 小结

本章详细讲解了 USB 设备的上位机程序开发。首先介绍了在 Visual C++ 6.0 平台下的开发方法，主要采用 CY3684 开发包中的 CYIOCTL 控制函数类和 CyAPI 控制函数类。接着，介绍了在 Visual C#平台下的开发方法，主要采用了 SuiteUSB.NET 2.0 开发包中的 CyUSB.dll 提供的各种控制函数类。最后，介绍了在 LabVIEW 平台下的开发方法，主要采用了 VISA 控制函数来实现。USB 设备经常用于上位机中供人机接口及控制。掌握各种开发平台下的 USB 设备开发方法十分重要，读者应该能够熟练运用。