USB Simple Terminal

Problem Statement

I have an existing RS232 device that is connected to the serial port of my PC. My application software accesses the device using the COMx system device in Windows.

I want to connect this existing device to the PC using USB. What hardware do I need to convert USB-to-RS232-to-USB and what changes do I need to make to my application software?

For a more detailed problem statement, and a discussion of how to cost reduce this solution, refer to Chapter 8: Building USB Bridges of USB Design By Example (ISBN 0471370487)

Example Solution

The example solution exists in two sections; the PC Host software and the USB I/O device firmware. The source code for each section is supplied and this note documents the structure of the code so that it may be more easily modified to suit your application.

This bridge solution is implemented as a USB HID (Human Interface Device) so no driver, DLL or other magic operating system software need be written. The PC Host software is implemented as an applications-level program. This solution works well for low-to-medium data rate peripherals – it is not suitable for high bandwidth communications.

The example solution uses an Anchorchips EZ-USB component as a USB-to-RS232-to-USB bridge. Less than 5% of the capability of this device is used: a simpler microcontroller may be substituted (an exercise for the reader) or the remaining 95% could be used to implement features and functions of the “existing RS232 device”.

PC Host application software

The PC Host software was developed using the “Windows Application” template in the Microsoft Visual C++ development toolkit. This template creates the basic framework for an application and custom routines are added to respond to user inputs.

The program needs to support two independent tasks – monitor the keyboard for input and monitor USB for input. Two program threads are required since the Windows system I/O routines that communicate with the USB subsystem are blocking calls (ie they wait for data). A *ListenToUSB* thread to is created after the ‘Serial1’ USB device is detected as being attached to the PC Host. The *OpenUSBdevice* function searches through a system table of the currently attached HID devices and identifies the USB I/O device by its unique product name. A later release of this example software will allow for multiple ‘Serial1’ devices, this V0.9 Beta software only supports a single device.

Characters are collected in a large *Buffer* and the whole screen is repainted for each new character. A “real” simple terminal program would manage special characters such as backspace, delete, TAB and others. This is left as an exercise for the reader and I would request that you send program enhancements to me so that I could redistribute them. A more elegant *WM_PAINT* routine, which just updated the section of window that changed, rather than repaint the whole window, would also be appreciated.

A menu item selects Full or Half-Duplex mode. In half-duplex the keyboard characters are also sent to the display to create a local echo.

The program runs until its window is closed or Exit is chosen.

USB I/O Device

The EZ-USB program is written in six modules just like all of the other “USB Design By Example” examples. All of the enumeration code is reused. The descriptors define a HID with a single byte input report and a single byte output report.

Speed matching and buffering between the USB “side” and the RS232 “side” is implemented by two circular buffers. A *SendBuffer* is filled by USB Ouput Reports and is emptied by transmitting the characters on the RS232 line. A *ReceiveBuffer* is filled via received RS232 characters and emptied via USB Input Reports. Flags are set if either buffer becomes full and the newest characters are discarded (a later version may do something more elegant). All buffer filling/emptying is done under interrupt control.

Integrating this example into your design

Notice that *OpenUSBdevice(“Serial1”)* is called rather than *CreateFile(“COM1”, . .).* A file handle is returned in both cases and this is used in later *ReadFile*, *WriteFile* and *Close* requests. This is a small change to your application program. Yes, it would be nice for this USB device to enumerate as COM5 or COM6 but this is technically very difficult to do and will restrict later expansion of this solution.

Note that the bytecount in the *ReadFile* and *WriteFile* requests MUST match the (Report Sizes + 1) declared in the HID descriptor. This example program supports single byte transfers so that it looks as similar to RS232 as possible. The next example, Serial8, will support 8-byte block transfers and will thus use USB more efficiently.

As always, please send any comments, corrections or questions to me using John@USB-By-Example.com.