USB PC Driver based on HID Class

Features

Full duplex bandwidth communication

Guaranted bandwidth up to 512 Kbit/s in each direction

Windows 98SE, Me, 2000 and XP supported

AT89C513x and AT90USB devices supported

1. Introduction

The USB interface becomes very complex when the user data do not fit with USB standard classes (Mass Storage, Audio, Video...). Specific drivers must then be developed, requiring a significant amount of development time. Atmel has developed a solution to save time and development efforts.

This solution is based on HID class which is supported by all Microsoft O/S from Windows 98SE and later. It is also supported by most other O/S running on PCs (at time of publication).

It ensures a full duplex transfer between the device and the PC (up to 64Kbytes/s). Adding to the data exchange, this application allows the user to upgrade firmware without any hardware setup.

This document gives information on integrating the ATMEL USB HID DLL functions and provides an example of use.

A familiarity with the HID specification (http://www.usb.org/develop-ers/hidpage) is assumed.

USB Microcontrollers

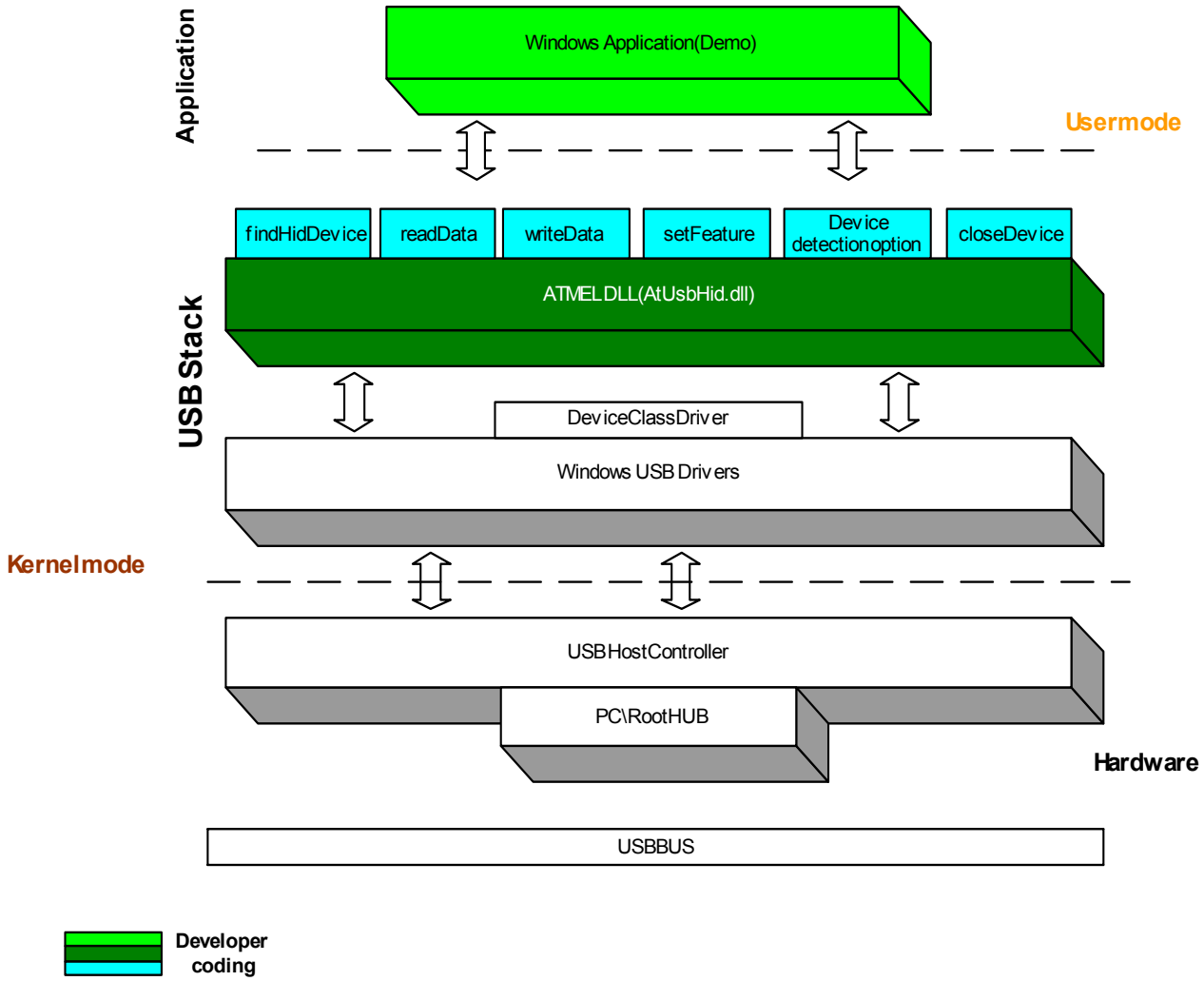
Application Note

7645A–USB–04/06

2. Solution Overview

The DLL provides a full duplex transfer with a USB HID device. The data transfer uses two endpoints (IN and OUT) with interrupt transfer mode and the command transfer uses the endpoint 0 with control transfer mode.

Figure 2-1. Solution Overview



3. DLL Functions Description

To hide the heavy USB driver development, Atmel offers a DLL solution with the following functions:

3.1 findHidDevice

Scans the connected USB devices for a HID one matching the specified Vendor ID and Product ID. Once the device is found, this function open the communication (create the handle).

Input

```
const UINT VendorID: this is the vendor ID
const UINT ProductID: this is the product ID
```

Output

0 if fail

1 if connected successfully

3.2 closeDevice

Closes the USB device and all handles.

3.3 writeData

Sends data to the device using the OUT endpoint. Call this function when data length is smaller than or equal to the report size.

Note: The buffer must not exceed the write buffer capabilities of the connected device.

Input

```
UCHAR* buffer: pointer to the message to be written.
```

Output

0 if fail ,

1 if message send successfully

3.4 readData

Reads data from the device using the endpoint IN.

Call this function when data length is smaller than or equal to the report size.

Input

```
UCHAR* buffer: Pointer to the buffer that will contain the received packet
```

Output

0 if no data are available.

1 if data as been written in UCHAR* buffer.

3.5 setFeature

This function manages the command and control transfer using the endpoint 0.

Input

```
UCHAR type: this parameter can be used by the developer to indicate the
command type to the device.
```

UCHAR direction: this parameter can be used by the developer to indicate the direction of the data which should be sent following this command (IN or OUt direction).

unisigned int length: this parameter indicates the length of the data which should be sent following this command.

The use of these parameters depends of the firmware implementation.

Output

0 if fail to send the comand

1 if the command has been sent successfully.

3.6 Device detection option

The DLL provides three functions which can be used only with Visaul C++ project to detect the plug and the remove of the device. These functions are described below:

3.6.1 hidRegisterDeviceNotification

Registers the device for which a window will receive notifications.

Input

HWND hWnd - Handle to a window.

Output

0 if the function fails, To get extended error information, call GetLastError

1 if the function succeeds

3.6.2 hidUnregisterDeviceNotification

this function closes the specified device notification handle.

Input

HWND hWnd - Handle to a window.

Output

0 if the function fails. To get extended error information, call GetLastError.

1 if the function succeeds.

3.6.3 isMyDeviceNotification

This function must be call to check if this is a device our device which a trigger the call

of OnDeviceChange.

Input

DWORD dwData, the value given by OnDeviceChange 2nd paramerters

Output

1 if this is our device that change it connected status

0 if this is another device

4. DLL Integration

4.1 Load the DLL in Visual C++ Application

The file AtUsbHid.h provides macros that help to load and use functions present in the ATMEL USB HID DLL.

When designing an application using the DLL you need to do the following:

- create a handler to the DLL: **HINSTANCE hLib = NULL;**
- Load the DLL using the function **hLib =LoadLibrary(AT_USB_HID_DLL);**
- Load each DLL function using **loadFuncPointers(hLib)**

Once these steps have been performed without error, DLL and function are loaded in your application and can been called using the macro **DYNCALL(**DllFunction()**);**

When program ends. It is convenient to free DLL from memory using function **FreeLibrary(hLib);**

You must be sure that USB device has been closed before freeing the DLL from memory.

4.2 Using Automatic Device Connection/Disconnection Feature

The DLL provides functions that help developer to detect if device status change.

To do so you have to do the following in your Application.

Register you application to get device change notification using: **DYNCALL(hidRegisterDeviceNotification)((m_hWnd));**

Add the function **ON_WM_DEVICECHANGE()** in your Message Map application

Create a Function called **OnDeviceChange(UINT nEventType, DWORD dwData)** that will be called each time a device status changes.

In the function **OnDeviceChange**, call function **DYNCALL(isMyDeviceNotification(dwData));** that will tell you if this is your device status that changed. (See code demo code in UsbHidDemoCodeDlg.cpp)

When exiting the application it is convenient to unregister your application using the function: **DYNCALL(hidUnregisterDeviceNotification(m_hWnd));**

4.3 Using readData

As data can be sent continuously by the device. It's interesting to read data using a timer base function. This avoid to poll continuously the readData function.

To do so you have to do the following in your application.

Add the function **ON_WM_TIMER()** in your Message Map application

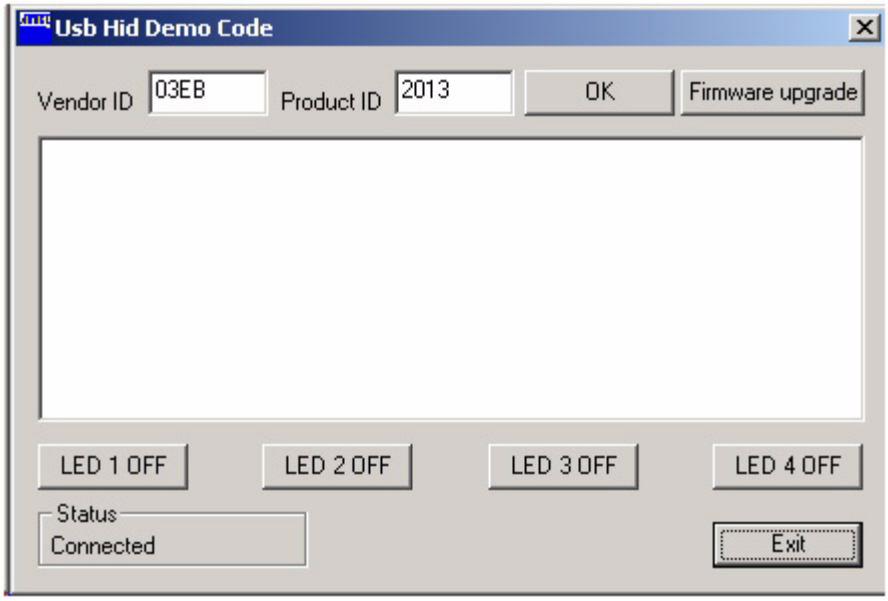
Create a function **OnTimer(UINT nIDEvent)** that will call function **DYNCALL(readData(sbuffer)**

Then you add to set the Timer to a read period using **SetTimer(1,50,0)**; when you device is connected. The function SetTimer set the read period.

Kill the timer using **KillTimer(1)** when your device is disconnected.

5. Example of use

Figure 5-1. Atmel HID Example



This is a Dialog Box application which can be used to set LEDs present on the board, capture button action, set board to firmware upgrade mode, change default PID and VID, detect if a device has been connected or detected.Demo applications have been written in Visual C++ 6.0 and Visual C++ service pack 6. The USB HID DLL can be found with its include file in the directory: AtUsbHid. Please refer to the source code of this example to have more information concerning the use of this DLL. The hardware resources required for this example are the following:

- The AT89STK-05 (the AT989C5130/31 evaluation board) if you use the C51 package
- The STK525 (the AT90USB evaluation board) if you use the AVR package

Please refer to the HID Generic implementation document for further details concerning the firmware package