VOLUME 5
2007

SOUTH-WEST
UNIVERSITY
·NEOFIT RILSKY·
BLAGOEVGRAD, BULGARIA

SCIENTIFIC Research

ISSN 1312-7535

ELECTRONIC ISSUE

A new approach to embedded applications based on PIC microcontrollers use USB interface to communicate with PC

Anton Stoilov
South-West University, Blagoevgrad, Bulgaria

***Abstract**: This paper presented one possibility to development a simple and small component count USB data acquisition system, used Delphi on PC side to communicate. The main core of USB device is PIC18F4550. This device affords an opportunity to switch different sensors and registered their evidences via USB interface on PC. All received data was storages in .txt file, with standard structure. The software support for the data acquisition system device includes the sources of the program on the chip and the Windows application communicated with device as well as the driver. The real device was produced, tested and analyzed. The results are presented in this article.*

***Keywords:** data acquisition, USB, software.*

1. INTRODUCTION

The RS-232 serial interface is no longer a common port found on a personal computer (PC). This is a problem because many embedded applications use the RS-232 interface to communicate with external systems, such as PCs. A solution is to migrate the application to the Universal Serial Bus (USB) interface. There are many different ways to convert an RS-232 interface to USB, each requiring different levels of expertise. The simplest method is to emulate RS-232 over the USB bus. An advantage of this method is the PC application will see the USB connection as an RS-232 COM connection and thus, require no changes to the existing software. Another advantage is this method utilizes a Windows® driver included with Microsoft® Windows® 98SE and later versions, making driver development unnecessary. The objectives of this application note are to explain some background materials required for a better understanding of the serial emulation over USB method and to describe how to migrate an existing application to USB. A device using the implementation discussed in this paper shall be referred to as a USB RS-232 emulated device. All references to the USB specification in this document refer to USB specification revision 2.0. Features in version 1.0 of the RS-232 Emulation firmware:

- A relatively small code footprint of 3 Kbytes for the firmware library
- Data memory usage of approximately 50 bytes (excluding the data buffer)
- Maximum throughput speed of about 80 Kbytes
- Data flow control handled entirely by the USB protocol
- Does not require additional drivers; all necessary files, including the .inf files for Microsoft® Windows® XP and Windows® 2000, are included.

2. OVERVIEW

A Windows application sees a physical RS-232 connection as a COM port and communicates with the attached device using the CreateFile, ReadFile, and WriteFile functions. The UART module on the PICmicro® device provides an embedded device with a hardware interface to this RS-232 connection. When switching to USB, the Windows application can see the USB connection as a virtual COM port via services provided by two Windows drivers, usbser.sys and ccport.sys. In-depth details regarding these Windows drivers are outside the scope of this document. A virtual COM port provides Windows applications with the same program-

ming interface; therefore, modification to the existing application PC software is unnecessary. The areas that do require changes are the embedded hardware and firmware. For hardware, a microcontroller with an on-chip full speed USB peripheral is required to implement this solution. The PIC18F4550 family of microcontrollers is used here as an example. References to the device data sheet in this document apply to the "PIC18F4550 Data Sheet" [1]. Firmware modifications to the existing application code are minimal; only small changes are needed to call the new USB UART functions provided as part of the USB firmware framework written in C. Figure 1 shows an overview of the migration path. Migrating to USB using the RS-232 serial emulation method provides the following advantages:

• It has little or no impact on the PC software application

• It requires minimal changes to the existing application firmware

• It shortens the development cycle time

• It eliminates the need to support and maintain a Windows driver which is a very demanding task

• Finally, the method described here utilizes a clear migration path from many existing PICmicro devices to the PIC18F4550 family of microcontrollers, making the upgrade to USB straightforward

Since the USB protocol already handles the details of low-level communication, the concept of baud rate, parity bit and flow control for the RS-232 standard becomes abstract.

Devices in the PIC18F4550 family incorporate a fully featured Universal Serial Bus communications module that is compliant with the USB Specification Revision 2.0. The module supports both low-speed and full-speed communication for all supported data transfer types. This family of devices offers the advantages of all PIC18 microcontrollers – namely, high computational performance at an economical price – with the addition of high endurance, Enhanced Flash program memory. In addition to these features, the PIC18F4550 family introduces design enhancements that make these microcontrollers a logical choice for many high-performances, power sensitive applications. Devices in the PIC18F4550 family are available in 28-pin and 40/44-pin packages.

3. OVERVIEW OF USB

USB device functionality is structured into a layered framework graphically shown in Fig.1. Each level is associated with a functional level within the device.

The highest layer, other than the device, is the configuration. A device may have multiple configurations. For example, a particular device may have multiple power requirements based on Self-Power Only or Bus Power Only modes. For each configuration, there may be multiple interfaces. Each interface could support a particular mode of that configuration.

The PIC18F4550 device family contains a full-speed and low-speed compatible USB Serial Interface Engine (SIE) that allows fast communication between any USB host and the PIC® microcontroller. The SIE can be interfaced directly to the USB, utilizing the internal transceiver, or it can be connected through an external transceiver. An internal 3.3V regulator is also available to power the internal transceiver in 5V applications. Some special hardware features have been included to improve performance. Dual port memory in the device's data memory space (USB RAM) has been supplied to share direct memory access between the microcontroller core and the SIE. Buffer descriptors are also provided, allowing users to freely program endpoint memory usage within the USB RAM space. A Streaming Parallel Port has been provided to support the uninterrupted transfer of large volumes of data, such as isochronous data, to external memory buffers. Fig. 2 presents a general overview of the USB peripheral and its features.

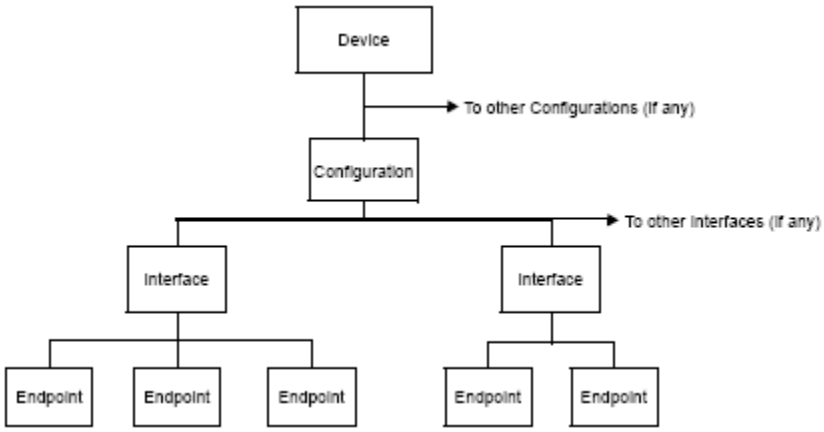

Fig. 1: Usb Layers

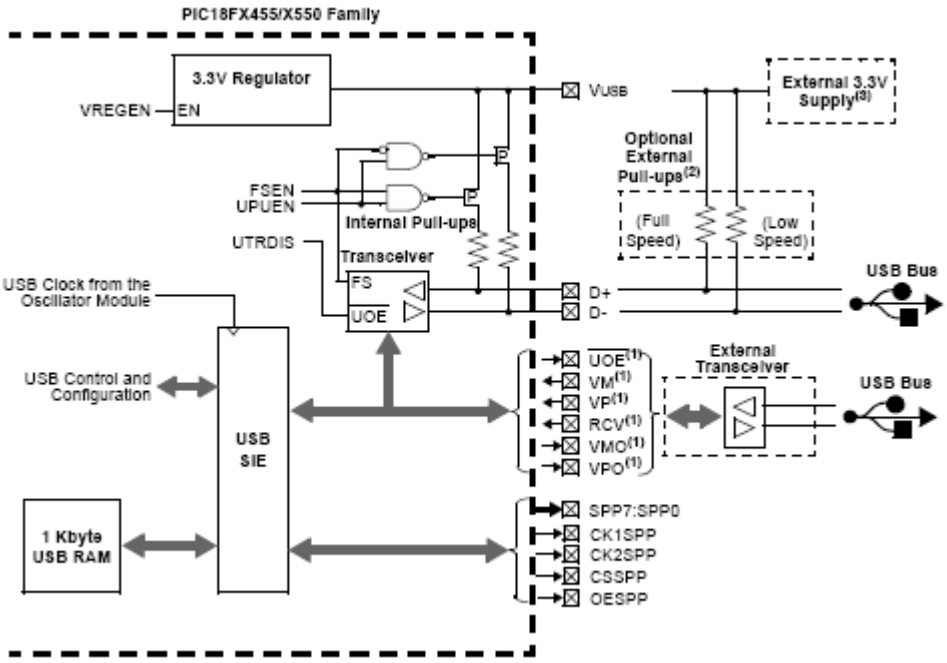

Fig. 2: USB PERIPHERAL AND OPTIONS

The Communication Device Class (CDC) specification defines many communication models, including serial emulation. All references to the CDC specification in this document refer to version 1.1. The Microsoft Windows driver, usbser.sys, conforms to this specification. Therefore, the embedded device must also be designed to conform with this specification in order to utilize this existing Windows driver. In summary, two USB interfaces are required. The first one is the Communication Class interface, using one IN interrupt endpoint. This interface is used for notifying the USB host of the current RS-232 connection status from the USB RS-232 emulated device. The second one is the Data Class interface, using one OUT bulk endpoint and one IN bulk endpoint. This interface is used for transferring raw data bytes that would normally be transferred over the real RS-232 interface. Designers do not have to worry about creating descriptors or writing function handlers for Class-Specific Requests. Descriptors for a USB RS-232 emulated device and all required handlers for Class-Specific

4. SHEMATIC

This USB Data acquisition enable 8 Digital output, 8 Digital input, 8 Analog output. The device no external power required. The below given circuit in fig.3.

Devices in the PIC18F4550 family incorporate a different oscillator and microcontroller clock system than previous PIC18F devices. The addition of the USB module, with its unique requirements for a stable clock source, make it necessary to provide a separate clock source that is compliant with both USB low-speed and full-speed specifications. To accommodate these requirements, this device include a new clock branch to provide a 20 MHz clock for full-speed USB operation.

When connect this USB Data acquisition with PC then windows ask for driver after driver installed(only first time) you will have a new COMx if not connect the COMx not create by Windows.

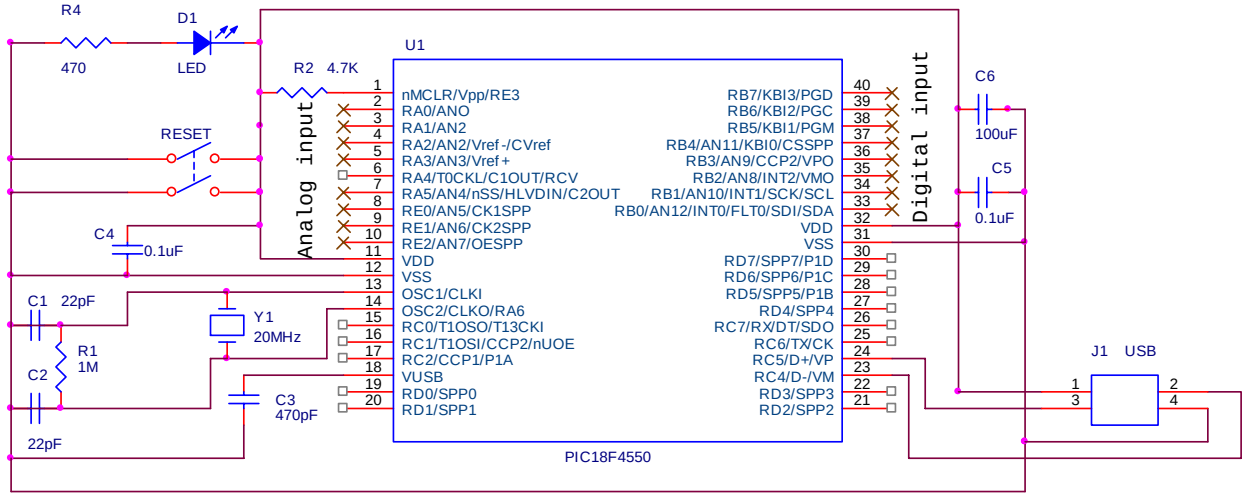

Fig. 3: Schematic of the USB Data acquisition.

Power is available from the Universal Serial Bus. The USB specification defines the bus power requirements. Devices may either be self-powered or bus powered. Self-powered devices draw power from an external source, while bus powered devices use power supplied from the bus. The USB specification limits the power taken from the bus. Each device is ensured 100 mA at approximately 5V (one unit load). Additional power may be requested, up to a maximum of 500 mA. Note that power above one unit load is a request and the host or hub is not obligated to provide the extra current. Thus, a device capable of consuming more than one unit load must be able to maintain a low-power configuration of a one unit load or less, if necessary. The USB specification also defines a Suspend mode. In this situation, current must be limited to 500 µA, averaged over 1 second. A device must enter a Suspend state after 3 ms of inactivity (i.e., no SOF tokens for 3 ms). A device entering Suspend mode must drop current consumption within 10 ms after Suspend. Likewise, when signaling a wake-up, the device must signal a wake-up within 10 ms of drawing current above the Suspend limit.

5. WINDOWS APPLICATION AND COMUNICATION

While the provided USB UART functions greatly simplify the integration of USB into an application, there are still some considerations to keep in mind when developing or modifying application code.

5.1. *Code design*

The Microchip USB firmware is a cooperative multitasking environment. There should not be any blocking functions in the user code. As USB tasks are polled and serviced in the main program loop, any blocking functions that are dependent on the state of the USB may cause a deadlock. Use a state machine in place of a blocking function.

mUSBUSARTTxRom and mUSBUSARTTxRam expect a data pointer of type rom byte* and byte*, respectively. Type casting may be necessary. 3. while(!mUSBUSARTIsTxTrfReady()); is a blocking function. Do not use it.

putrsUSBUSART, putsUSBUSART, mUSBUSARTTxRom and mUSBUSARTTxRam are not blocking functions. They do not send out data to the USB host immediately, nor wait for the transmission to complete. All they do is set up the necessary Special Function Registers and state machine for the transfer operation. The routine that actually services the transfer of data to the host is CDCTxService(). It keeps track of the state machine and breaks up long strings of data into multiple USB data packets. It is called once per main program loop in the USBTasks() service routine. Because of this, back-to back function calls will not work; each new call will override the pending transaction.

A correct set of descriptors for the CDC class must be used. Refer to the reference design project for an example.

The endpoint size for the data pipes are defined by CDC_BULK_OUT_EP_SIZE and CDC_BULK_IN_EP_SIZE, located in the header file usbcfg.h. Since these endpoints are of type bulk, the maximum endpoint size described must either be 8, 16, 32 or 64 bytes.

The type byte is defined as an unsigned char in the header file typedefs.h.

Always check whether the firmware driver is ready to send more data out to the USB host by calling mUSBUSARTIsTxTrfReady().

5.2. *Setting up the code project*

1. Insert #include "system\usb\usb.h" in each file that uses the CDC functions.
2. USB_USE_CDC should be defined in the file usbcfg.h when using the CDC functions.
3. The source and header files, cdc.c and cdc.h, should be added to the project source files. They can be found in the directory “system\usb\class\cdc\”.

5.3. *USB Vendor ID (VID) and Product ID (PID)*

The VID and PID are important because they are used by the Windows operating system to differentiate USB devices and to determine which device driver is to be used. The VID is a 16-bit number assigned by the USB Implementers Forum (USB-IF). It must be obtained by each manufacturer that wants to market and sell USB products. The VID can be purchased directly from USB-IF. More detailed information can be found at: http://www.usb.org/developers/vendor. Each VID comes with 65,536 different PIDs which is also a 16-bit number. In the Microchip USB firmware framework, the VID and PID are located in the file usbdsc.c. Both values can be modified to match different product VID and PID numbers.

5.4. *Drivers for Microsoft Windows® 2000 and Windows® XP*

Microsoft Windows does not have a standard .inf file for the CDC driver. The drivers are, however, part of the Windows installation. The only thing necessary to do is to provide an .inf file when a CDC device is first connected to a Windows system.

Example .inf files are provided with the CDC RS-232 Emulation Reference Project and are located in the source code directory <Install>\fw\CDC\inf. Before using them, they must be modified to reflect the application's specific VID and PID. This is in addition to any changes to usbdsc.c that have already been made and must match those values. The VID and PID are located in the string "USB\VID_xxxx&PIDyyyy", where "xxxx" is the hexadecimal VID and "yyyy" is the hexadecimal PID. The string is generally part of one of the lines under the heading "[DeviceList]". If desired, users may also modify the variable definitions under the heading "[Strings]". This changes the device identification text seen by the user in the device manager.

6. SUMMARY

The RS-232 serial emulation provides an easy migration path for applications moving from UART to USB. On the PC side, it requires minimal software modification. On the embedded device side, the PIC18F4550 family of microcontrollers provides a simple hardware upgrade path from the PIC16C745/765 and PIC18FXX2 families of devices. Library firmware with user-friendly APIs are also included for convenience. Tutorial exercises are available as part of the CDC RS-232 Emulation Reference

7. REFERENCES

[1] "PIC18F2455/2550/4455/4550 Data Sheet" (DS39632), Microchip Technology Inc., 2004.

[2] "USB Specification Revision 2.0", USB Implementers Forum Inc., 2000.

[3] "USB Class Definitions for Communication Devices Version 1.1", USB Implementers Forum Inc., 1999.