DEVELOPER FRIENDLY USB:
A detailed look at how easily a USB-based device can be developed

By Don L. Powrie

As the popularity of USB continues to increase, so does the desire and need for easier ways to develop USB-based products. Lots of integrated circuit manufacturers are now offering USB serial engines and integrated USB/Microcontroller chips. The use of these devices assumes you will take on the burden of all required software development.

The development of most new USB-based products begins with the laborious and frustrating task of writing a device driver. This task can easily consume the bulk of a developer's free time (nights and weekends that is) trying to stay on schedule. This task can be simplified somewhat by using the Human Interface Device (HID) class, or it can be eliminated altogether by using FTDI's royalty-free drivers. The only catch with using FTDI's drivers is that they only work with FTDI's chips. FTDI offers two versions of their USB chips (USB to UART and USB to parallel FIFO) as well as two versions of their drivers (Virtual Com Port and DLL).

This article will describe a complete USB-based product design--start to finish--for acquiring temperature data from two digital temperature sensors. The USB-UART version of the FTDI device (FT232BM) will be used as well as their Virtual Com Port drivers.

MICROCONTROLLER

As mentioned above, FTDI (www.ftdichip.com) makes both a parallel and a serial version of their USB chip. Connection to the parallel version (FT245BM) requires 8 data lines and 4 handshaking lines to properly implement the interface. What if you only have 2 free port pins on your microcontroller? The serial version of their chip presents an industry standard RS232 interface (RX, TX, CTS, RTS, etc.) for easy connection to a legacy RS232 device. All that is typically needed to connect a legacy RS-232 device to the FT232BM is a garden-variety receiver/driver IC (MAX-232, HIN232, etc.) and a DB9 connector. Using the

FT232BM USB-UART chip instead of the parallel version frees up 10 micro port pins for use in your project.

For this project, we will interface the FT232BM directly to an 8-pin, PIC 12F629 Flash-based microcontroller. Two of the micro's pins connect to a 20MHz crystal, two are used for power and ground, two for communicating with a host PC (TX and RX), and two for talking to a pair of Dallas Semiconductor DS18B20 digital temperature sensors. Refer to Figure 1 for the schematic.

<Refer to Schematic on DLP-Temp product page>

Figure 1.

The micro only has 1K of ROM and, surprisingly, the firmware for this project only required 54% of the available ROM thanks to PCM, a highly-optimized C compiler from CCS (www.ccsinfo.com). Since the micro is Flash based and a programming header is included in the design, the firmware can be easily updated (or replaced by your own code) without having to remove the micro from the board. And that's significant for this design since the micro we've selected is in an SOIC8 surface-mount (i.e. no socket) package.

All that is needed to interface to each DS18B20 is a single port pin and a pull-up resistor. Neither the micro nor the DS18S20 ever drive the port pin high. The only time it is high is when neither is driving it low. An open collector (or in this case, open drain) port pin on the micro would be perfectly suited for this application. However, since this design doesn't have one available, the software will have to pick up the slack. The functionality of an open drain output can be easily simulated by toggling the port pin between input and output-low. When the port pin is configured as input, the pull-up resistor pulls the data line high.

The design presented in this article shows how to interface two temperature sensors to the USB port. It could have just as easily demonstrated how to interface two switches, two outputs (for driving relays, LED's, etc.) or a mix of the two. Also, a larger micro like the 16F84A could have been used for more port pins.

Programming the 12F629 Flash-based micro is accomplished via a 5-pin header that consists of power and ground, clock, data, and a programming voltage of about 12.5 volts. For this project, I used the EPIC Plus Programmer (www.melabs.com). You will have to make your own programming cable using both 5-pin and 10-pin headers in order to make the connection to the programmer. Keep in mind that Sensor 2 must be disconnected from the board during firmware upload so that it does not interfere with the programming process.

VIRTUAL COM PORT DRIVERS

The Virtual Com Port (VCP) drivers make your USB-based product appear to your Windows application as though it is connected to an RS-232 port. Your application simply opens a com port and sends data as though it was going to the DB9 connector on the back of the PC. If your application opens a com port that is assigned to the VCP drivers (via the Device Manager window), then the data bytes sent are redirected to the USB scheduler and then out the USB port. Any data returned to the host via USB simply appears in the buffer that was created when your application opened the RS-232 port.

Some things you won't have to worry about are USB related details like endpoints, token packets, data packets, handshake packets, ACK's, NACK's… the list goes on and on. The VCP drivers do a splendid job of hiding these details from the user. The desired baud rate, number of data bits, stop bits, and parity selection for the data sent from the FT232BM USB chip to the target electronics can be set by the host application. Setting these parameters in the host application has no effect on transmitted USB data packets.

The maximum data rate that can be expected using the FT232BM is somewhere in the neighborhood of 1-3 megabaud (depending on how your circuit is configured), and all standard baud rates are supported. Applications that require data rates approaching 8 megabits per second fall more under the purview of the FT245BM and DLL drivers.

FIRMWARE

The complete microcontroller C source code listing for this project can be downloaded from www.dlpdesign.com/usb/temp.html. The main() function is shown in Figure 2. As you can see, in this implementation there are only two commands for communicating with the microcontroller. The first is a "ping" command for detecting the presence of the board and microcontroller. This command can be used by the host application to help locate the COM port to which the micro is connected. The second command calls a function that reads the temperature data from both of the temperature sensors, sends 18 bytes of temperature data back to the host, and calls another function [start_convert()] that starts the next temperature conversion. Note that the start_convert() function is called at power up so that temperature data is ready for read the first time the "R" command is issued.

```
void main()
{
	int8 byte_read;

	start_convert();//start conversions on both sensors
	while(1)//continuous loop
	{
		while( !kbhit() ) //while nothing received
			RESTART_WDT(); //pet the dog
		byte_read = getc(); //read one byte from host via USB

		if(byte_read == 'P') //if byte_read is 'P' then send response 'Q'
			putc('Q');

		//don't call this function more than once per second…
		if(byte_read == 'R') //if byte_read is 'R' then read
			read_convert(); //both devices and start new conversion

		byte_read=0;
	}
}//end of main()
```

Figure 2. Source code for main()

The code in main() will wait, continuously resetting the watchdog timer, until Port Pin GP3 is pulled low by the USB chip indicating the start of an incoming serial character. The getc() function is then called to receive the character. Error correction (crc, checksum, etc.) could be added if desired to make the interface more bulletproof.

The DS18B20 temperature sensors take up to 750 milliseconds to perform a full 12-bit temperature conversion and write the data to their internal scratch pad memory. For this reason, there is no reason for the host application to issue the "R" command more than about once per second. Additional firmware could be added to the micro to allow for 9-bit temperature conversions so that much faster temperature measurements (less than 95 milliseconds) can be performed in the DS18B20. Keep in mind that faster readings come at a price. As the speed goes up, the accuracy goes down. (Refer to the DS18B20 datasheet for details.)

WINDOWS SOFTWARE

The source code for simple Visual C++ and Visual Basic programs that demonstrate how to read the temperature data from the 12F629 micro can be downloaded for free from the dlpdesign.com website. The code will request the temperature data once per second, convert the 9 bytes of data returned from each sensor to a floating point value, and display those values. Figure 3 shows a screen shot of this program:

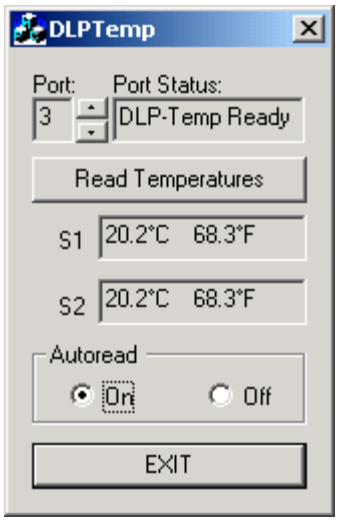

Figure 3. Visual C++ and Visual Basic Demonstration Program

Also available is a Windows application that will read the data and both chart the data graphically and log the data to the hard drive. This program can be purchased for a shareware level fee of $20 ($13 with another purchase). Figure 4 shows a screen shot of this software:

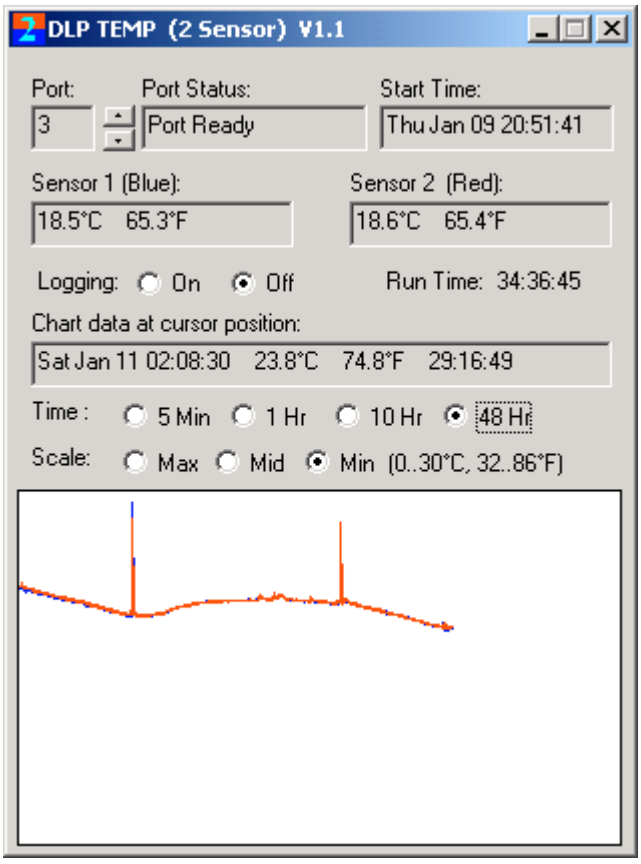

Figure 4.

Figure 5 – DLP-TEMP1 ($35 from dlpdesign.com)

PCB

Figure 5 shows the printed circuit board for this project and just how small a complete USB product can be made using the USB chips from FTDI. The bottom layer of this 2-layer board is mostly a ground plane to help keep the noise and EMI to a minimum. The board comes with one sensor. The sensor is not soldered to the board so that the user has the option of locating the sensor at a distance from the board using Category 5 cable. This design has been successfully tested with both sensors located 200 feet (Figure 6) away from the board using Category 5 cable.

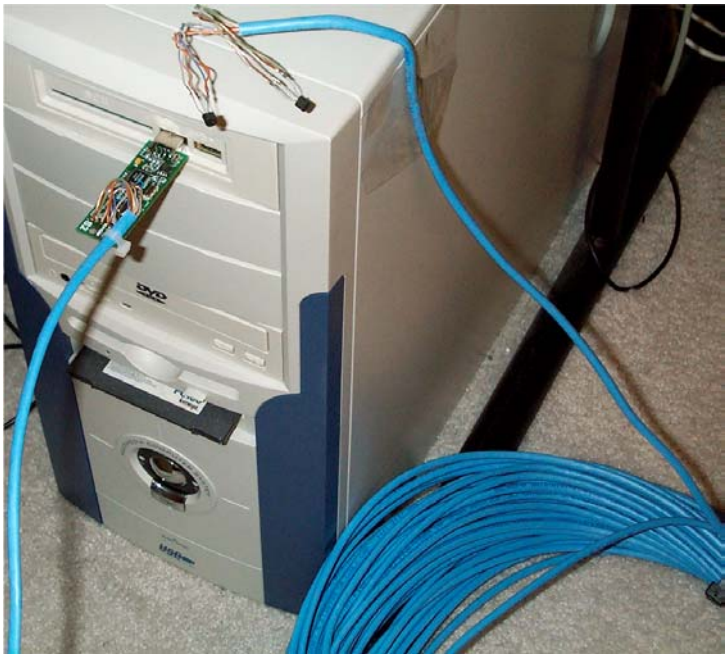

Figure 6 – Tested with 200 feet of Cat 5 cable

Two pairs of wires in the Cat 5 cable are required for the connection; one pair for power and ground, and the other pair for data and ground. If using both sensors, you can either connect them to the board using one cable for both sensors (since the cable has 4 pairs), or 2 cables if the sensors are to be placed in different locations.

Since this design incorporates a male Type A USB plug, it can be directly plugged into the PC's USB port. If you want to bring the board up to your workbench while still communicating with the board, then a USB extension cable can be used.

POWER CONSIDERATIONS

The 5 volts from the USB port is used to power both the circuitry on the PCB as well as the two temperature sensors. This does not mean to imply that the USB port can supply unlimited current to your external circuitry. The maximum current that can be drawn from the port is 500 milliamps. This assumes that the board has been configured as a high-powered device. This also assumes that the board is connected to a port on the PC or powered hub. If connected to a self-powered hub, then the maximum power available for use is reduced to a total of just 100 milliamps.

When the board is first connected to the host PC, the maximum current that can be drawn is 100 milliamps. After enumeration is complete the board can draw the full 500 milliamps. To prevent drawing excessive current before enumeration, this design includes a MOSFET transistor to switch the power

going to the micro and sensors. The switch is controlled by the PWREN# signal from the FT232BM.

CONCLUSION

While FTDI's drivers and chips do relieve the developer of having to learn 95% of what can be learned about USB, I still recommend getting a good book on the topic like Jan Axelson's ***USB Complete Second Edition*** (www.lvr.com) if for no other reason than to become familiar with the complexity of USB and to develop realistic expectations of USB's capabilities. For instance, USB Specification 1.1 proclaims a data rate of 12 megabits per second. In reality, the actual data throughput for a full speed device in bulk transfer mode is closer to a maximum of 8 megabits per second once you add all of the overhead involved in making USB usable and reliable.

If your project calls for higher data throughput or if you would prefer working with a parallel interface and have the required number of micro port pins available, additional reading on using the FT245BM is available on-line at dlpdesign.com/pub.shtml.

USB-based product development really is within reach for those of us (myself included) who don't have the ability to develop device drivers. While I'm almost always intrigued by the challenge of learning something new, the ever-approaching deadline always seems to win out over my desire to take on a new grand venture. So if you want to learn how to write device drivers (a noble venture indeed, and one that might well increase your salary once mastered!), then pick up a book on the topic of WDM device drivers and start reading. If you want to get your new USB-based product to market on time, then have a look at FTDI's chips. I think you'll be glad you did!