USB—MICROCONTROLLERS FOR THE MASSES

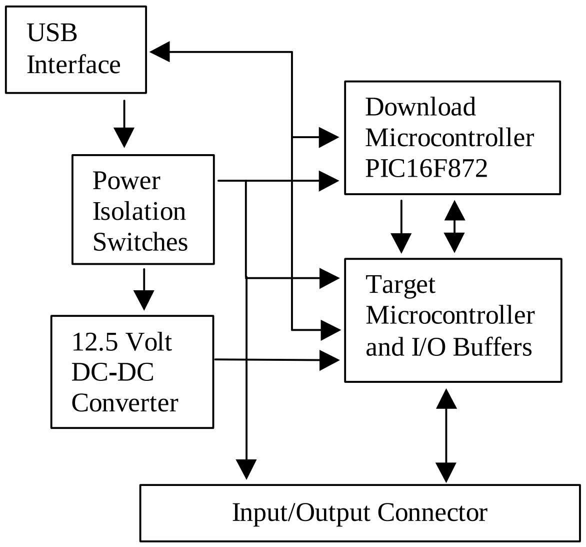
By Don L. Powrie

Microcontroller, Flash programmer, and high speed USB-to-PC interface…all in one tidy little package.

Introduction

Using a microcontroller to interface an electronic device to a PC typically requires the acquisition and development of three things: the microcontroller and its support circuitry (target electronics), a compiler or assembler to create the firmware for the micro-controller, and a device programmer for downloading the firmware into the Flash (ROM) of the target microcontroller. Once you have obtained all of these components and start developing the firmware for your project, you can expect to spend a significant amount of time on the "download" process. This process consists of removing the microcontroller from the target electronics and erasing it either by bulk erasing (Flash device) or placing it under an ultraviolet light source for 10-20 minutes (EPROM device). Then the device must be loaded into the device programmer so that the latest version of your firmware can be downloaded. Once the download has been verified, the device must be returned to the target electronics so that you can continue developing the project. If the project is complex and development time lengthy, this download process may need to be repeated dozens or even hundreds of times.

Wouldn't it be nice if you didn't have to remove the microcontroller from the target electronics, thereby risking hardware and ESD damage, every time you want to download new firmware? This article will detail a design that accomplishes just that. This design


USB Interface
Download Microcontroller PIC16F872
Power Isolation Switches
Target Microcontroller and I/O Buffers
12.5 Volt DC-DC Converter
Input/Output Connector

contains a USB interface for communicating with a host PC, a target microcontroller, and an additional "download" microcontroller that does nothing but handle firmware downloading. The download is performed via a USB interface that can also be used in your project by the target microcontroller for communicating with the host PC.

Free C Compiler

The Microchip PIC 16F84A was chosen as one of two target processors for this project due to the availability of a free, optimized C compiler from Hi-Tech. The free version is limited by the inavailability of printf() support for longs or floats, and only supports a few devices such as the 16F84A. Hi-Tech's standard PIC C compiler must be used for larger processors like the 16F877.

Various other companies offer both C and Basic compilers that work well with Microchip microcontrollers and range in price from under $100 to several thousand dollars. To access a few of them, follow the links shown below:

HI-TECH Software LLC	http://www.htsoft.com/
Custom Computer Services, Inc	http://www.ccsinfo.com/
MicroEngineering Labs, Inc.	http://microengineeringlabs.com/
Grich RC Incorporated	http://members.aol.com/piccompile/
ByteCraft	http://www.bytecraft.com/
IAR	http://www.iar.com/
B Knudsen Data	http://www.bknd.com/

Free Assembler

Microchip offers as a free download their MPASM assembler. MPASM is a full-featured universal macro assembler that will produce HEX files for any PIC microcontroller. This, of course, requires that you know how to program in assembly language. While programs created in assembly language are more compact and, in some cases, execute faster, the learning curve can be considerably longer than learning a high-level language such as C. With careful coding, some C compilers can approach the compact size and execution speed of firmware created with assembly language.

Download Microcontroller

For this project, I utilized a Microchip PIC16F872 for the download microcontroller. The target microcontroller can be either a 16F84A or a 16F877. The schematic (Figure 1) details a design that supports both microcontrollers. Figure 2 illustrates my prototype of the board that was used to develop all of the software. This prototype shows the 16F84A as a surface mount device. The finished product will have the target processors socketed.

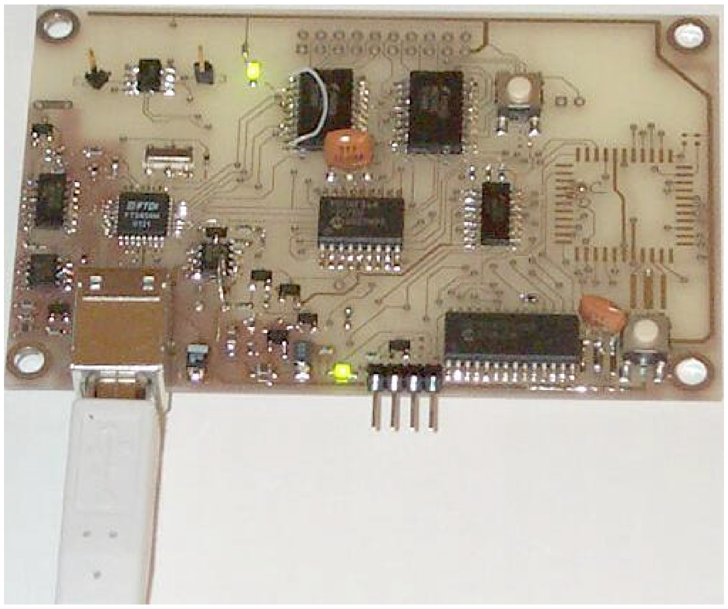


Fig. 2

The download microcontroller serves two purposes: the reset function and the firmware download process. When the reset button is pressed, the target micro will be reset and held in the reset state by the download processor for approximately 5 seconds while flashing the LED. During the 5 seconds, if the host attempts to communicate and initiate a download, then the LED will stay on, and the target microcontroller will remain in the firmware download mode until released. If the host does not attempt to communicate within 5 seconds, then the target micro is released from reset and begins executing its firmware.

Target Microcontroller

When the download microcontroller is activated by pressing the reset button, it starts by taking the MCLR line low (reset) on the target micro. If a firmware download is then initiated by the host, port pins RB6 and RB7 on the target micro are held low, and 12-14 volts are applied to the MCLR line with a fast enough rise time such that the target micro does not begin to increment the program counter. If the rise time is too slow and the program counter in the target microcontroller is able to increment by one, then the firmware will not download into Flash location zero and the microcontroller will not run correctly.

Once the MCLR line reaches 12 volts, all port pins on the target micro are configured as inputs (high impedance), the program counter is pointing to address zero, and the device is ready for firmware download. The download micro communicates with the target micro over a serial interface using only a clock and a bi-directional data line. The command set in Table 1 is all that is required to read and write the Flash memory in the target microcontroller. Each command is a 6-bit command, and is written to the target by toggling the clock line 6 times while writing the command LSB first in the data line.

COMMAND MAPPING FOR PIC16F84A/PIC16F877

Command Mapping	(MSB … LSB)
Load Configuration	(0 0 0 0 0 0) + data
Load Data for Program Memory	(0 0 0 0 1 0) + data
Read Data from Program Memory 4	(0 0 0 1 0 0) + data
Increment Address	(0 0 0 1 1 0)
Begin Erase Programming Cycle	(0 0 1 0 0 0)
Begin Programming Only Cycle	(0 1 1 0 0 0)
Load Data for Data Memory	(0 0 0 0 1 1) + data
Read Data from Data Memory	(0 0 0 1 0 1) + data
Bulk Erase Program Memory	(0 0 1 0 0 1)
Bulk Erase Data Memory	(0 0 1 0 1 1)

Table 1

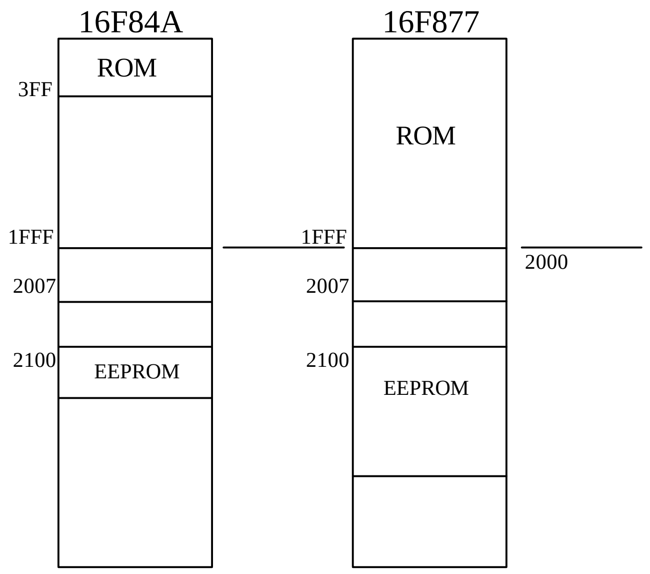
Data is written and read 16 bits at a time in the format of SDDDDDDDDDDDDDDS where the "S" character represents the start and stop bits and "D" is the data. For timing requirements, refer to the Microchip application note EEPROM Memory Programming Specification (DS39025E).

To write HEX program data to the Flash memory, the Load Data for Program Memory command is issued followed by 14 bits of data and then the Begin Programming command. After a 10-millisecond pause, the Increment Address command is called, and the process is repeated until done. Locations in the microcontroller's memory that are not written are left in the all 1's (0x3FFF) state.

To write configuration data to the configuration register, the Load Configuration command is issued. This command sets the program counter to address 0x2000. Since the configuration register resides at address 0x2007, the Increment Address command must be called 7 times. Finally, the Load Data for Program Memory and Begin Programming commands can be called. Ten milliseconds later, the data is stored and is ready for use.

Memory Map

The internal EEPROM memory resides at address 0x2100 and can also be programmed at download time. EEPROM data is only 8 bits wide, but writing the data still requires that 16 bits be clocked in even though only the least significant 8 bits are actually stored.



Once in Load Configuration mode, the program counter will wrap around to 0x2000 if incremented continuously. The only way to get the program counter back to address zero is to reset the device. After a reset, the program counter will wrap back to zero once it reaches 0x1FFF.

HEX File Format

Each line of an Intel HEX data file is structured as follows:

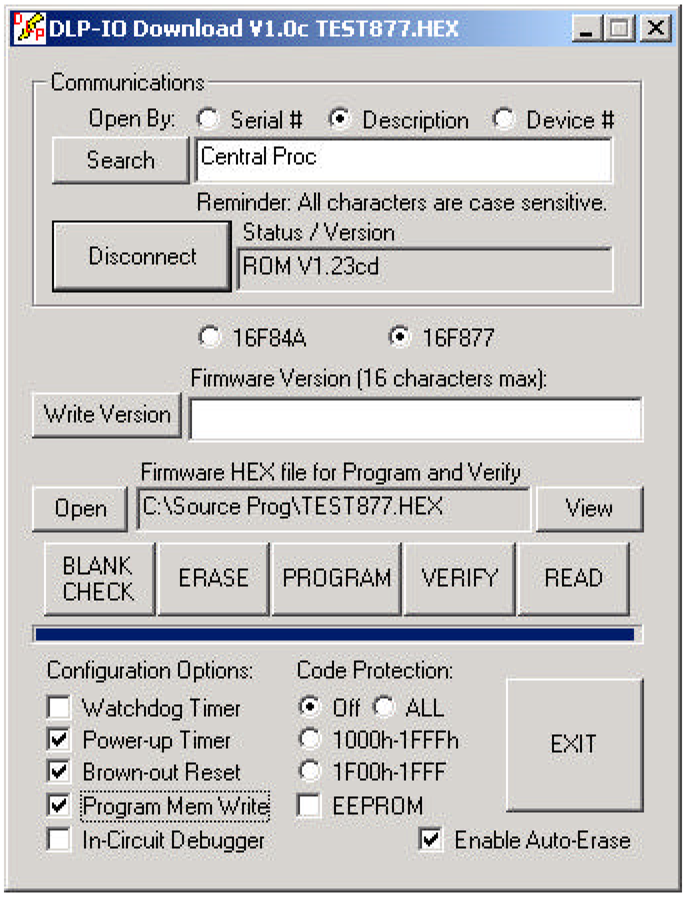
:nnaaaattdddddddddddddddddddddddddddddddd…cc

":" indicates the start of a data record, "nn" indicates the number of bytes in the record, "aaaa" is the load address of the record, "tt" is the record type, "d" is the actual data, and "cc" is the checksum. All values are in ASCII/HEX. The checksum is calculated by performing 8-bit additions of every byte between nn and the end of data. The two's complement is then taken of this sum to get the checksum.

The HEX file stores the actual firmware that is to be written into the Flash of the target microcontroller. A HEX file can also contain configuration data and EEPROM data. The Flash programming system is responsible for setting the program counter in the target microcontroller for the appropriate location in memory that should be written.

Windows Download Software

A 32-bit Windows application (Figure 3) that performs the download process with the 16F872 can be downloaded for free from my website (dlpdesign.com) and supports both the 16F84A and 16F877 microcontrollers. The application uses the DLL version of FTDI device drivers that are also available for download.


DLP-IO Download V1.0c TEST877.HEX

Communications

Open By: ( ) Serial # [✓ Description] ( ) Device #

Central Proc

Reminder: All characters are case sensitive.

Status / Version: ROM V1.23cd

( ) 16F84A [✓ 16F877]

Firmware Version (16 characters max):

Firmware HEX file for Program and Verify: C:\Source Prog\TEST877.HEX

Configuration Options:
- [ ] Watchdog Timer
- [✓] Power-up Timer
- [✓] Brown-out Reset
- [✓] Program Mem Write
- [ ] In-Circuit Debugger

Code Protection:
- [✓ Off] ( ) ALL
- ( ) 1000h-1FFFh
- ( ) 1F00h-1FFF
- [ ] EEPROM

[✓] Enable Auto-Erase

Fig. 3

The DLL version of FTDI's drivers has an "OpenEx" function that allows multiple DLP-IOx devices to be connected to the same PC in a well-organized manner. This function will open the named device and return a handle that will be used for subsequent accesses. The device name can be its serial number or a device description; both are text strings that are programmed into an EEPROM that is directly connected to the FT8U245AM. The above-mentioned Windows download software has a search function that can quickly locate and present a list of all FTDI devices connected to the PC such that a specific target can be selected for firmware download. A third method, open by device number, can be used for new boards that have not had a serial number or description string written to the non-volatile RAM.

Download options include enabling or disabling the Watchdog Timer, Power-up Timer, Brown-out Reset, In-Circuit Debugger and Code Protection modes. (Refer to the datasheet for each device for a full description of each download option.) If a PIC Flash device is code protected, the read function will return all 1's for the data read, and a standard bulk erase will not work correctly. The only way to erase a device that was previously programmed with code protection turned on is by issuing commands 1 and 7 while in Load Configuration mode. The download application mentioned above will perform both a standard bulk erase and a code-protected bulk erase when the Erase function is selected. This allows the target processor to be code protected such that the firmware cannot be read but will still allow for future downloads to occur.

A firmware version string of 16 characters (maximum) can be written to the EEPROM memory in the download microcontroller. This string is read automatically every time the download process is initiated, and can be written any time the download application is running and connected to the target. EEPROM memory in the download microcontroller was used for storage of the version string instead of the target microcontroller's EEPROM so that all of the EEPROM memory in the target would remain available for project use.

USB Interface

The USB interface is implemented with FTDI's FT8U245AM (see the June, 2001 issue of *Nuts & Volts* for more information), which connects to the target microcontroller via its 8-bit data bus and 4 handshaking lines. The USB interface also connects to the download microcontroller via the 8-bit data bus, although only 4 bits of the bus are actually used for communication.

After firmware download, the USB port can be used by the target microcontroller to communicate with the host PC. The actual USB data rate is limited by the speed of the target microcontroller. In the case of a PIC microcontroller running at 20MHz, you can expect maximum data rates of approximately 2 megabits per second.

The most noteworthy aspect of the USB interface is the fact that no Windows driver need be developed. The latest version of the drivers can be downloaded from either FTDI or DLP Design.

Two versions of the driver are available from FTDI: Virtual COM Port (VCP) and DLL. Since the firmware download program uses the DLL version of the drivers, if your project uses the VCP version of the drivers, then you will have to uninstall the VCP version of the drivers and reinstall the DLL version. (This process is quite simple, and only takes a few seconds to perform.)

The VCP drivers are the easier of the two to use. Your application software running on the host PC simply has to open, read from, and write to the standard RS-232 ports. The VCP drivers intercept the serial data, reroute it to the Windows scheduler, and then send it out the USB port to the target electronics. The DLL version of the drivers requires the application to open and read the DLL at runtime. Example code for several programming languages (Visual C++, Visual Basic, etc.) that illustrate the use of both driver types can be downloaded from either ftdichip.com or dlpdesign.com.

Inputs and Outputs

As stated earlier, the design presented here contains two different target processors. The first processor chosen was the 16F84A because of the availability of a free C compiler from HiTech. The 16F84A will hold 1K of ROM, 68 bytes of 8-bit RAM, and 64 bytes of 8-bit EEPROM. The second was the 16F877, which was the largest 14-bit Flash device available from Microchip when this project was under development. The 16F877 will hold 8K of ROM, 368 bytes of 8-bit RAM, and 256 bytes of 8-bit EEPROM.

The DLP-IO16 uses the 16F84A and provides 8 buffered digital inputs and 8 latched digital outputs. The DLP-IO16 is limited in the number of inputs and outputs available to the target electronics primarily due to the 12 port pins that are required to connect to the USB interface. The DLP-IO26 provides 8 latched digital outputs and 18 bi-directional, general-purpose digital I/O lines, 8 of which can be configured as 10-bit analog-to-digital converter inputs for measuring voltages in the range of 0-5 volts. Adding an external resistor voltage divider can extend the voltage input range. These boards can be purchased from dlpdesign.com in either configuration.

Theory of Operation

The schematic for this project shows a 40-pin connector at JP3. The DLP-IO16 only uses the first 26 pins, while the DLP-IO26 uses all 40 pins. Both boards are shipped without a header installed since connection to your project can be made in a number of ways.

As mentioned earlier, the target microcontroller enters Flash program mode when its MCLR line is driven to 12 volts. The 12 volts are generated by U9 which is a Linear Tech LT1615 switching regulator. U9 takes 5 volts from the USB interface, and outputs approximately 12.5 volts when the VPPSW input is high. The output voltage is set by resistors R20 and R23. The output of the regulator will generate whatever voltage is necessary to maintain 1.23 volts at the feedback pin (Pin 3).

U8 is a dual MOSFET switch configured to isolate the target electronics' power and ground from the USB interface when the host PC enters standby mode. If the target electronics are self-powered, then a jumper (JP4) can be removed. If using USB port power, be careful not exceed a total of 500 milli-amps for both the DLP-IOx board and your target electronics.

Both the DLP-IO16 and DLP-IO26 have an LED and a momentary contact switch that can be used for firmware debug or with the target electronics. Example code at the end of this article shows how to use the LED.

Since both the download and target microcontrollers need to access the USB interface, a pair of AND gates were added to handle the read and write functions. During the firmware download process, the target microcontroller is disabled, and its outputs to the AND gates are left high. This allows the download microcontroller to perform both the USB read and write functions. When the target microcontroller is running, the download microcontroller leaves its USB read and write outputs high so that the target micro can have access to the USB port.

A 74HC138 3-8 decoder was added to expand the input/output capability of the DLP-IO16. The 74HC138 allows the selection of USB read, USB write, digital data out, and digital data in. An example of how to access the decoder is shown in the next section.

Example C Source Code

Using the free C compiler from Hi-Tech, the following C source code will produce a HEX file that will run on the DLP-IO16. All the code does is toggle the LED at a rate that is received from the host PC via USB. Visual C++ source code for the Windows software that accompanies this example can be downloaded from my website (dlpdesign.com).

```
#include    <pic.h>

#define MUXA     RA0
#define MUXB     RA1
#define MUXG2A   RA2
#define TXE      RA3
#define RXF      RA4
#define LED      RB0

void long_delay(unsigned char x);//function prototype

//*****************************************************************
void main()
{
        unsigned char dtime;
        //set port A inputs/outputs
        //1=input  0=output
        TRISA4 = 1; //RXF
        TRISA3 = 1; //TXE
        TRISA2 = 0; //MUXG2A
        TRISA1 = 0; //MUXA
        TRISA0 = 0; //MUXB

        PORTB = 0xff;//Port B all high
        TRISB = 0xfe;//d0-output, the rest inputs

        MUXB = 1; //correct decode state for DOUT
        MUXA = 0;
        MUXG2A = 1; //data latch
        dtime=200; //initial setting for the LED flash rate

        while(1)//continuous loop
        {
```

```
		if(!RXF)//if USB chip reports a byte was received then read
		{
			MUXB = 0; //correct decode state for USB Read
			MUXA = 0;
			TRISB =0xff;//Port B set to input
			MUXG2A = 0;//enable read
			//read USB data without any error detection (crc, etc...)
			dtime = PORTB;//dtime=0 will cause watchdog timer to reset the device
			MUXG2A = 1;
			TRISB = 0xfe;//d0-output, the rest inputs
			MUXB = 1; //decode state for DOUT
			MUXA = 0;
		}

		LED=0;//LED on
		MUXG2A = 0;//latch data
		MUXG2A = 1;
		long_delay(dtime);

		LED=1;//LED off
		MUXG2A = 0;
		MUXG2A = 1;
		long_delay(dtime);
	}
}

//*********************************************************************
void long_delay(unsigned char x)
{
	int e;

	while(x--)
		for(e=0; e<330; e++)//1mS
			CLRWDT();
}
```

Conclusion

The firmware for your next project can be developed using either the DLP-IO16 or DLP-IO26, and then a finished product can be designed which excludes the download microcontroller as a cost-saving measure.

Many companies sell device programmers that will take a HEX file and write it to the Flash of a microcontroller. To my knowledge, no one offers a USB-based programmer simply because the speed of USB is not required. A minimum ten-millisecond delay must follow every write to Flash memory. Given this restriction, there is no advantage to designing a high-speed programmer. However, when the USB port remains available to the target microcontroller for host communications after firmware download--as is the case in the design presented here--then there is ample reason to include USB in the design.

Developing quality firmware can be challenging regardless of your level of experience.
Why not simplify the process by including a device programmer in the target electronics?