scene double

www.scene-double.co.uk

USB MSD Bootloader

Ray Gordon

ray@scene-double.co.uk

17 July 2008

Overview

Background

A requirement existed for an OS independent means to upgrade firmware on a USB PIC microcontroller without the use of any custom application or drivers on the host. The implementation needed to allow end-users (*not developers)* to upgrade embedded application firmware as simply and quickly as possible.

Perhaps the simplest solution is to allow the user to *drop* the firmware file on to the device in order to upgrade it.

This document describes the design of a Bootloader which appears to the host as a USB disk drive (Mass Storage Device). In order to upgrade application firmware, the user drags a specially formatted firmware file on to the drive.

This is not a Bootloader for those obsessed with fitting the code into the Boot Block. In fact the converse is true: in order to offer the user such simplicity, the firmware has to include a USB stack, MSD class handler, and File Allocation Table (FAT) emulation. The code space used by the example code is in the order of 7KB. If you can spare the memory this might be a useful solution.

We accept the valid school of thought that a Bootloader should be as simple as possible. Nonetheless, this has been an *interesting* project which could have wider application.

The source code released into the public domain is a cut-down version of that employed by our company in a commercial product. The project does have its quirks and we welcome any enhancements that are suggested.

Scope

The accompanying firmware is intended for use with Microchip PIC18 USB MCUs. The example code is presented as Microchip MPLAB projects and requires the C18 compiler. To aid evaluation, the code is targeted for the PICDEM FS-USB demo board which uses a PIC18F4550 controller, but is easily modified for other devices.

The Bootloader firmware consists of three elements:

USB Stack

A *slightly* modified version of the full v1.3 stack provided by Microchip Technology.

USB Mass Storage Device Driver

A modified and cut-down version of the the code provided by Microchip Technology and described in their Application Note AN1003.

Bootloader Core and FAT Drive Emulation

Provided by the author.

In addition, an example Windows application ***HEXStream*** has been provided by the author to format HEX files for use with the Bootloader.

License

Original code provided by the author is subject to the following license:

> Copyright (C) 2007-2008 Scene Double Ltd. / Ray Gordon
>
> This software is provided 'as-is', without any express or implied warranty. In no event will the authors be held liable for any damages arising from the use of this software.
>
> Permission is granted to anyone to use this software for any purpose, including commercial applications, and to alter it and redistribute it freely, subject to the following restrictions:
>
> 1. The origin of this software must not be misrepresented; you must not claim that you wrote the original software.
> 2. If you intend to use this software (or derivative) within a commercial product you must inform the authors prior to product release.
> 3. Altered source versions must be plainly marked as such, and must not be misrepresented as being the original software.
> 4. This notice may not be removed or altered from any source distribution.

Whilst the author's original code may be freely used with other (non-Microchip) processors, It should be noted that code authored by Microchip is only licensed for use with Microchip's own products.

Operation

Perhaps the easiest way to describe the Bootloader is to explain its basic operation.

Example Files

Two compiled example files are provided (for use with PICDEM FS-USB hardware).

File	Description
USB MSD Bootloader.hex	Bootloader firmware **Note:** This will overwrite the default Bootloader on the PICDEM FS-USB board. You must use a programmer to write this file as it cannot be loaded using the Microchip Bootloader application. We recommend you erase the PIC prior to programming.
Sample Application.pfw	A very simple user application (formatted with HEXStream) to be loaded using the Bootloader. The application simply displays preset LED patterns at regular intervals. The patterns may be changed using one of the features of the Bootloader. The application also allows the user to re-enter *boot mode.*

The following description assumes you have loaded USB MSD Bootloader.hex on to the demo board.

Basics

The Bootloader and user application are two separate programs. A means must be provided for the user application to enter *boot mode* and also for the Bootloader to enter *user mode*.

To ensure integrity, the Bootloader should not allow the user application to run unless a correct application image is loaded.

After reset the Bootloader will begin to run, and will only handover to the user application if *boot mode* is not requested and the user application is valid.

Bootloader Entry

The Bootloader will run (and enter *boot mode*) if:

- The user application is not loaded (or has previously failed to load correctly). This will be the case when you first run after loading USB MSD Bootloader.hex.
- The user has requested entry to *boot mode.* In the examples provided this will happen if switch, *S2,* is depressed when the board is powering up (or comes out of reset), or if the switch is depressed whilst in *user mode*.

Note: When in *boot mode* LED0 - LED3 will all be lit.

Boot Mode

After performing the previously described initial checks, the Bootloader will enumerate as a USB MSD called *Loader*.

Important: This is a virtual disk emulation on which specific operations may be performed with specially formatted files. You cannot use it as a (small) general purpose disk drive.

The device will report as a 2.56MB FAT16 drive with a formatted capacity of 511KB. The drive parameters have been chosen to ensure it operates under Windows, OS X & Linux. Please consult the source code for further details.

On most operating systems a Loader window will pop-up containing the following files:

File	Description
FW_xx	This is a virtual file which reports the current user firmware version. Where xx are the major & minor versions (e.g. FW_31 for v3.1). If the user application is not loaded (or has previously failed to load correctly), the file name will appear as FW_BAD. You may not perform any operations on this file.
BOOT	This file provides a means to quit *boot mode* and enter *user mode*. **Delete this file to quit *boot mode*.** You may not perform any other operations on this file **Note:** The Bootloader will remain in *boot mode* if the user application is not loaded or is invalid.
CONFIG	This file enables user application settings to be saved or updated, and may contain up to 506 bytes of data. In this example, the file contains the first 128 bytes of PIC EEPROM data. To save the data, simply drag or copy the file elsewhere. To update the data drag a suitably formatted data file on to the Loader window and the first 128 bytes of EEPROM will be changed. See the description below for more details.

Notes

1. Any files other than specifically formatted firmware or application settings files that you drag on to the Loader window are ignored by the Bootloader.

2. File dates and times are fixed and are always reported as 1st June 2008, 00:00.

Loading User Firmware

To load or upgrade the user application, simply drag a specially formatted firmware file on to the Loader window. If successful, the Loader window will be refreshed and will show the revised firmware version number.

As a first test, drag the file, *Sample Application.pfw,* on to the loader window. The version file will change name from FW_BAD to FW_10 (i.e. v1.0). To enter *user mode* you may now delete the BOOT file, and, after a few seconds, Sample Application will run.

Quirks

- After an update we attempt to 'refresh' the folder display to show the results of the operation. We do this by reporting that the media has been removed (in response to a SCSI Test Unit Ready command). Under Windows the refresh is instant. However, when used with other operating systems (such as OS X & some Linux), the MSD device will be reported as removed by the OS and will then automatically re-attach. You may need to re-open the emulated disk folder or do a folder refresh to correctly view the current status.

- Operating systems such as Linux (some variants) & Windows 2000 may take up to 50 seconds to write data/delete from any MSD device: WAIT & BE-CAREFUL! Note: If you want to reduce this time for your OS, investigate disabling write caching for handling surprise drive removal. It is fairly simple to adjust under Linux. However, unless you are doing a lot of updates it is probably better to wait.... OS X and Windows XP SP2 (or later) have no delay.

Note

When you drag a formatted firmware file on to the Loader window its name and extension are irrelevant. The Bootloader recognizes it as user firmware due to headers inserted by the formatting utility *HEXStream*. You may give your firmware files any name or extension

User Application Settings

In many applications it is useful to save/restore some configuration data. The Bootloader example illustrates a method for doing this via the CONFIG file (described above). In this case the CONFIG file reports the first 128 bytes of PIC EEPROM data.

The file is a binary file and has a fixed length of 512 bytes. The first six bytes are a specific header (described later) and the remaining 506 bytes are available for user data. In the Bootloader example the seventh byte will correspond to EEPROM address 0.

To save the file simply copy it or drag it off the Loader window.

You may edit the file (with a hex editor), *but not within the Loader window*. To update the config settings, drag the edited file back.

You may test this feature with the included Sample Application by editing the CONFIG file (away from the Loader window). Bytes 6 to 13 (i.e. EEPROM addresses 0 to 7) contain values (lower four bits) which correspond to which LEDs are lit. Edit these values and drag the file back on to the Loader window. When you quit *boot mode* the LED patterns will have changed.

The feature could easily be modified to provide a window on to a block of program memory instead of EEPROM.

Note

When you drag a formatted application settings file on to the Loader window its name and extension are irrelevant. The Bootloader recognizes it as config data due to the six byte header. You may give your config files any name.

Example Application

Description

A template application which demonstrates the Bootloader's features is included:

- *Sample Application* displays preset LED patterns (on LED0 to LED3) at 800mS intervals.
- The pattern sequence (wrapped) is stored in the first eight bytes of EEPROM.
- Depressing switch, S2, at any time will cause the Sample Application to quit and enter *boot mode.*
- When run for the first time, Sample Application writes a default pattern sequence (binary count) to the EEPROM. This sequence may then be altered/saved/restored through the Bootloader's CONFIG file feature (described above).

Source Code

Fully documented source code is included. The application may be compiled and run without the Bootloader, or formatted using HEXStream (as a .pfw) for use with the Bootloader.

File Formats

Basics

Data is written to (or read from) the emulated disk in 512 byte sectors. In order to make the firmware simpler (it is not a general purpose disk), a tag is required at the head of every firmware or config file sector written to the drive.

The firmware uses these headers to determine the type of data being written.

Firmware File Format

To further simplify the Bootloader, all firmware files must be pre-formatted. The *HEXStream* utility turns HEX files generated by MPLAB into formatted binary files for use with the Bootloader.

Within the the generated file, the following eight byte sequence is inserted every 512 bytes (sector start):

0xC1, 0xD2, 0xD3, 0x00, 0x00, 0xA1, 0xRR, 0xYY

where,

YY = 01 if the firmware file is encrypted (otherwise YY = 00), and RR is reserved (currently 00).

Following each sector header are 28 *lines* of 18 bytes. Each line contains an address word followed by 16 bytes of program data. Therefore, each sector contains 448 bytes of program data handy for programming 32/64-byte blocks of flash memory without concern about boundaries.

The current Bootloader firmware and formatting application are limited to program memory sizes of < 64KB. However, larger devices could be handled without changing this file format by using the reserved byte, RR, as the upper memory page number. The firmware and formatting application are easily modified.

It is important that the *whole* user application memory range (including areas of unused program memory) must be included in the image as its size needs to be known by the Bootloader (see Generating Firmware).

No checksums are included in the file. The formatting utility verifies the initial hex file and it is assumed that the error handling within USB makes further checks unnecessary.

The bootloader will ignore any memory address within its reserved memory area (i.e. it cannot be directly overwritten).

Application Data File Format

Application data files (CONFIG) are exactly 512 bytes in length and are in binary format.

Each file has the following six byte header:

0xC1, 0xD2, 0xD3, 0x00, 0x00, 0xAB

The remaining 506 bytes are user data.

Generating Firmware

Exporting a HEX File from MPLAB

Once you have written some firmware (using Sample Application as a guide) you will need to provide a hex file to format for use with the Bootloader.

DO NOT use the hex file generated directly by MPLAB when you compile the application, as this may not include the full memory range (we need a file of a known size), and also contains configuration bits which we do not want.

Instead, compile and the choose File->Export. Set it to only output Program Memory (no EEPROM or configuration bits). The memory range should be the whole range of the device (or, choose the start to be the starting address of your user application space, which is 0x2400 in the Sample Application example, and then end to be the top program memory address in the device).

HEXStream

This windows utility is included to format the above hex file for use with the Bootloader. The application is written in Visual Basic 2008 Express. Source code and executables are included.

Note: Installation of HEXStream may take a *long* time if you do not have .NET Framework 3.5 already installed. Treat yourself to several cups of coffee whilst this bloatware does its stuff (your machine has not crashed)!

Operation is very simple. Select the hex file to format and it will output a .pfw file of the same name.

You may also choose to encrypt the file stream. This is very basic and is really just obfuscation. You need to enter a single byte (hex) key. This must be the same key as defined in the Bootloader's *boot.h* file (currently set to 0xAA).

Generating a Single Bootloader/User Application Hex File

In a production environment it is normal to program the device with a single hex file containing both the Bootloader and current user application. To generate such a hex file you need to utilize MPLAB's merge facility. Here is a basic guide for doing this:

1. Enable MPLAB to open multiple projects. To do this navigate to Configure->Settings->Projects and uncheck 'Use one-to-one project-workspace model'.
2. Under Configure->Settings->Program Loading uncheck everything apart from 'Clear memory after successfully building a project'.
3. Open both the Bootloader and the application projects.
4. Select the application project and make it active. Compile it.
5. Select the Bootloader project and make it active. Compile it.
6. Choose File->Export and export the file with the Configuration Bits & User ID.
7. Save the workspace for future use.

Miscellaneous

Source Code

All source is included and the portions written by Scene Double are extensively documented.

No particular effort has been made to optimize the Bootloader size. The author uses all C18 optimizations and usually runs in Extended Mode.

In the examples the Bootloader uses program memory space up to 0x23FF (to allow for debugging, or use without full optimizations). This size may be reduced (see source) to <7k but you should ensure you end on a 64-byte boundary.

USB Stack

The Bootloader uses a modified version of the full Microchip v1.3 USB Device Stack.

No USB problems have (so far) been encountered and therefore no effort has been made to port the application to the v2.1 stack. This should not be a difficult task.

The Bootloader and stack work fine using *Extended Mode*.

USB Vendor ID & Product ID

The Bootloader example uses a VID/PID that belongs to Scene Double.

Please use your own VID/PID

Note: If your user application itself is USB then this will need a separate VID/PID to that used within the Bootloader.

Compatibility

The Bootloader has been tested under various flavours of Windows, Linux and OS X. It works best under Windows, and fine under other OS if you bear in mind the quirks (described above). If you get any problems hit reset (to re-enumerate).

Extending the Concept

This example illustrates a basic MSD Bootloader. The concept may be extended: as an example, in Scene Double's commercial product, multiple devices connected over slow serial links may be updated.

Enhancements

Please let us know about any errors, fixes or improvements so that we can update the project for everyone.

Contacting the Author

We would prefer if most comments are made via the Microchip forum. However, the authors will readily answer any *sensible* questions via e-mail.