

An I²C™ Bootloader for the PIC16F1XXX Enhanced Core

Author: Cristian Toma
Microchip Technology Inc.

INTRODUCTION

The new PIC16F1XXX enhanced core microcontroller has the ability to write its own program memory under software control. This allows the microcontroller to make modifications in the program Flash memory. This application note explains the implementation of a bootloader using I²C™ as a communication channel.

BOOTLOADER CONCEPT

The bootloader allows a processor to change its firmware without any physical intervention to the device itself. There is no need for a hardware programmer when using the bootloader. The firmware is downloaded from a host system by means of a communication channel, usually through a USB or a serial port. This bootloader implementation uses a standard I²C bus as a communication channel between the microcontroller and the host system.

In general, when using a bootloader, the code is transferred from a host device. Typically, this is comprised of a PC computer using a RS-232 serial cable or a USB port. Any other device can act as a host device.

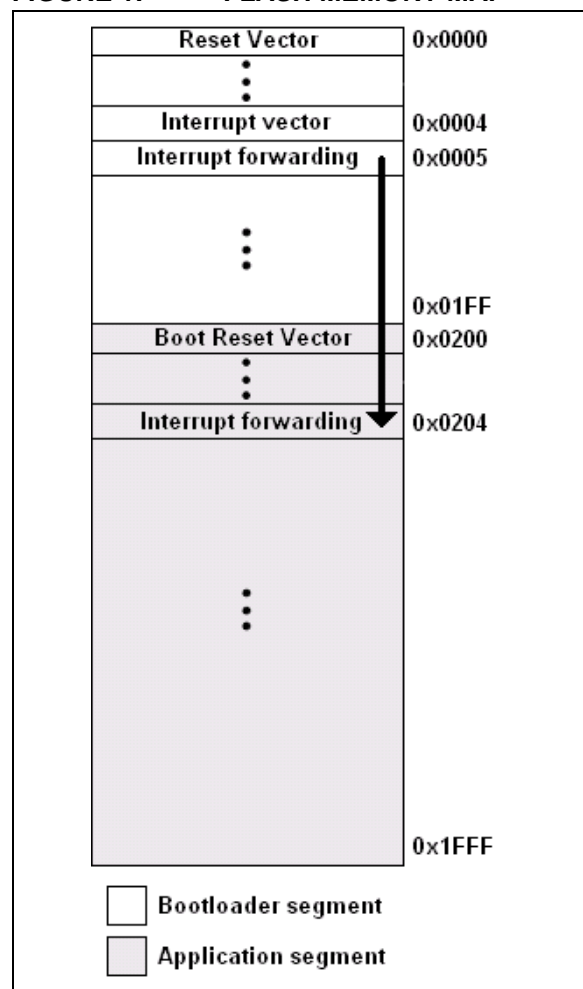
The target device needs bootloader firmware. This is basically a software that accepts commands and data from a host device and reads/erases/writes data to the Flash program memory. This firmware is typically located on the lower part of the program memory. The bootloader code is started by default at system Reset. It then checks if there is any application code loaded and jumps into application code if any is found. Alternatively, a button can be used to instruct the bootloader to stay in bootloader code even if a firmware is already present. This is useful when reading the application firmware or in case of an application firmware update.

BOOTLOADER REQUIREMENTS

The bootloader makes it unnecessary for any physical intervention with a hardware programmer. The target device must have some form of data connection with the host. In this application note, I²C communication is used. If the device uses the I²C port to communicate with other devices on the bus, then no additional connection is needed.

The bootloader will require the lower side of the program memory reserved for the bootloader firmware (see Figure 1). The rest of the program memory can be used by application. No RAM memory needs to be reserved, since it is used in different contexts and not at the same time.

FIGURE 1: FLASH MEMORY MAP



BASIC OPERATION

The basic functionality of a bootloader is to receive, interpret and execute a set of commands known both by the host software and the target firmware. The command set can be more or less complex than shown in the application note, but it must generally support commands to read, erase and write the Flash memory. Additionally, it must support a command to jump from the bootloader code to the application code. This command is typically issued at the end of a program/verify cycle. The application code might require a command to exit the application code and enter the bootloader code.

ERASING PROGRAM MEMORY

The program memory can be erased under software control only in rows of instructions. A row consists of 32 instruction words of program memory. Please note that the address must be aligned to a 32-word boundary (EEADRL <4:0> = 0 0 0 0).

Because the erase sequence can modify existing data, the special unlock sequence is needed. The unlock sequence consists in consecutive write values 0x55, 0xAA to the EECON2 register.

UPDATING PROGRAM MEMORY

Before writing to the program memory, an erase operation should be performed. The program memory is written in blocks. A block consists of 8 words of program memory data with sequential address, aligned to an 8-word boundary (EEADRL <2:0> = 0 0 0). If smaller blocks of memory need to be modified, then a read-modify-erase program approach should be used, as the program memory can only be written in 8-word blocks.

An erase operation sets the value of 0x3FFF to all erased locations. This corresponds to an all-ones in binary.

BOOTLOADER COMMANDS

The bootloader firmware used in this design supports a set of commands used to read, erase and write to the Flash memory of the target device. The commands are shown in Table 1.

TABLE 1: BOOTLOADER COMMANDS

Command	Name	Read/Write	Description
0x01	Set/Get Address Pointer	R/W	Sets/Reads the Address Pointer. Data is written/read in two bytes format, MSB first.
0x02	Download data to internal buffer	-/W	8 words of data are downloaded from the host into an internal buffer. Data is downloaded MSB first starting from the lower address. This command is usually followed by a Write command (0x05).
0x03	Read Flash command	R/-	Reads 8 words from the Flash memory starting with the address specified by the Address Pointer. The Address Pointer is automatically incremented by 8 words upon a read.
0x04	Erase Flash command	R/-	Erases a row of 32 words starting with the address specified by the Address Pointer. The Address Pointer must be aligned to a 32-word boundary (0x0000, 0x0020, 0x0040, 0x0060.....). The Address Pointer is automatically advanced by 32 words. This command uses clock stretching and returns a dummy 0x00 byte after command completion.
0x05	Write Flash command	R/-	The data that is stored into the buffer is written into the Flash memory at the address pointed to by the Address Pointer. The Address Pointer is automatically incremented by 8 words upon a successful write. This command uses clock stretching and returns a dummy 0x00 byte after command completion.
0x06	Jump to application code	R/-	Usually used after a successful firmware download. The application resets itself and jumps to the application segment. This command uses clock stretching and returns a dummy 0x00 byte after command completion.

BOOTLOADER FIRMWARE

As mentioned before, the bootloader firmware resides in the lower part of the Flash memory. The firmware receives, interprets and executes a set of I²C commands (the complete list of commands is described in Table 1). In order to respond to the I²C commands, the firmware must access the Flash memory. For some commands that involve operations with the internal Flash memory that takes time to complete the operation. These commands are implemented as I²C Read command. They will return a dummy 0x00 byte after the operation has completed. During this time, the slave will clock stretch the I²C bus by pulling the SCL line low to signal the master device that the operation has not completed. The download of data to the internal buffer does not need any response from the target system, thus the command is implemented as a normal I²C Write command.

The structure of the I²C commands is described in the following. These are implemented as the target being an I²C slave device.

General configuration of an I²C write operation (see Table 2):

1. Send a "Start" condition to the slave device
2. Send the device address with the R/W bit low (even address). If the sent address matches the slave device address, then the slave device sends an ACK signal.
3. Send the internal register number you want to write to.
4. Send data. Depending on the command used, the master can send multiple data bytes. Please note that the master device can send a multiple number of bytes.
5. Send a "Stop" condition.

TABLE 2: I²C WRITE COMMAND

Start	Address	Register	Data (nbytes)	Stop
-------	---------	----------	---------------	------

General configuration of an I²C read operation (see Table 3):

1. Send a "Start" condition.
2. Send the device address with the R/W bit low (even address). Notice that the slave sends an Acknowledge signal if the address matches.
3. Send the bootloader command number you want to read from.
4. Send a "Restart" condition.
5. Send the device address again. This time with the R/W bit high (odd address).
6. Read a number of bytes from the slave device. If data is not yet available, the slave will use clock stretching (pull SCL line low).
7. Send a "Stop" condition

TABLE 3: I²C READ COMMAND

Start	Address	Register	Restart	Address	Data (nbytes)	Stop
-------	---------	----------	---------	---------	---------------	------

The bootloader code must exit after completing the download of the new firmware. This is typically implemented using a Reset instruction. The decision to go to bootloader code or application code is based on a bootloader flag placed in the Flash memory. The location of this flag is at the highest address of Flash memory for the target device. In the current implementation using the PIC16F1937, this location is 0x1FFF. Other devices might have more or less available Flash program memory and the last available Flash memory location will vary accordingly. Following an Erase command, the data at address 0x1FFF is 0x3FFF. After a successful code download and verify, the data contained at this address will be changed to a special value (0x55), indicating that an application is loaded. When the application code needs to jump to the bootloader code it will write an all-zeros value to the bootloader flag address. When writing an all-zeros value to a Flash memory location there is no need for a previous erase operation. An erase operation sets the Flash memory location to an all-ones value. The special value can be either contained in the application hex file or written by the bootloader upon a successful code update.

Alternatively, any means of code-present flags can be used, such as flags in EEPROM memory or any other address in the Flash memory.

The full source code of the bootloader is available with this application note. The firmware runs on the F1 Evaluation Platform containing a PIC16F1937. Three demo applications are provided along with this application note. These are relatively simple and are provided with the purpose of demonstrating firmware upload. The first two applications are just "Hello world" kind of firmware. The third application showcases the use of interrupt forwarding.

FIRMWARE FUNCTION DESCRIPTIONS

Among the functions of the bootloader firmware, some of them are more important:

`do_i2c_tasks`: This function services all the I²C related events. The event decoding is based on the SSPIF flag and the SSPSTAT register.

`flash_memory_read`: This function takes a 16-bit value as a address parameter and returns the data read from Flash memory at the specified location.

`flash_memory_write`: This function has two parameters. The first one is a 16-bit value as an Address Pointer. Please note that the address must be properly aligned to an 8 word boundary. The second parameter is a pointer to buffer containing the data being written. It does not return any value (void).

`flash_memory_erase`: Performs an erase operation starting with the address specified as a parameter. 32 instruction words are erased in a row and the specified address must be aligned to a 32 word boundary.

`service_isr`: This function performs the interrupt forwarding feature. It is implemented as a GOTO instruction. For more information see the following section.

INTERRUPT FORWARDING

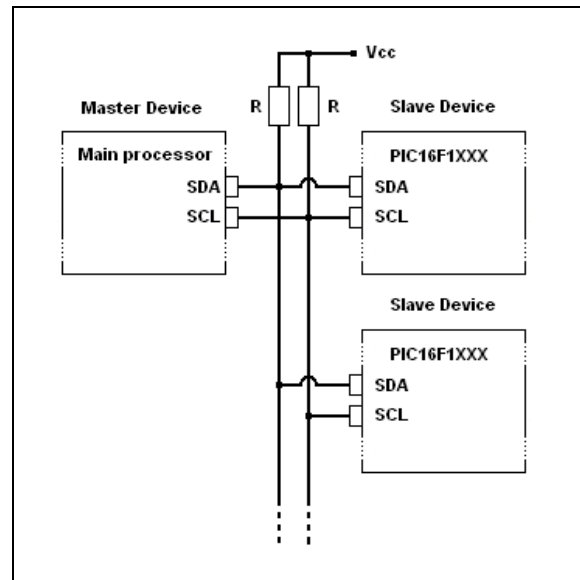
The PIC16F1XXX enhanced core has the interrupt vector at 0x0004. This address is located in the area where the bootloader firmware is located. To maintain the interrupt functionality it is required that the Interrupt Service Routine (ISR) be relocated at a different address, in the application segment. In the bootloader code, at the interrupt vector location (0x0004), a GOTO instruction is present. The Interrupt Service Routine is contained in the application code. The application code is shifted by 0x200, leaving room for the bootloader in the lower part of the Flash memory. It will also contain the Interrupt Service Routine at 0x0204. The bootloader code will contain a "GOTO 0x0204" instruction. When an interrupt occurs, the program counter (PC) jumps to 0x0004 in the bootloader segment. At this address, the "GOTO 0x0204" instruction will cause the PC to jump to 0x0204 in the application segment, where the Interrupt Service Routine is located (see Figure 1). Please notice that the PIC16F1XXX enhanced core features Automatic Context Saving.

The interrupt latency will be increased by the GOTO instruction execution time. See the example demo application(s) for more details.

MULTIPROCESSOR SYSTEM

Since the proposed bootloader uses a standard I²C protocol, the target microcontroller can be interfaced with a larger microcontroller having more processing power (such as a main processor in a multiprocessor system). Several such devices can be connected in parallel on the I²C bus, each slave device having its own address (see Figure 2).

FIGURE 2: MULTIPLE SLAVE DEVICES CONNECTED TO THE SAME BUS



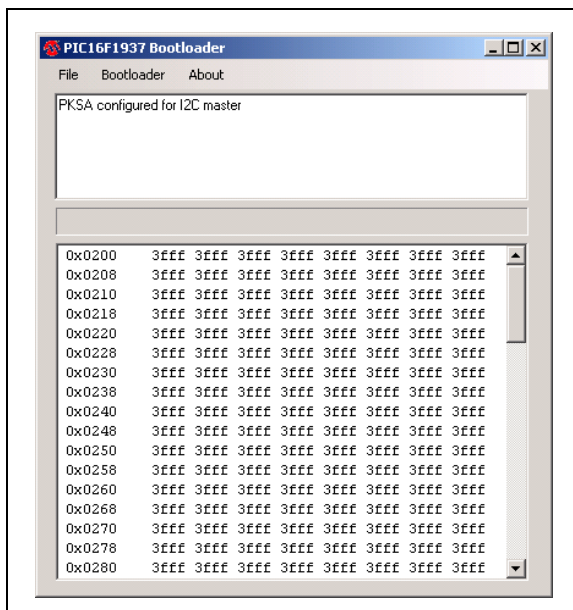
PC SIDE SOFTWARE

The bootloader presented here uses an I²C bus as a communication channel. The main use for such a bootloader is in a multiprocessor system where the main processor can update the firmware of several processors in the same system.

We can, however, use PC software if we use a device that will act as a master I²C device controlled by PC software. We used the PICkit™ Serial analyzer configured as I²C master device.

The software is written using Visual C# 2008, Express Edition. The PC software has the primary function of importing the hex file, splitting it into smaller packets and sending the data to the bootloaded device. Reading back program data from the device is also possible. When designing the firmware for the main processor in a multiprocessor system, the above software can serve as a starting point.

FIGURE 3: PC HOST APPLICATION WINDOW



To upload a new hex file to the device you must first open the hex file. This can be done using the Import Hex command found in the File menu (see Figure 3). You will notice that the content of the currently loaded file is displayed in the lower side of the window. Once the new firmware is loaded, it is ready to be sent to the target device. This is done by using the Program command under the Bootloader menu. The programming operation will consist in a full erase followed by a firmware download. The erase-only operation is available in the same menu. The firmware can also be read from the target device using the Read command in the Bootloader menu. The program memory data can be saved into a hex file using the Export Hex command in the File menu. After a successful firmware download operation, switching to the application code is done using the Run firmware command in the Bootloader menu.

CONCLUSION

Using a bootloader is a very efficient way to allow firmware upgrades in the field. Products can be easily upgraded to support new features. Fixing code bugs is also easier.

The bootloader code occupies the lower part of the program memory between 0x0000 and 0x01FF (512 bytes). The bootloader features interrupt forwarding to allow applications to use interrupts.

When using I²C communication, multiple bootloaded devices can be connected on the same bus, each device having its own address.

AN1302

NOTES:

Note the following details of the code protection feature on Microchip devices:

- Microchip products meet the specification contained in their particular Microchip Data Sheet.
- Microchip believes that its family of products is one of the most secure families of its kind on the market today, when used in the intended manner and under normal conditions.
- There are dishonest and possibly illegal methods used to breach the code protection feature. All of these methods, to our knowledge, require using the Microchip products in a manner outside the operating specifications contained in Microchip's Data Sheets. Most likely, the person doing so is engaged in theft of intellectual property.
- Microchip is willing to work with the customer who is concerned about the integrity of their code.
- Neither Microchip nor any other semiconductor manufacturer can guarantee the security of their code. Code protection does not mean that we are guaranteeing the product as "unbreakable."

Code protection is constantly evolving. We at Microchip are committed to continuously improving the code protection features of our products. Attempts to break Microchip's code protection feature may be a violation of the Digital Millennium Copyright Act. If such acts allow unauthorized access to your software or other copyrighted work, you may have a right to sue for relief under that Act.

Information contained in this publication regarding device applications and the like is provided only for your convenience and may be superseded by updates. It is your responsibility to ensure that your application meets with your specifications. MICROCHIP MAKES NO REPRESENTATIONS OR WARRANTIES OF ANY KIND WHETHER EXPRESS OR IMPLIED, WRITTEN OR ORAL, STATUTORY OR OTHERWISE, RELATED TO THE INFORMATION, INCLUDING BUT NOT LIMITED TO ITS CONDITION, QUALITY, PERFORMANCE, MERCHANTABILITY OR FITNESS FOR PURPOSE. Microchip disclaims all liability arising from this information and its use. Use of Microchip devices in life support and/or safety applications is entirely at the buyer's risk, and the buyer agrees to defend, indemnify and hold harmless Microchip from any and all damages, claims, suits, or expenses resulting from such use. No licenses are conveyed, implicitly or otherwise, under any Microchip intellectual property rights.

Trademarks

The Microchip name and logo, the Microchip logo, dsPIC, KEELOQ, KEELOQ logo, MPLAB, PIC, PICmicro, PICSTART, rfPIC and UNI/O are registered trademarks of Microchip Technology Incorporated in the U.S.A. and other countries.


FilterLab, Hampshire, HI-TECH C, Linear Active Thermistor, MXDEV, MXLAB, SEEVAL and The Embedded Control Solutions Company are registered trademarks of Microchip Technology Incorporated in the U.S.A.

Analog-for-the-Digital Age, Application Maestro, CodeGuard, dsPICDEM, dsPICDEM.net, dsPICworks, dsSPEAK, ECAN, ECONOMONITOR, FanSense, HI-TIDE, In-Circuit Serial Programming, ICSP, Mindi, MiWi, MPASM, MPLAB Certified logo, MPLIB, MPLINK, mTouch, Octopus, Omniscient Code Generation, PICC, PICC-18, PICDEM, PICDEM.net, PICKit, PICtail, PIC³² logo, REAL ICE, rLAB, Select Mode, Total Endurance, TSHARC, UniWinDriver, WiperLock and ZENA are trademarks of Microchip Technology Incorporated in the U.S.A. and other countries.

SQTP is a service mark of Microchip Technology Incorporated in the U.S.A.

All other trademarks mentioned herein are property of their respective companies.

© 2009, Microchip Technology Incorporated, Printed in the U.S.A., All Rights Reserved.

 Printed on recycled paper.

QUALITY MANAGEMENT SYSTEM
CERTIFIED BY DNV
== ISO/TS 16949:2002 ==

Microchip received ISO/TS-16949:2002 certification for its worldwide headquarters, design and wafer fabrication facilities in Chandler and Tempe, Arizona; Gresham, Oregon and design centers in California and India. The Company's quality system processes and procedures are for its PIC[®] MCUs and dsPIC[®] DSCs, KEELOQ[®] code hopping devices, Serial EEPROMs, microperipherals, nonvolatile memory and analog products. In addition, Microchip's quality system for the design and manufacture of development systems is ISO 9001:2000 certified.



WORLDWIDE SALES AND SERVICE

AMERICAS

Corporate Office
2355 West Chandler Blvd.
Chandler, AZ 85224-6199
Tel: 480-792-7200
Fax: 480-792-7277
Technical Support:
<http://support.microchip.com>
Web Address:
www.microchip.com

Atlanta
Duluth, GA
Tel: 678-957-9614
Fax: 678-957-1455

Boston
Westborough, MA
Tel: 774-760-0087
Fax: 774-760-0088

Chicago
Itasca, IL
Tel: 630-285-0071
Fax: 630-285-0075

Cleveland
Independence, OH
Tel: 216-447-0464
Fax: 216-447-0643

Dallas
Addison, TX
Tel: 972-818-7423
Fax: 972-818-2924

Detroit
Farmington Hills, MI
Tel: 248-538-2250
Fax: 248-538-2260

Kokomo
Kokomo, IN
Tel: 765-864-8360
Fax: 765-864-8387

Los Angeles
Mission Viejo, CA
Tel: 949-462-9523
Fax: 949-462-9608

Santa Clara
Santa Clara, CA
Tel: 408-961-6444
Fax: 408-961-6445

Toronto
Mississauga, Ontario,
Canada
Tel: 905-673-0699
Fax: 905-673-6509

ASIA/PACIFIC

Asia Pacific Office
Suites 3707-14, 37th Floor
Tower 6, The Gateway
Harbour City, Kowloon
Hong Kong
Tel: 852-2401-1200
Fax: 852-2401-3431

Australia - Sydney
Tel: 61-2-9868-6733
Fax: 61-2-9868-6755

China - Beijing
Tel: 86-10-8528-2100
Fax: 86-10-8528-2104

China - Chengdu
Tel: 86-28-8665-5511
Fax: 86-28-8665-7889

China - Hong Kong SAR
Tel: 852-2401-1200
Fax: 852-2401-3431

China - Nanjing
Tel: 86-25-8473-2460
Fax: 86-25-8473-2470

China - Qingdao
Tel: 86-532-8502-7355
Fax: 86-532-8502-7205

China - Shanghai
Tel: 86-21-5407-5533
Fax: 86-21-5407-5066

China - Shenyang
Tel: 86-24-2334-2829
Fax: 86-24-2334-2393

China - Shenzhen
Tel: 86-755-8203-2660
Fax: 86-755-8203-1760

China - Wuhan
Tel: 86-27-5980-5300
Fax: 86-27-5980-5118

China - Xiamen
Tel: 86-592-2388138
Fax: 86-592-2388130

China - Xian
Tel: 86-29-8833-7252
Fax: 86-29-8833-7256

China - Zhuhai
Tel: 86-756-3210040
Fax: 86-756-3210049

ASIA/PACIFIC

India - Bangalore
Tel: 91-80-3090-4444
Fax: 91-80-3090-4080

India - New Delhi
Tel: 91-11-4160-8631
Fax: 91-11-4160-8632

India - Pune
Tel: 91-20-2566-1512
Fax: 91-20-2566-1513

Japan - Yokohama
Tel: 81-45-471- 6166
Fax: 81-45-471-6122

Korea - Daegu
Tel: 82-53-744-4301
Fax: 82-53-744-4302

Korea - Seoul
Tel: 82-2-554-7200
Fax: 82-2-558-5932 or
82-2-558-5934

Malaysia - Kuala Lumpur
Tel: 60-3-6201-9857
Fax: 60-3-6201-9859

Malaysia - Penang
Tel: 60-4-227-8870
Fax: 60-4-227-4068

Philippines - Manila
Tel: 63-2-634-9065
Fax: 63-2-634-9069

Singapore
Tel: 65-6334-8870
Fax: 65-6334-8850

Taiwan - Hsin Chu
Tel: 886-3-6578-300
Fax: 886-3-6578-370

Taiwan - Kaohsiung
Tel: 886-7-536-4818
Fax: 886-7-536-4803

Taiwan - Taipei
Tel: 886-2-2500-6610
Fax: 886-2-2508-0102

Thailand - Bangkok
Tel: 66-2-694-1351
Fax: 66-2-694-1350

EUROPE

Austria - Wels
Tel: 43-7242-2244-39
Fax: 43-7242-2244-393

Denmark - Copenhagen
Tel: 45-4450-2828
Fax: 45-4485-2829

France - Paris
Tel: 33-1-69-53-63-20
Fax: 33-1-69-30-90-79

Germany - Munich
Tel: 49-89-627-144-0
Fax: 49-89-627-144-44

Italy - Milan
Tel: 39-0331-742611
Fax: 39-0331-466781

Netherlands - Drunen
Tel: 31-416-690399
Fax: 31-416-690340

Spain - Madrid
Tel: 34-91-708-08-90
Fax: 34-91-708-08-91

UK - Wokingham
Tel: 44-118-921-5869
Fax: 44-118-921-5820