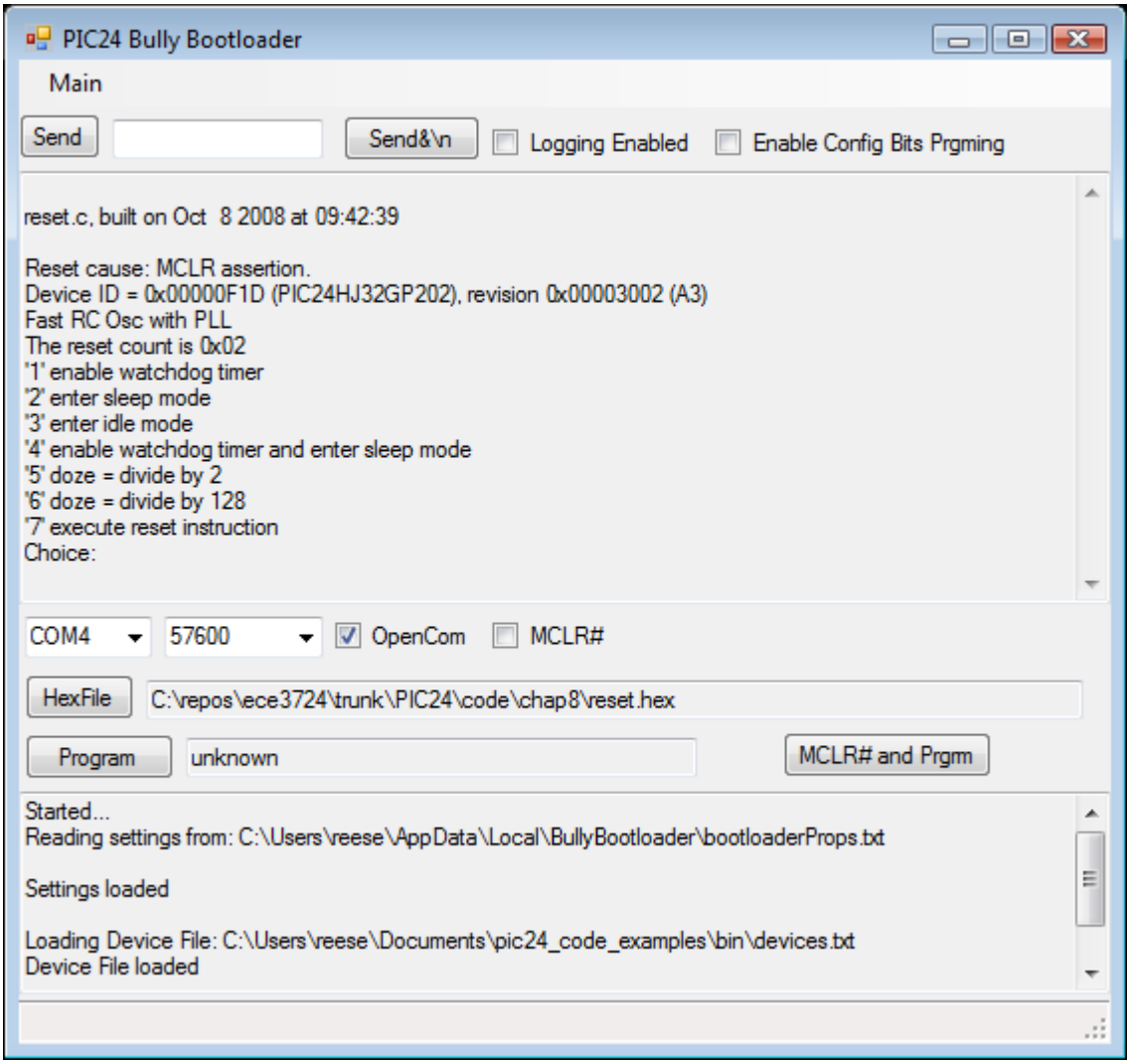
www.reesemicro.com has more documentation, and the source code.

PIC24 Bully Bootloader Documentation

This is some minimal documentation on the PIC24 Bully bootloader for the PIC24H/F families. This bootloader runs under Windows, a screenshot is below:



This bootloader was developed from *C* and assembly code originally distributed by Microchip (see the enclosed AN1094 application note. I have tested the bootloader on the p24FJ64GA002, p24HJGP202, and p24HJ32GP202 targets. The bootloader window application is a .NET application and the supplied executable has been tested with WinXp, Vista, and Vista64. The supplied executable has been reported to be incompatible with WinXP64; but recompiling from the source may work (untested, the source files are supplied in the archive).

After unpacking, you will have the following directories (assuming that you got this from the complete code archive at www.reesemicro.com)

www.reesemicro.com has more documentation, and the source code.

- **bootloader/24h_24f_target/** - Source and MPLAB project for the 24H/F bootloader firmware code to be loaded on the target processor. You will probably have to modify this source to map the UART TX/RX pins to your desired target, as well change to your desired starting baudrate, clock configuration. The current code uses RB10 for RX, RB11 for TX, and the internal oscillator @ 40 MHZ FCY (PIC24H). If you compile to a new target, please be aware that you have to provide a customized linker file for the bootloader so that it starts at location 0x400, and not the default of 0x200. Some linker files for the bootloader are in this directory.
- **bootloader/winbootldr/** - Source for windows bootloader application (Visual Studio 2005)
- **hex/** - Some bootloader hex firmware files for some targets built from the 24h_24f_target/ source.
- **lkr/** - these are some sample linker files for difference processors that are used when building your application to be compatible with the bootloader. Look at the comments inside the linkder files as well as the AN1094 application to see the changes from the standard linker files. Be careful when modifying linker files for the PIC24F family, you must change both the code origin and length fields (see the sample PIC24F linker file provided and the comments).
- **bin/** - Contains a *winbootldr.exe* file that is the Windows bootloader application. To install copy this .exe and the *devices.txt* file to some target directory. The *devices.txt* file must have an entry for your target processor; the comments at the top of this file indicate the format of each record. If you processor is not currently listed, simply use a text editor to add it.

Installation of the Windows bootloader application

Copy the *bin/winbootldr.exe* and *bin/devices.txt* to a target directory and try executing *winbootldr.exe*. If it does not run a couple of problems may be:

- This is a .NET application, so try installing the latest .NET runtime from Microsoft (try this first).
- You may need to unzip and install the *vcrdistr_x86.zip* which are some runtime DLLs that come with Microsoft Visual Studio 2005.

.NET Requirements

You need at least .NET 2.0.XXX or above runtime framework installed. You can download this by searching for ".NET framework" at www.microsoft.com/downloads . If you do not have at the .NET runtime installed, then you may get this error message:

www.reesemicro.com has more documentation, and the source code.

Using the bootloader

This assumes you have a PIC24H/F target programmed with the bootloader firmware, and connected to your PC via a serial port. If you have the RTS# line the USB-to-Serial cable connected to MCLR#, then the RTS# checkbox pulls MCLR# low when checked, and MCLR# when unchecked.

In the bootloader application:

1. Choose the appropriate COM part and baud rate (the default baudrate in the bootloader firmware is 57600, the bootloader firmware does not autobaud). Click on the port checkbox to open the port.

2. Use the "HexFile" button to browse to a hex file for download. The Hex file must be have been compiled using a modified linker file such as those found in **lkr/.** Our default test program is the chap8/reset.mcp project. The project assumes a PIC24HJ32GP202 processor and uses the linker file lkr/p24HJ32GP202_bootldr.gld file.

3. **Downloading a program via Power cycle or MCR#:** Cycle power or assert MCLR# to your target PIC24H/F; when power is applied or reset is released, you have about 2 seconds to press the "Program" button in the Bootloader window. If the bootloader is able to establish a connection, you will see your hex file downloaded to the target device.

4. **Downloading a program via "MCLR# and Prgrm" button:** If you have the RTS# or DTR# line of the USB-to-Serial cable connected to MCLR#, then press this button to program. The bootloader will pulse the MCLR# input via RTS# or DTR# (both are pulsed), and then download the program.

5. The upper part of the bootloader application window shows what is happening on the serial port, you can use the 'Send' button to send data in the type-in field to your application via the serial port (the 'Send&\n' sends the data with a new line)

If the bootloader does not work

- If the bootloader hangs after programming the first block, be sure that you have compiled the bootloader firmware so that it starts at 0x400 and not 0x200 (if it starts at 0x200, it will erase itself).

- If the bootloader complains that the device is unrecognized, add your target device to the 'devices.txt' file.

- If the bootloader does not connect all, this could be a multitude of problems on the target side - your clock is not configured correctly, the baud rate is wrong, the TX/RX pins are configured incorrectly. To test this, just connect a dumb terminal program to the bootloader, and modify the firmware to put out a 'hello' message to see if your serial port is working correctly.

www.reesemicro.com has more documentation, and the source code.

The Bootloader Firmware and Persistent Data

The bootloader's firmware stack pointer is initialized around 0x0E50 the last time I checked. This means that if your application has persistent data above that point, it may (will) get stepped on. The bootloader's static data buffer that takes up the space before this is persistent, so any of the application's persistent data in this space will be safe. If the bootloader loads a program, then it sets the POR bit to simulate a power-on reset for the new program; this way if your program checks the POR bit to initialize persistent data you will get correct behavior.

The Bootloader and Configuration Bits

Configuration bits on PIC24H/PIC24F devices determine things like initial clock source selection, watchdog timer timeout, etc. The file **24h_24f_target/pic24_configbits.c** file sets the configuration bits for the bootloader firmware, currently the bootloader uses the internal FRC +PLL as the initial clock source (16MHz FCY for PIC24F, 40 MHz FCY for PIC24H). Feel free to modify these config bits to whatever you need.

For PIC24H devices, configuration bits are located in a special area of flash memory. For PIC24F devices, a packed version of the configuration bits are located near the end of the last page of flash memory, and at device reset time, these are unpacked into the configuration registers.

Configuration bits can also be specified in the application hex file that is downloaded by the bootloader.

Before version 0.19, the bootloader and associated firmware had the following behavior in terms of configuration bits programming if the configuration bits were present in the hex file.

- PIC24H devices: configuration bits always programmed if present in the hex file.
- PIC24F devices: configuration bits were never programmed, the last page of flash memory for PIC24F devices was not programmed (1 page is 64 * 8 instructions = 512 instructions).

Beginning with Version 0.19, there is now a checkbox that allows control of configuration bit programming if they exist in the application hex file. This checkbox setting is only used if the Version 2.0 or better of the firmware is loaded (the bootloader now checks the firmware version during programming, and if it lower the Version 2.0, then the old behavior for configuration bit programming is used).

WithVersion 0.19 and later, if the configuration bit programming is enabled and configuration bits are present in the application hex file, then the configuration bits are programmed for both PIC24H and PIC24F devices. If configuration bit programming is disabled, then configuration bits are not programmed for either PIC24H or PIC24F devices. For PIC24F devices, this has the nasty side effect of not programming the last page of flash memory (the last 512 instructions of program memory), so make sure that you do not have any program code there.

www.reesemicro.com has more documentation, and the source code.

The reason to disable configuration bit programming is that the bootloader may become inoperable if the application has incorrect configuration bit settings (i.e. specify an external crystal as a clock source and there is no crystal present).

I would recommend for PIC24H devices to set the bootloader configuration bits the way you want them in the bootloader firmware by editing **24h_24f_target/pic24_configbits.c** , and then disable configuration bit programming when downloading applications. This way, you cannot kill the bootloader via incorrect configuration bits in the application.

For PIC24F devices, there is not a good solution. If you enable configuration bit programming, then you may kill the bootloader if the application has incorrect configuration bits. If you disable configuration bit programming, then the last page of flash memory is not useable as the bootloader aborts programming if it detects program instructions (and not configuration bits) on the last page of flash memory. The choice is up to you.

Comments

Send comments to Bob Reese (reese@ece.msstate.edu).

Version 0.31 Jan 2011:

User-submitted bug fix for program code > 64K.

Version 0.3 (version bumped to match bootloader firmware of V3.0) August 2010:

Both the firmware and application GLD files have been changed to remap the interrupt vector table such that the bootloader space is never written during bootloading. Previously, memory page 0 was written during the download process in order to copy the application's interrupt vectors -- if this page write to page 0 failed, then a 'dead bootloader' could result. This costs an extra 0x200 in program space usage, and also a few cycles of extra latency in interrupt service. We had a few complaints of dead bootloaders, and so made this switch. The GUI will detect if it is connected to a bootloader of firmware version less than 3.0 and will do the old behavior. Because the application and bootloader GLD files now require more complex changes to them, a Python script that is now included (**code/bootloader/24h_24f_target/lkr/convert_gld.py**) that will produce all necessary bootloader linker files (**code/bootloader/24h_24f_target/lkr/*.gld**) and application linker files **(code/lkr/*.gld**).

The python script is:

code/bootloader/24h_24f_target/lkr/convert_gld.py

The script looks in the default root directories for the *.gld* files:

C:\Program Files (x86)\Microchip\MPLAB C30\support
C:\Program Files\Microchip\MPLAB C30\support

If the script cannot find one of these two directories, then the script aborts and prints an error message -- if you have installed C30 in a different directory, then edit the setting of the *'C30_homedir*' variable in the script.

Otherwise, just open a command window, change to *the code/bootloader/24h_24f_target/lkr/* directory, and execute:

python convert_gld.py

It will convert all PIC24, PIC24F linker files that it finds. Bootloader linker files are placed in this directory, while the application linker files are placed in *code/lkr*.

Version 0.24, Sept 2009

Changes to both firmware and GUI to support the PIC24FK family (has only been tested on the PIC24F16KA102).

Version 0.23, Sept 2009

Minor change to support dsPIC33 family.

Version 0.21-22 Summer 2009

Minor changes to support higher baud rates

Version 0.20, Nov 23 2008

Fixed a problem in the configuration bit programming - configuration bits were still being programmed even if the check box for disabling this was checked. A side effect was that configuration bits could be corrupted in this case. Also, verification was fixed - it was indicating a match even if the program contents did not match.

Version 0.19, October 9 2008

Added support for enabling/disabling configuration bit programming, see the section titled “The Bootloader and Configuration bits”. The changes affected both the GUI and the firmware. Also, the COM port, baud rate, and other settings are now saved on exit in the local applications folder, and restored when restarted.

Version 0.18, October 7 2008

Fixed a problem with the configuration bits sometimes getting corrupted by the verification process.

www.reesemicro.com has more documentation, and the source code.

Version 0.17, September 3 2008

GUI changes: Added verification of program memory after programming, and a check box for enabling logging (the log file is named *bullyBootloaderLog.txt* is in the same directory as the bootloader executable).

Version 0.16, Aug 19 2008

Firmware now sets the POR bit if a program is loaded, this way a new program thinks that a POR has occurred and the C runtime initializes persistent variables. Also, added a the RTS#(MCLR#) check box and the "RTS# and Prgrm" button to the bootloader.

Version 0.15, Jul 13 2008

Added firmware support for Explorer16/100pin board (see 24h_24f_target/Explorer16_100p_bootloader.mcp). Fixed problem with bootloader GUI not detecting location clash between user application and bootloader firmware - bootloader GUI will not load application code if a clash exists).

Version 0.14, Jun 5 2008

If no serial ports exits, handle gracefully.

Version 0.13, May 27 2008

Added a 'small RAM' flag to devices.txt to flag devices with less < 2K RAM, such as the PIC24HJ12GP202. They cannot hold an entire program memory page in SRAM at time, so programming is done a half-page at a time. This also affected the firmware in the 24H_F_target directory.

Version 0.12, May 26 2008

Added some fixes by He Wen Guang, with the main being to change the configuration space size for 24H devices to include the User ID words. This affected the firmware in the 24H_F_target directory as well.

Version 0.11, May 23 2008

Added revision name to processor ID that is printed out, and restored process ID to the 'devices.txt' file. Processor that connect is now checked for both correct device ID and process ID (upper byte of revision number).

Initial Release

www.reesemicro.com has more documentation, and the source code.

Version 0.10, May 19 2008