Encrypted USB HID Bootloader System

© 2003-2010 Andrew Smallridge
asmallri@brushelectronics.com
www.brushelectronics.com

Brush Electronics' Encrypted USB HID Bootloader have been developed to support remote firmware upgrade for the Microchip PIC base Micrcontroller systems deployed in the field. The Encrypted USB HID Bootloader is derived from our Encrypted Ethernet Bootloaders that operated on Encrypted image files utilizing the XTEA (eXtended Tiny Encryption Algorithm).

The Brush Electronics' Encrypted USB HID Bootloader System comprises of three main elements:

1. Hex File Encrypter application
2. Bootloader Firmware resident in the target hardware platform
3. Programmer application executing on the controlling host

For the sake of subsequent explanation, the term Bootloader refers to the code (firmware) executing in the PIC microprocessor (target), the Programmer refers to the programming application running on the controlling host computer and the Encrypter refers to the application that encrypts the original hex file.

The Brush Electronics' Encrypted USB HID Bootloader is available for the Microchip PIC32 family of Microcontrollers.

Key attributes of Brush Electronics' Encrypted USB HID Bootloaders:

- The HID interface does not require a custom USB driver on the programming host PC

- Incremental Bootloader. As little as a single byte can be modified

- No resources are required on the target PIC other than the flash memory holding the boot code

- The PIC32 version is located in low program memory. User applications must be compiled with a linker script reserving the Bootloader memory. Sample linker scripts for linking the User Application are included in the system package.

- Encrypts standard Intel Hex Files

Bootloader Memory MAP

The Bootloader code resides in a portion of the program memory space. Because this program memory is flash memory based, bootloader configuration parameters are also stored in this space in the *bldr_param_mem* memory region defined in the bootloader's linker script.

The user's code MUST implement a GOTO instruction (a long jump) in the first 4 instructions. Typically this GOTO instruction is automatically inserted during the linking stage. The Microchip C32 compiler, when used with its respective standard linker scripts, automatically insert the GOTO instruction.

The PIC32 version of the Bootloader is located in low program memory from 0x9D00_0000 to approximately 0x9D00_6FFF. The Bootloaders only use a fraction of the space currently allocated with the balance of free space left to allow developers to enable the extensive debugging capabilities incorporated in the code via a series of #define conditional compile directives and to accommodate significant customisation of the software. Developers can readily minimize the Bootloader's memory footprint by modifying the Bootloader's linker script to move the parameter block *bldr_param* lower down in program memory to just above the Bootloader's code and then modifying the applications's linker script to use the freed up space. Refer to the platform specific header file platform.h and associated linker script for the actual memory map usage.

The PIC32 version of the Bootloader does not intercept the user applications reset vector – it does not need to do so. Instead it executes a GOTO to the application program's "well known" entry point as defined in the application linker script.

Basic Operation

The Bootloader code in the PIC and PC based programmer application use flags to control the boot load process. Two of these flags determine the boot up behaviour of the PIC:

- The PENDING RESET VECTOR is used to indicate that the user's RESET vector has not yet been received.
- The INVALID USER CODE SPACE flag indicates that an Intel End-of-File hex record has not yet been received.

The PIC's Bootloader code is executed automatically as a result of a power on reset, a reset command or a jump to absolute address 0x0000. Wherever possible a CPU reset should be used to enter the Bootloader as this forces all registers associated with interrupts to a known state. The Bootloader is now in BOOTLOADER DISCOVERY MODE. If either of the two critical flags is set the programmer will automatically enter LOADER COMMAND MODE. This generally indicates the user code space is empty.

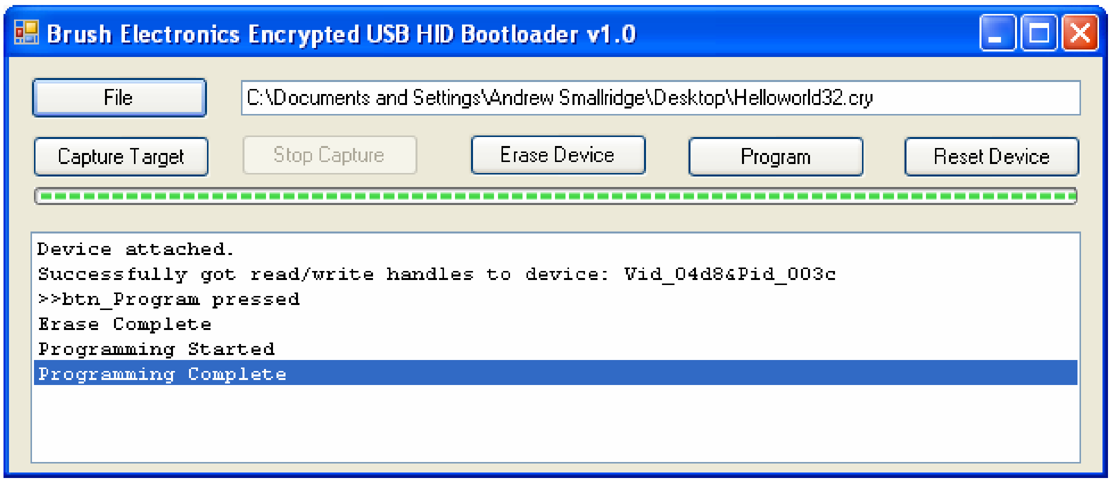
If neither of the critical flags has been set, the loader will wait for five seconds looking for a command from the Programmer via the serial interface to put the Bootloader into LOADER COMMAND MODE. If no command is received the Bootloader passes control to the user's application code via the remapped reset vector. Note that the Bootloader and the user's code do not operate concurrently – this explains why the loader does not require any of the PICs resources other than the consumed program memory.

The Bootloader accepts commands from the Programmer via a USD interface..

The Bootloader implements an incremental programming mechanism. This means that it will program only the bytes specifically contained in the record to be programmed. Code changes down to a single byte granularity are supported and the remaining program memory space contents are preserved.

Using the programming application

The following image is a screenshot of the Encrypted USB HID Bootloader Utility. The Encrypted file to be program is shown selected in the File window. Encrypted files have the extension .cry as per the example.



C:\Documents and Settings\Andrew Smallridge\Desktop\Helloworld32.cry

```
Device attached.
Successfully got read/write handles to device: Vid_04d8&Pid_003c
>>btn_Program pressed
Erase Complete
Programming Started
Programming Complete
```

The PIC bootloader firmware upon reset, enters bootloader mode for approximately 7 seconds. If the ***Capture Target*** button is clicked during this time the bootloader remains in bootloader mode and executes the selected command. The bootloader is captive in bootloader mode until the PIC is reset either via the ***Reset Device*** button or by resetting the PIC. When the bootloader has been programmed into the PIC and there is no user application present in the PIC, then the PIC remains in bootloader mode.

The ***Program Status*** information is displayed in the lower memo pane.

Steps for downloading code to a target via the Bootloader and Programmer Application:

Step 1 – Using the PC programmer application, open the Encrypted file to be programmed into the target. Note that to program the Bootloader from a file, you must first select the source file using the ***File*** button. Encrypted source files have the extension ".cry". The identified file and path will be displayed in the file window. The source file is opened when the ***Program*** button is clicked and closed at completion of the programming cycle. The next time the ***Program*** button is clicked the source file is again opened. This is important because it means that the Bootloader is always being programmed with the current contents of the source file.

Step 2 – Click the ***Capture Target*** button

Step 3 – Apply power to the target system or reset the PIC on the target system.

Step 4 – Wait 2 to 3 seconds for the USB bus to enumerate. The programmer application will capture the Bootloader.

Step 5 – Select the desired operation via the ***Erase Device*** and/or ***Program*** buttons.

Step 6 – Once the target has been successfully programmed, click the ***Reset Device*** button which will issue a LOADER RESET COMMAND to the Bootloader.

When the LOADER RESET COMMAND is executed the target executes the Bootloader code and waits for approximately 7 seconds to be captured. If not captured and the critical flags are clear then the loader passes control to the user's application program.

Encrypter Application

The XTEA.EXE Encrypter application is a Windows console application that accepts either one or three command line arguments. These arguments are the source file name of the standard Intel hex file to be encrypted, the 16 byte *Cipher Key* and the number of *Iterations* the cipher should be applied to the cipher text. The *Cipher key* must be exactly 16 bytes and must match the hard coded *XTEA_Key* in the Bootloader source code. The *Iterations* (typically 16) must match the hard coded *XTEA_Iterations* in the Bootloader source code. If only the source filename is specified on the command line then the XTEA's applications hard coded *XTEA_Key* and *XTEA_Iterations* constants are used.

The Encrypter generates the encrypted output file using the same filename as the source substituting *.cry* for the file extension.

Usage: XTEA sample.hex

Usage: XTEA sample.hex 123helloworld321 16

Programmer Application
The commands available via the Programmer's GUI interface are self explanatory and further information can be found in the source code for the Programmer Application. The lower 4 command buttons are not used by the bootloader but are included in the user interface to enable developers to add additional functionality – for example, to be able to modify specific region for use with parameter tables – access to these regions would typically not be encrypted.

The Programmer and Bootloader are implemented in a client / server arrangement. The Programmer (client) issues commands to the Bootloader (server) which executes the commands and returns status information for each command. The Programmer application is multithreaded, implementing a read thread for processing packets received from the Bootloader. Packets sent to the Bootloader are handled by the main thread. The Programmer Application uses a timer control as part of the error detection and processing mechanism for the various ERASE and PROGRAM commands.

The Programmer Application is written in Microsoft Visual C++ 2008 .

Customization
The bootloader must be customized to support different hardware plaforms (targets) and microcontrollers. In general the customisation requires the following steps:

- Define target hardware platform in the platform.h file
- In the Microchip IDE, select the processor in the the menu Configure / Select Device
- Create the processor specific bootloader linker script for compiling the bootloader. Refer to the sample bootloader linker scripts supplied with the bootloader package which contains instructions for modifying a standard linker script
- Create the processor specific bootloader linker script for compiling the user application to coreside with the bootloader. Refer to the sample application linker scripts supplied with the bootloader package which contains instructions for modifying a standard linker script
- Modify the bootloader's main source file to specify the target specific fuses

Limitations of the Bootloader

The following limitations of the Bootloader must be taken into account:

- The target's registers are not preserved by a RESET
- No support for WDT fuse bit. WDT support if required must be implemented by the software enabled WDT feature
- The Bootloader ignores Configuration Records and ID records

Need Something Special?

What if you need some unique feature added to the Bootloader or a Bootloader developed for some other product? Brush Electronics specializes in the development of Bootloaders for Microchip Microcontrollers and welcome the opportunity to work with you to develop a custom product that meets your specific needs.

Brush Electronics
2 Brush Court
Canning Vale
Western Australia 6155
Australia

Tel: +61 (0) 894676358
Email: info@brushelectronics.com
www: www.brushelectronics.com
