

Encrypted Serial Bootloader System

© 2003-2010 Andrew Smallridge

asmallri@brushelectronics.com

www.brushelectronics.com

Brush Electronics' Encrypted Serial Bootloaders have been developed to support remote firmware upgrade for the Microchip PIC base Microcontroller systems deployed in the field. The Encrypted Serial 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 Serial 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 Serial Bootloaders are available for the Microchip PIC18F, PIC24, dsPIC33 and PIC32 families of Microcontrollers.

Key attributes of Brush Electronics' Encrypted Serial Bootloaders:

- Ability to bootload a remote system
- 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 PIC18, PIC24/dsPIC33 and PIC32 versions are 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. For development environments that do not use linker scripts, such as the CCS PCH compiler, build directives are used to reserve program memory used by the bootloader.
- 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, user configurable optional parameters are also stored in this space. For the PIC18F and PIC24/dsPIC33 versions, the Bootloader

transparently uses the PIC's reset vector, that is, the lower 8 bytes of program memory. The user's reset vector is transparently mapped into a program memory block in the Bootloader's code space into the region referred to as the Bootloader parameter block. The location of the bootloader parameter block is defined in the bootloader's linker script. For the PIC18 and PIC24/dsPIC33 versions, this parameter block is the *userjump* memory region. For the PIC32 version, this parameter block is the *bldr_param_mem* memory region.

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 C18, C30 and C32 compilers, when used with their respective standard linker scripts, automatically insert the GOTO instruction. For other compilers, such as the CCS PCH compiler, build directives are used to achieve the same result.

The PIC18F version of the bootloader, currently available for the CCS PCH compiler, is located in low program memory from 0xF0 to approximately 0x1CFF. The PIC24/dsPIC33 version of the Bootloader is located in low program memory from 0x400 to approximately 0x2420. The PIC32 version of the Bootloader is located in low program memory from 0x9D00_0000 to approximately 0x9D00_BFFF. 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. For the PIC32 bootloader version, 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 application'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 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 PIC's resources other than the consumed program memory.

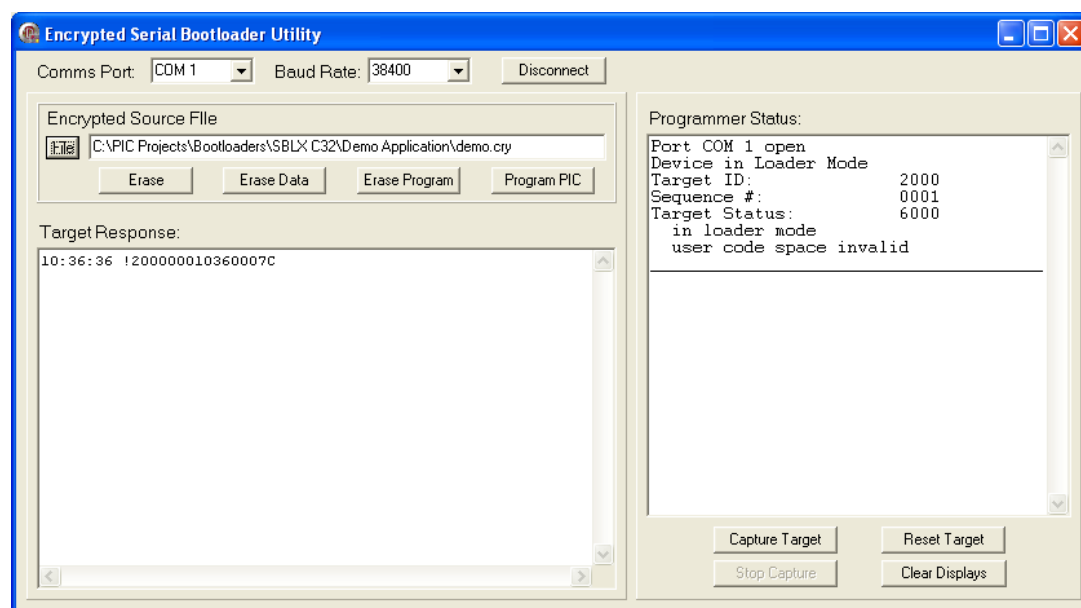
The Bootloader accepts commands from the Programmer via a serial interface. Serial communications, especially in noisy environments or as a result of long communications paths, can be unreliable resulting in lost or corrupted characters therefore the programmer application, in conjunction with the Bootloader firmware, implements error detection, reporting and, where applicable, retransmission.

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 Serial Bootloader Utility. The Target's Bootloader has been captured and the encrypted hex file demo.cry has been selected for programming into the target via the Encrypted Serial Bootloader.

The *Program Status* information is displayed in the right hand side memo pane. The *Target Response* memo pane shows the time the last message was received from the Bootloader as a response to control and data messages from the Encrypted Serial Bootloader Utility.



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

Step 1 – Reset the Target system.

Step 2 – Within 3 seconds of Step 1, Click **Capture Target** to initiate device discovery and capture. The programmer application will send a capture message to the Bootloader and wait for an acknowledgement from the Bootloader indicating it has been captured. This sending of the capture message will repeat every 500ms (retry interval) until the Bootloader responds to the command or until the **Stop Capture** button is operated.

Step 3 – Select the desired erase or program operation. 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 4 – Once the target has been successfully programmed, click the **Reset Target** 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 5 seconds to receive the capture command. If this command is not received 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 *Capture Target* command (Loader Mode) captures the Bootloader to prevent the Bootloader passing control to the user's application program at the end of the initial 5 second period from reset. In this mode the Bootloader is under the control of the Programmer application. If the user's code space is valid and the target is reset while in this mode then after 5 seconds control will once again be passed to the user's program.

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.

All commands from the Programmer application to the Bootloader will result in a status packet being sent from the Bootloader to the Programmer Application indicating the success or otherwise of the command. In the event that no status is returned by the Bootloader for the current record, the timer will expire and a consecutive error counter variable will be incremented. If this is consecutive error counter value below the maximum consecutive error threshold then the timer is restarted and the previously saved current record is retransmitted.

The Programmer Application is written in Delphi for the windows environment. The application uses the freeware Ararat Synapse SynaSer serial library. The serial library can be downloaded from <http://synapse.ararat.cz/doku.php/download>

Customization

The bootloader must be customized to support different hardware platforms (targets) and microcontrollers. In general the customization 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 coexist 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
- User's program must execute a GOTO instruction within the first four instructions (PIC24/dsPIC33 versions only)
- 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
