Brush Electronics

2 Brush Court, Canning Vale, WA 6155, Australia • Tel: +61 894676358 • info@brushelectronic.com
ABN 50087977811

# Debugging Your First Bootloader Port

Andrew Smallridge

This document examines the process for debugging the port of a Brush Electronics bootloader. Brush Electronics produce a range of bootloaders for Microchip PIC18F, PIC24/dsPIC33 and PIC32 Microcontrollers. Although the programming language and memory layout varies between implementations, the basic structure is the same. From the bootloader's perspective, the program memory is divided into the following sections:

- Reset Vector (located at the restart address of 0x0)
- Processor Interrupt Vector Table(s) (part of User Program Memory)
- Bootloader code space
- Bootloader parameter block
- User Program Memory

In general our bootloaders do not use interrupts and, where appropriate, leave sufficient program memory around the Interrupt Vector Tables to enable common small interrupt handler to be implemented without requiring an addition jump to the interrupt handler. This can be an important consideration for interrupt handlers that are very time critical where the application behaviour must be the same with or without the presence of the bootloader.

The bootloader code space contains the executable body of the bootloader. Several Microcontrollers include a dedicated boot block for use with bootloaders. However, in general, our bootloaders do not fit into these dedicated blocks due to the various encryption, error recover techniques and additional features we employ in our bootloaders which make them too large to fit in these dedicated boot blocks. Therefore our bootloaders tend to be located in standard program memory space. The bootloader include program logic to prevent a user application from overlaying the bootloader during the bootload process.

The bootloader parameter block contains bootloader status information used to identify if a user application is present in program memory. The bootloader uses the parameter block in such a way that if the parameter block was erased, the bootloader would see consider the user application to not be present. This parameter block is typically located on the first PIC erase size boundary after the bootloader body. For example, consider a PIC18F97J60. This PIC performs an erase of program memory in 1K chucks. Therefore if the bootloader body was to finish at the address 0xB123 then the parameter block would be located on the next erase size boundary at the address 0xB400.

 

The User Program Memory is all the flash memory with the exception of the reset vector, the bootloader code space and the bootloader parameter block.

When the bootloader performs an erase operation it erases all the User Program Memory space as well as the bootloader parameter block. After an erase operation, the bootloader, depending on the PIC family, may pre-populate the interrupt vector table.

When porting a bootloader to a new hardware platform and/or processor variant, it is necessary to built linker scripts for the bootloader as well as a linker script for the application. The bootloader linker script is used to define the location of the bootloader parameter block. The application's linker script reserves the program memory used by the bootloader. For our PIC18F and PIC24/dsPIC bootloaders the bootloader determines the application entry point from the application reset vector record therefore the application linker script can define an origin anywhere in the User Program Memory space.

For our PIC32 bootloaders, the bootloader is hardcoded with the applications entry point therefore the application linker script and the configuration section of the bootloader, usually located in the platform.h file, have the same user application address information. In the linker script this is the **`exception_mem`** parameter and in the platform.h file this is the **`USER_BASE`** parameter.

## Test Procedure

The following test procedure requires code protection to be disabled in the application and the bootloader. Once the operation of the bootloader has been validated the code protection fuses can be enabled. A common mistake when developing applications that co-exist with bootloaders it the mis-configuration of the config fuses. Our bootloader ignore the config settings in the user application hex file. This prevents a bug in the application config fuse settings from rendering the bootloader inoperable. Ensure you have configured the bootloader fuse settings to match the target hardware requirements. Then **COPY the bootloaders config fuse settings to the user application**.

1. Compile the user application without the application linker script that was developed to allow the application to coexist with the bootloader.

2. Install and test the user application. If the application does not run correctly then debug the application and repeat go to step 1.

3. Compile the user application using the application linker script that was developed to allow the application to coexist with the bootloader. Let's call the resultant hex file "**output.hex**". If the application does not run correctly then debug the application liker script and repeat this step

4. Using MPLAB, read the contents of PIC and then, using the EXPORT command from the file menu, export the contents of program memory to the file **app.hex** leaving the default options selected.

 

5. Compile the bootloader and install the bootloader onto the target hardware platform. Using MPLAB, read the contents of PIC and then, using the EXPORT command from the file menu, export the contents of program memory to the file **ldr.hex** leaving the default options selected.

6. If the bootloader is an encrypted bootloader then encrypt the **output.hex** file from step 3 to produce the **output.cry** file

7. Use the bootloader to load the appropriate output file. For non encrypted bootloaders this will be the **output.hex** file produced at step 3. For encrypted bootloaders this will be the **output.cry** file produced at step 6. Using MPLAB, read the contents of PIC and then, using the EXPORT command from the file menu, export the contents of program memory to the file **ldr-app.hex** leaving the default options selected.

8. Using a file compare program, such as the freeware ExamDiff application (www.prestosoft.com/edp_examdiff.asp), compare the ldr.hex and ldr-app.hex files. The ldr.hex file should be fully contained in the ldr-app.hex file with the exception of the bootloader parameter block as this will now contain status information.

9. Using a file compare program, such as the freeware ExamDiff application (www.prestosoft.com/edp_examdiff.asp), compare the app.hex and ldr-app.hex files. The app.hex file should be fully contained in the ldr-app.hex file with the exception of the reset vector (excluding PIC32). The reset vector for the PIC18F and PIC24/dsPIC33 processors will be located in the bootloader parameter block.

Once the final step has been successfully validated the PIC contains valid bootloader and application images. If at this stage the application does not run correctly then the most likely cause of the problem is the incorrect initialization of I/O, registers or variables. When the bootlaoder exists the bootload process, registers and I/O do not match the power-on-reset defaults. The user application must not assume power on defaults.