Freescale Semiconductor
Application Note

Document Number: AN3927
Rev. 1, 02/2012

Freescale S08 USB Mass Storage Device Bootloader

by: Derek Snell
Field Applications Engineer

1 Introduction

Freescale offers a broad selection of microcontrollers with USB. Designing a product with USB allows for an easy interface to do field updates of the product's firmware. This application note describes a mass storage device (MSD) USB bootloader written to work with several Freescale USB families. A device with this bootloader is connected to a host computer, and the bootloader enumerates as a new drive. The new firmware is copied onto this drive, and the device reprograms itself.

Freescale does offer other bootloaders. For example, the application note titled *USB Bootloader for the MC9S08JM60* (document AN3561) is a different USB bootloader written for the Flexis JM family. This MSD bootloader is offered as another option with these advantages:

- It does not require a driver to be installed on the host

Contents

freescale™ semiconductor

- It does not require an application to run on the host
- Any user can use it with a little training. The only action required is to copy a file onto a drive.
- It can be used with many different host operating systems. It requires no host software or driver
- The bootloader and application are integrated in one project. The device needs to be programmed only once in production. Separate programming steps for the bootloader and application are not required.

This bootloader was specifically written for several families of Freescale microcontrollers that share similar USB peripherals. These families include, but are not limited to the following: The Flexis JM family MC9S08JM and MCF51JM, MC9S08JS, and the ColdFire MCF522xx parts with USB. This bootloader must work on all of these devices with few changes, examples were written and tested on the following:

Tested Freescale microcontrollers:

- MCF52259—32-bit ColdFire V2 with USB, Ethernet, CAN, and external bus
- MCF51JM128—32-bit ColdFire V1 with USB, part of the Flexis JM Family
- MC9S08JM60—8-bit S08 with USB, part of the Flexis JM Family

Tested development boards:

- M52259DEMO—Low-Cost development board for MCF5225x family
- DEMOJM—Low-Cost development board for Flexis JM family

Tested operating systems:

- Windows XP Pro with Service Pack 2
- Windows Vista business for 32-bit with Service Pack 1
- Fedora 8 with Linux kernel 2.6.26

2 Functional Description

2.1 General Overview

The bootloader is integrated with an application that performs the product's main functions. At reset, the bootloader is executed and does some simple checks to see if the application should start, or if the bootloader should start. If it enters bootloader mode, it uses USB to enumerate with the host computer. During this enumeration process, the device declares itself as an MSD. The host then creates a new drive in the system. You can then copy an S-record file onto the drive, and the device re-programs itself.

S-record files are common ASCII files used to specify the program data stored in devices. Freescale's software tool chain called CodeWarrior generates S-record files automatically when projects are compiled. These files have the extension .S19 and are referred to as S19 files.

After the S19 file has been transferred to the device and the device re-programs itself, the device re-enumerates with the host. A file is displayed in the drive that represents the status of the bootloader operation.

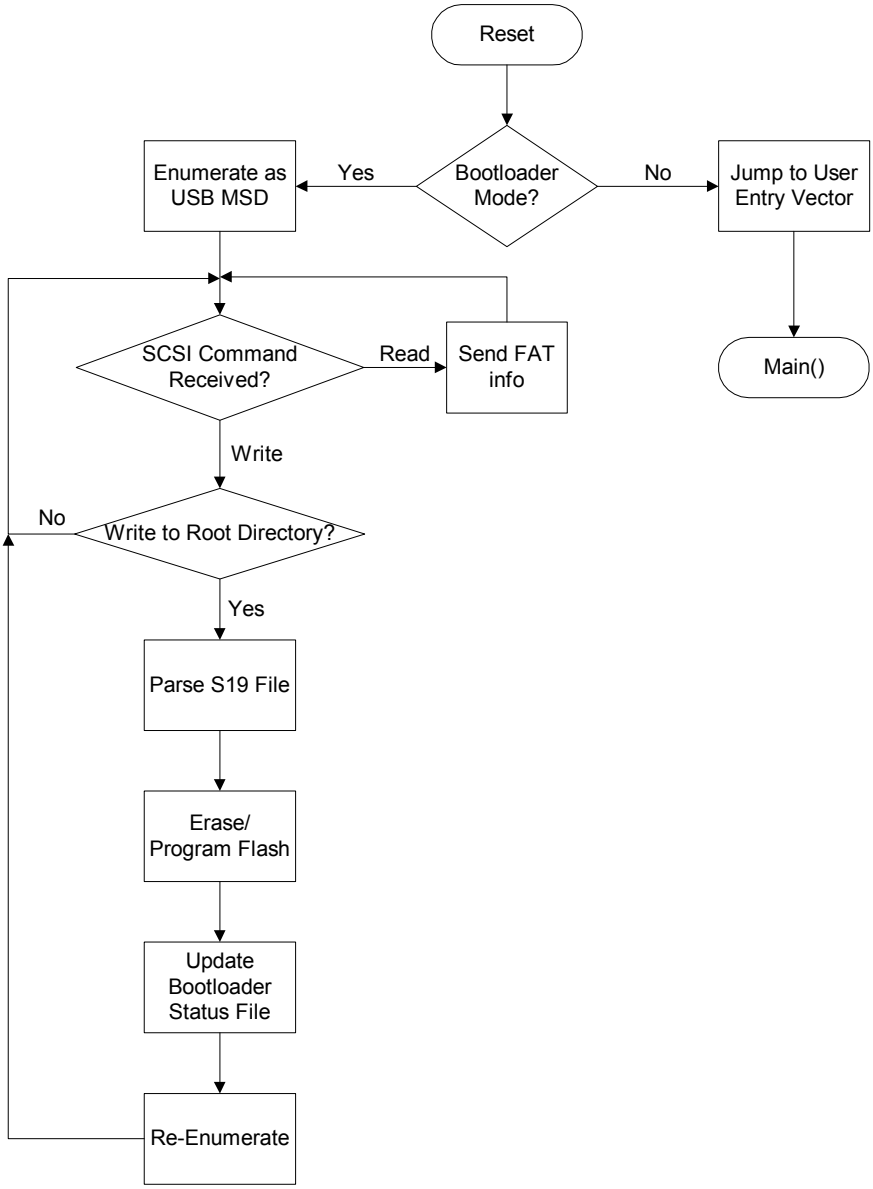

Figure 1. Bootloader functional flow chart

2.2 Bootloader Mode vs. Application Mode

The bootloader starts executing immediately after reset and determines what mode is entered. It is important for the bootloader to be in control after reset to allow the bootloader to run if the application is not present, has been erased, or has become corrupted. Normally, the bootloader attempts to enter application mode. It does this through a user entry vector. The user entry vector is a jump vector at a specified absolute address that jumps to the entry point of the application. This user entry vector is used to prevent cross-calling (see Section 4.1, “Prevent Cross-Calling,” on page 16). The address of the application entry point might change with each compile. But the address of the user entry vector is always the same. Therefore, the bootloader is programmed to always jump to the absolute address of the user entry vector that then jumps to the application entry point.

Before entering application mode, the bootloader performs some simple checks to ensure the application is present. It checks that the user entry vector includes the JUMP op-code. It also checks that the jump address in that vector is not erased. If the user entry vector is a good jump vector, then the bootloader jumps to the application. If the vector is not good, then the bootloader loads to allow an application update.

The bootloader mode can also be forced during a reset. The included examples monitor a button, and if the button is pressed during reset the bootloader mode is entered regardless of the user entry vector status. The button used to enter this mode depends on the development board; see Section 4.2, "Using Bootloader on the Demo Board," on page 18 for the specific button. Many applications need to change the method used to force bootloader mode. This can be executed by modifying the usr_entry_xx.c file.

2.3 USB Enumeration

After the bootloader enters bootloader mode, it enables the USB peripheral to communicate with the host computer. This starts the USB enumeration process. For more details on enumeration, please refer to application note titled *USB and Using the CMX USB Stack* (document AN3492). The USB stack used by this bootloader is Freescale's USB Mini-Stack, which has a very small memory footprint. The device sends the host descriptors that declare it as a mass storage device. The host loads the device as a new drive, and uses bulk transfers to communicate with the device. These bulk transfers send SCSI commands to talk to what the host thinks is a removable storage drive.

2.4 Pseudo-FAT

After the host thinks the bootloader is a storage drive, it attempts to read the file allocation table (FAT) of the drive, and reads the drive's contents. The bootloader responds to these queries with information that looks like a 1 GB drive. The drive has the volume name of BOOTLOADER. It looks like it has no directories and has a file called READY.TXT in the root directory. The bootloader does not use a true file system, and is only capable of acting like a drive to the host. It also accepts files to be written to the drive to transfer the S19 file. However, because the bootloader is only a Pseudo-FAT system, it does not update the FAT while the host attempts to change it. The FAT is always fixed, and the only change is the name of the file in the root directory that specifies the status of the bootloader. When the host writes data to the drive, the bootloader ignores all data that is not written to the root directory of the drive. When file data is written to the root directory, the bootloader assumes it is an S19 file. It starts the S-record parser to get the new firmware update. Only one file can be transferred to the drive after reset. After a file is transferred, the bootloader needs to restart to accept another file.

2.5 S-Record Parser

The bootloader includes an S-record parser to allow raw S19 files to be sent to the bootloader. After a file has been transferred to the bootloader, it assumes it is an S19 file and starts to parse it. Each S-record is checked to ensure it is a valid S-record with a correct checksum. The parser compares the S-record address to check if it is in the application space. It checks all S-records, but ignores the data in S-records outside of the application flash address range. After the first application S-record is successfully checked, the parser erases the entire application flash. Then, it programs each S-record after it is checked. The parser looks for a final S-record of type 7, 8, or 9 to know that the file transfer is complete. If the final S-record

is received, all S-records were valid, and all flash operations were successful, the parser returns with success. If at any point an S-record was invalid, the parser returns with the address of the invalid S-record.

NOTE

This parser can also be used by other bootloaders or applications that do not use USB. If a different communication is desired, the function GetUSBchar() can be replaced with a new function that receives a character through a different interface, like a UART or CAN.

2.6 Re-Enumeration

After the S-record parser finishes, the device re-enumerates with the host to update the status file on the drive. The status files are discussed in Section 2.7, "Bootloader Status," on page 5. Within the host computer, the drive disappears as the device drops off the USB bus. After a few seconds, the device re-enumerates and the drive re-appears. You can read the status file on the drive to see the status of the bootloader.

The bootloader re-enumerates because some host operating systems cache the file data on the storage drive, and do not refresh. For example, if using Fedora 8 with this bootloader, the operating system (OS) does not write the file to the device until the drive is unmounted. After unmounting, the device receives the S19 file, updates the firmware, and then re-enumerates to display the status. In Windows XP and Vista, the OS caches the drives FAT without re-reading it, and the status file is not updated. Re-enumeration forces the host to re-read the FAT and display the bootloader status file.

2.7 Bootloader Status

The bootloader uses the file system drive in the host computer to display the status. With this method, the bootloader status can always be seen regardless of the display or LEDs available on the device's hardware. In the case of an S-record error, the address of the error is also displayed. The status is displayed in the filename in the root directory of the drive. These files are empty and have no data. Before a file is transferred to the bootloader, the user must check the status file is READY.TXT. After the file transfer is complete, and the bootloader re-enumerates, the user must check the status file to see if it was a success. The status filenames are listed below:

- READY.TXT—Shows the bootloader is ready to receive an S19 file
- SUCCESS.TXT—Shows the bootloader completed a firmware update successfully.
- FFAILED.TXT—Shows the bootloader had a flash erase or program error. This error is triggered if the flash erase or program routine returns with an error. This error is usually triggered because the flash clock is not at the appropriate frequency, or an invalid address is used. If the address is not a flash address or is protected, this error occurs. To diagnose, place a breakpoint where this error is generated in ParseS19.c and run the debugger to find where the error occurs.
- SFxxxxxx.TXT—Shows the bootloader S-record parser found an error in an S-record. The parser checks each S-record to ensure it has a valid format and the checksum matches. If not, this error is generated. This error can also occur if the S-record specifies an invalid address. The address of the improper S-record appears in the filename. For example, if the S-record for address 0x1234 was

invalid, the status filename would be SF001234.TXT. If this error occurs, check the S-record for proper format, checksum, and address.

- STARTED.TXT—Shows a file has been received with data for the root directory and the S-record parser has started. Normally, this status should not be visible to the user because the drive only re-enumerates after a success or failure. If seen, debug the bootloader to learn why.

2.8 Flash Protection

A bootloader must always be protected from erasing itself or getting corrupted. With this protection enabled, the worst case scenario during a botched firmware update is that the application is erased or corrupted, but the bootloader is still in tact. The bootloader can then run again and re-load the new application.

This bootloader uses Freescale's flash protection features. Depending on the part used, the microcontroller has a non-volatile register that specifies what flash sectors are protected. The bootloader is stored in these protected sectors and the application is stored in the unprotected sectors. These non-volatile register settings are part of the bootloader and cannot be changed by the application. In addition, this flash protection also protects the reset and interrupt vectors from being changed by the application.

Here is a list of the flash protection registers by the core, the sectors protected, and the bootloader file that loads these registers. Refer to Table 4 to see how this affects the flash memory usage.

- ColdFire V2—Register CFMPROT is set to 0x1 and protects the flash address range 0x0 to 0x3FFF. This register is set in file mcf5225x_vectors.s.
- ColdFire V1—Register NVPROT is set to 0xF7, and protects the flash address range 0x0 to 0x1FFF. This register is set in the file Bootloader_V1.c.
- MC9S08—Register NVPROT is set to 0xEA, and protects the flash address range 0xEC00 to 0xFFFF. This register is set in the file Bootloader_S08.c.

NOTE

If the bootloader is modified, these flash protected sectors can be increased or decreased to properly protect the bootloader. If the address range is changed, change the above registers, and also change the memory map in the linker file and in Bootloader_xx.h.

2.9 Interrupt Vector Table Re-Direction

The application cannot change the default interrupt vector table of the microcontroller because that flash memory is protected. Therefore, the bootloader re-directs the interrupt vectors into the application space that allows the application to update its interrupt vectors. Because the bootloader re-directs the interrupt vectors, this also means the bootloader cannot use interrupts. The bootloader is implemented by polling the USB interrupt flag to see when a packet was sent or received.

The bootloader was written with each core using a different method for interrupt re-direction, and is further explained with examples in Section 4.3, "Creating a New Project with the Bootloader," on page 20.

2.10 Stack and RAM Usage

The bootloader interacts with the application only when it starts the application by jumping to the user entry vector. The application never cross-calls functions into the bootloader. Therefore, the bootloader and application do not need to have separate RAM memory. They can both use the same physical RAM. From the application's perspective, the bootloader uses no RAM, this is because the application has access to the entire RAM in the device. The linker files are setup to share the physical RAM between the bootloader and the application. They also share the stack, so the location and size of the stack are the same for both the bootloader and application.

2.11 Bootloader Memory Maps

The following sections show the memory maps for the three different device examples. The memory map for both the bootloader and application are shown to see how the RAM overlaps. Notice the application has access to all RAM and overlaps the bootloader RAM. All other memory sections are identical between the two.

2.11.1 ColdFire V2 Bootloader Memory Maps

Table 1. MCF52259 bootloader memory map

Addresses	Bootloader	Application
0x0000_0000 to 0x0000_03FF	Interrupt and exception vectors	Interrupt and exception vectors
0x0000_0400 to 0x0000_0417	Flash protection and security registers	Flash protection and security registers
0x0000_0420 to 0x0000_3FFF	Bootloader flash (15 kB)	Bootloader flash (15 kB)
0x0000_4000 to 0x0000_400F	User entry jump vector	User entry jump vector
0x0000_4010 to 0x0007_FFFF	Application flash memory (496 kB	Application flash memory (496 kB
0x0008_0000 to 0x1FFF_FFFF	Reserved	Reserved
0x2000_0000 to 0x2000_03FF	Re-directed interrupt vector table in RAM	Re-directed interrupt vector table in RAM
0x2000_0400 to 0x2000_05FF	Bootloader USB BDTs and buffers	RAM available for application (63 kB)
0x2000_0600 to 0x2000_F7FF	RAM available for bootloader	
0x2000_F800 to 0x2000_FFFF	Stack (2 kB)	Stack (2 kB)

2.11.2 ColdFire V1 Bootloader Memory Maps

Table 2. MCF51JM128 Bootloader memory map

Addresses	Bootloader	Application
0x(00)00_0000 to 0x(00)00_03FF	Interrupt and exception vectors	Interrupt and exception vectors
0x(00)00_0400 to 0x(00)00_040F	Flash protection and security registers	Flash protection and security registers
0x(00)00_0410 to 0x(00)00_1FFF	Bootloader flash (7kB)	Bootloader flash (7kB)
0x(00)00_2000 to 0x(00)00_21BF	Application interrupt vector table	Application interrupt vector table
0x(00)00_21C0 to 0x(00)00_21C7	User entry jump vector	User entry jump vector
0x(00)00_21C8 to 0x(00)00_21FF	Unused	Unused
0x(00)00_2200 to 0x(00)01_FFFF	Application flash memory (119 kB)	Application flash memory (119 kB)
0x(00)20_0000 to 0x(00)7F_FFFF	Reserved	Reserved
0x(00)80_0000 to 0x(00)80_01FF	Re-Directed interrupt vector table in RAM	Re-Directed interrupt vector table in RAM
0x(00)80_0200 to 0x(00)80_06DF	Bootloader USB BDTs and buffers	RAM available for application (14.5 kB)
0x(00)80_06E0 to 0x(00)80_3BFF	RAM available for bootloader	
0x(00)80_3C00 to 0x(00)80_3FFF	Stack (1 kB)	Stack (1 kB)

2.11.3 MC9S08 Bootloader Memory Maps

Table 3. MC9S08JM60 bootloader memory map

Addresses	Bootloader	Application
0x0000 to 0x00AF	Direct page registers	Direct page registers
0x00B0 to 0x00FF	Zero page RAM	Zero page RAM
0x0100 to 0x105F	RAM available for bootloader	RAM available for application (3.8 kB)
0x1060 to 0x10AF	Stack (80 B)	Stack (80 B)
0x10B0 to 0x17FF	Application flash memory1 (1872 B)	Application flash memory1 (1872 B)
0x1800 to 0x185F	High page registers	High page registers
0x1860 to 0x195F	Bootloader USB BDTs and buffers	Application USB BDTs and buffers
0x1960 to 0xEBA5	Application flash memory2 (52.5 kB)	Application flash memory2 (52.5 kB)
0xEBA6 to 0xEBFC	Application interrupt vector table	Application interrupt vector table
0xEBFD to 0xEBFF	User entry jump vector	User entry jump vector
0xEC00 to 0xFFAD	Bootloader flash (5 kB)	Bootloader flash (5 kB)
0xFFAE to 0xFFBF	Flash protection and security registers	Flash protection and security registers
0xFFC0 to 0xFFFF	Interrupt vectors	Interrupt vectors

2.12 Resource Usage

Table 4 shows the memory resource usage of the bootloader in the three different device examples. As discussed in Section 2.8, "Flash Protection," on page 6, the bootloader is protected in flash. The minimum flash protection size was selected to protect the bootloader. Therefore, the amount of flash used by the bootloader is dictated by the minimum protection size available in the device.

Table 4. Bootloader resource usage

Device	Flash Usage (kBytes)	Secured Flash (kBytes)	RAM Usage[1] (Bytes)
MCF52259	9	16	0
MCF51JM128	7	8	0
MC9S08JM60	5	5	0

[1] See Section 2.10, "Stack and RAM Usage," on page 7 for explanation why the bootloader uses no RAM

3 CodeWarrior Project Structure

The following sections describe the provided firmware source code for the bootloader and application examples. The source code included with this application note includes the bootloader and examples for all three cores.

3.1 Bootloader File Structure

Figure 2 shows the directory structure of the provided source code:

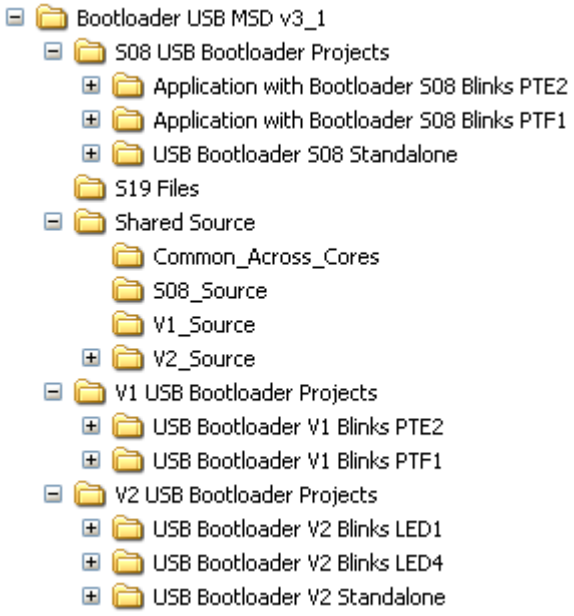

Figure 2. Bootloader file Structure

MC9S08—Three projects are included. The project "USB Bootloader S08 Standalone" is the bootloader itself. The two application projects are examples that use the bootloader and blink two different LEDs on the DEMOJM board. When building both these applications the bootloader S19 file is included using the HEXFILE command in the Project.prm file. The S19 files from these two applications can be used to demonstrate that the bootloader changes the application firmware.

S19 Files—Is a collection of the S19 files from all the application examples. It also includes the S19 files generated by the CMX example instructions in Section 4, "Using the Bootloader," on page 16. The collection of all the S19 files in this directory makes it convenient to test the bootloader demo by copying the files to the bootloader.

Shared Source—Is a collection of source files shared between multiple projects. The sub-directory **Common_Across_Cores** has the higher level source code for the bootloader that does not change among the three cores. The lower level source files that are specific to the core are in the other sub-directories.

ColdFire V1—Two projects are included. The ColdFire V1 does not have a stand-alone bootloader project. The bootloader source files are directly included with the application project. The two application projects are examples that use the bootloader and blink two different LEDs on the DEMOJM board. The S19 files from these two applications can be used to demonstrate that the bootloader changes the application firmware.

ColdFire V2—Three projects are included. The project **USB Bootloader V2 Standalone** is the bootloader itself, and generates a library file. The two application projects are examples that use the bootloader and blink two different LEDs on the M52259DEMO board. When building the project both of these applications include the bootloader library file. The S19 files from these two applications can be used to demonstrate that the bootloader changes the application firmware.

3.2 ColdFire V2 Project and Files

The ColdFire V2 bootloader has separate projects for the bootloader and application. The bootloader project is in the **USB Bootloader V2 Standalone** directory. It compiles the bootloader source files into a library file called V2_Bootloader.lib. This library file is included in the application project to integrate the bootloader with the application. Figure 3 shows the project window in CodeWarrior for this project. Each file is described below.

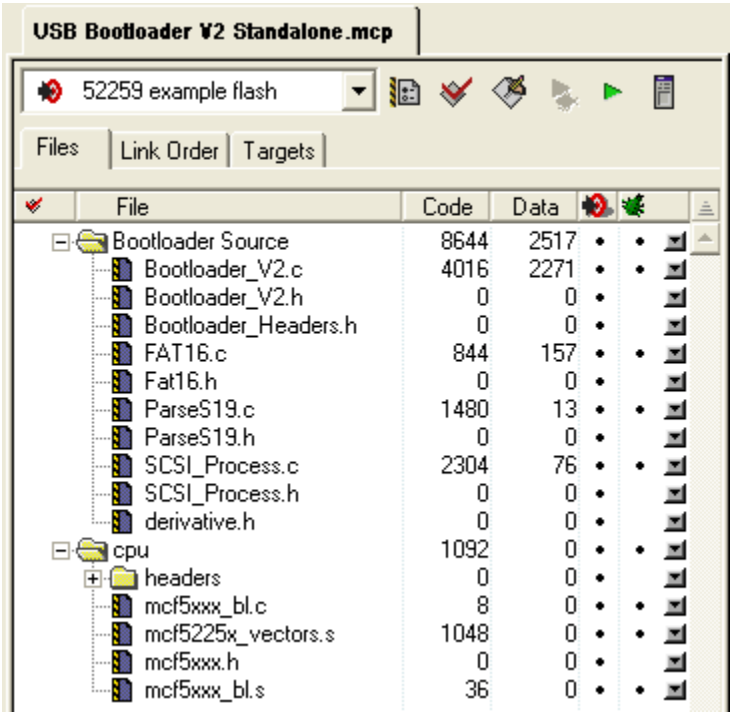

Figure 3. V2 Standalone bootloader project files

- Bootloader_V2.c—Part-specific file, contains USB stack, manages enumeration, copies USB packets to buffers, and manages flash programming algorithms. Also contains the main bootloader function after bootloader mode is entered.
- Bootloader_V2.h—Part-specific file contains macros for parts of the memory map, macros for the USB and bootloader routines, and structures for the USB peripheral
- Bootloader_Headers.h—Part-specific file points the shared source files to the part-specific header files.
- Fat16.c—Shared file that responds to the host to look like a file allocation table for the drive.
- Fat16.h—Shared header file for Fat16.c
- ParseS19.c—Shared file that parses the S-record file. It verifies each S-record in the file, and after verified, copies the S-record to the specified location in the flash.
- ParseS19.h—Shared header file for ParseS19.c.
- SCSI_Process.c—Shared file that manages SCSI commands from host.
- SCSI_Process.h—Shared header file for SCSI_Process.c.

- Derivative.h—Part-specific header used to make the shared source compatible with ColdFire V2 bootloader.
- mcf5xxx_bl.c—Part-specific file for exception handler
- mcf5225x_vectors.s—Part-specific file with reset and interrupt vectors. Has the flash protection and security register settings.
- mcf5xxx.h—Part-specific header file for the bootloader
- mcf5xxx_bl.s—Part-specific file for the exception handler.

There are two application examples included that blink different LEDs on the M52259DEMO board. Figure 4 shows the project window of the CodeWarrior project in the directory **USB Bootloader V2 Blinks LED1**. Below is a description of the files included with this project.

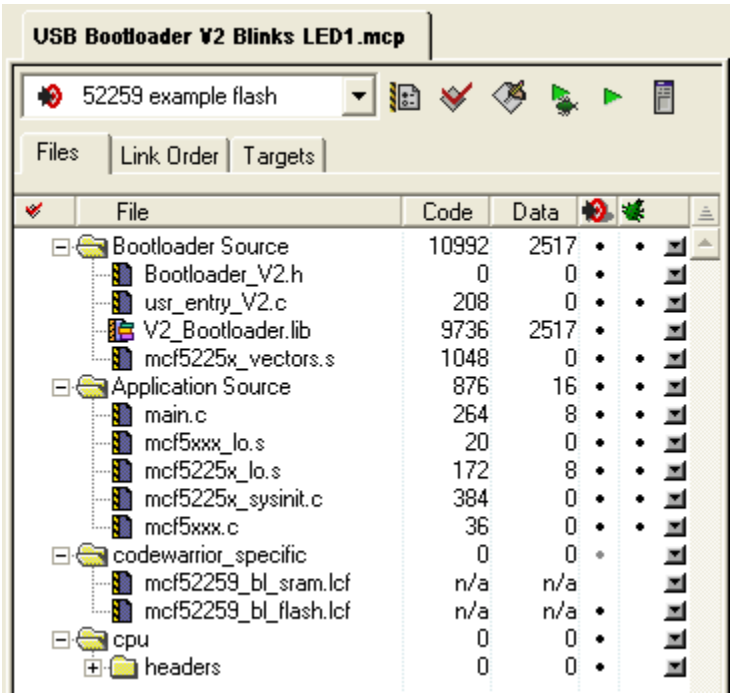

Figure 4. ColdFire V2 application with bootloader files

- Bootloader_V2.h—Part-specific file contains macros for the device's memory map, macros for the USB and bootloader routines, and structures for the USB peripheral
- usr_entry_v2.c—Part-specific file has the reset entry point and determines whether to enter application or bootloader mode. Required for the bootloader.
- V2_Bootloader.lib—Bootloader library file. Output file from the "USB Bootloader V2 Standalone" project.
- mcf5225x_vectors.s—Part-specific file with reset and interrupt vectors. It has the flash protection and security register settings. Required for the bootloader.
- main.c—Part-specific main file for application. Contains user entry jump vector. Interrupt vectors are initialized in RAM in main().
- mcf5xxx_lo.s—Project-specific, start-up files for application
- mcf5225x_lo.s—Project-specific, start-up files for application

- mcf5225x_sysinit.c—Project-specific, start-up files for application
- mcf5xxx.c—Project-specific, start-up files for application
- mcf52259_bl_sram.lcf—Part-specific linker command file used to load the bootloader and application into SRAM. Enables quick debugging of the application without requiring flash programming.
- mcf52259_bl_flash.lcf—Part-specific linker command file used to store bootloader routines in the protected flash, and application code in the unprotected flash.

3.3 ColdFire V1 Project and Files

The ColdFire V1 bootloader has combined projects for the bootloader and application. There are two application examples included that blink different LEDs on the DEMOJM board. Figure 5 shows the project window of the CodeWarrior project in the directory **USB Bootloader V1 Blinks PTE2**. Below is a description of the files included with this project.

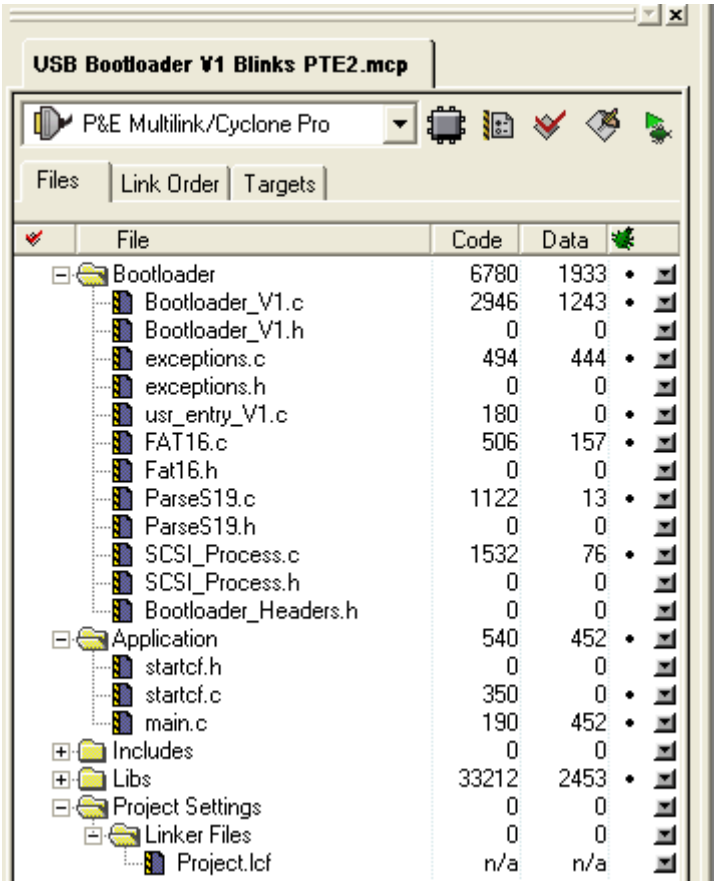

Figure 5. ColdFire V1 application with bootloader project files

- Bootloader_V1.c—Part-specific file, contains USB stack, manages enumeration, copies USB packets to buffers, and manages flash programming algorithms. Contains the main bootloader function after bootloader mode is entered.
- Bootloader_V1.h—Part-specific file contains macros for the device's memory map, macros for the USB and bootloader routines, and structures for the USB peripheral.

- usr_entry_v1.c—Part-specific file has the reset entry point, determines whether to enter application or bootloader mode.
- Fat16.c—Shared file that responds to the host to look like a file allocation table for the drive.
- Fat16.h—Shared header file for Fat16.c
- ParseS19.c—Shared file that parses the S-record file. It verifies each S-record in the file, and after verified, copies the S-record to the specified location in the flash.
- ParseS19.h—Shared header file for ParseS19.c.
- SCSI_Process.c—Shared file that manages SCSI commands from the host.
- SCSI_Process.h—Shared header file for SCSI_Process.c.
- Bootloader_Headers.h—Part-specific file points the shared source files to the part-specific header files.
- main.c—Project-specific main file for application. Contains user entry jump vector, copies application interrupt vector table to the re-directed interrupt vector table in RAM.
- Project.lcf—Part-specific linker command file used to store bootloader routines in the protected flash and the application code in unprotected flash.

3.4 MC9S08 Project and Files

The MC9S08 bootloader has separate projects for the bootloader and application. The bootloader project is in the **USB Bootloader S08 Standalone** directory. It compiles the bootloader source files into an S19 file called S08_Bootloader.abs.s19. This S19 file is linked into the application project. Figure 6 shows the project window in CodeWarrior for this project. Each file is described below.

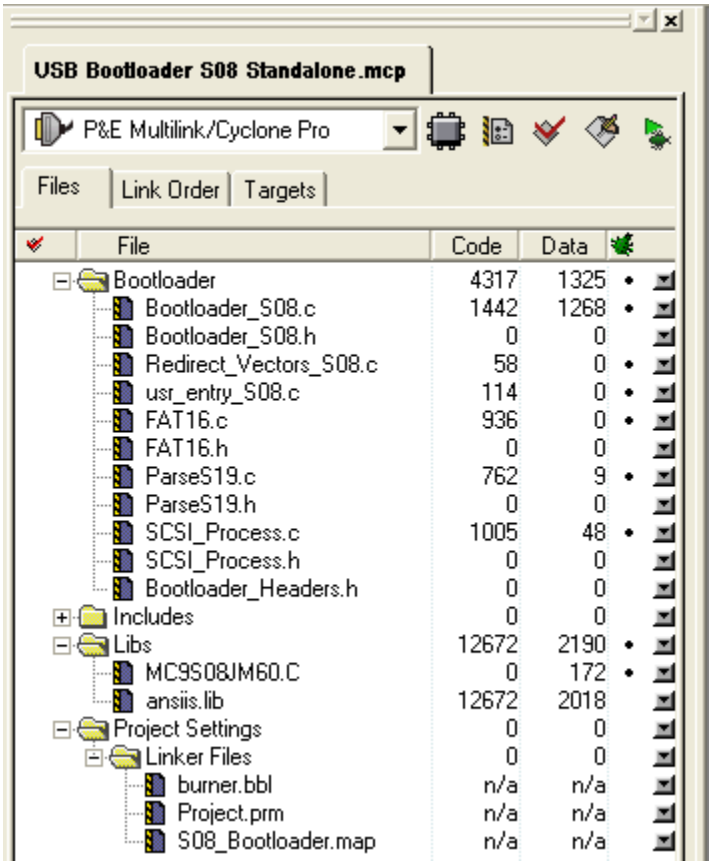

Figure 6. S08 Stand-alone bootloader project files

- Bootloader_S08.c—Part-specific file contains USB stack, manages enumeration, copies USB packets to buffers, and manages flash programming algorithms. Contains the main bootloader function after bootloader mode is entered.
- Bootloader_S08.h—Part-specific file contains macros for the device's memory map, macros for the USB and bootloader routines, and structures for the USB peripheral
- Redirect_Vectors_S08.c—Part-specific file contains interrupt vector table in the bootloader for part and points vectors to re-directed interrupt vector table in the application.
- usr_entry_S08.c—Part-specific file has the reset entry point and determines whether to enter application or bootloader mode
- Fat16.c—Shared file that responds to the host to look like a file allocation table for the drive.
- Fat16.h—Shared header file for Fat16.c
- ParseS19.c—Shared file that parses the S-record file. It verifies each S-record in the file, and after verified, copies the S-record to the specified location in the flash.
- ParseS19.h—Shared header file for ParseS19.c.
- SCSI_Process.c—Shared file that manages SCSI commands from the host.
- SCSI_Process.h—Shared header file for SCSI_Process.c.
- Bootloader_Headers.h—Part-specific file points the shared source files to the part-specific header files.

- ansiis.lib—Compiler library required for the S08 bootloader.
- Project.prm—Part-specific linker command file used to store bootloader routines in protected flash, and the application code in unprotected flash.

There are two application examples included that blink different LEDs on the DEMOJM board. Figure 7 shows the project window of the CodeWarrior project in the directory **Application with Bootloader S08 Blinks PTE2**. Below is a description of the files included with this project.

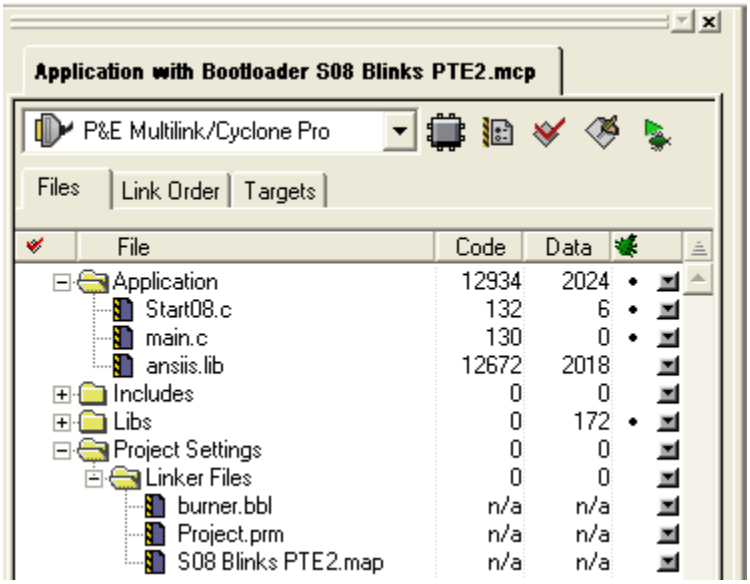

Figure 7. MC9S08 Application with bootloader project files

- main.c—Project-specific main file for the application. Contains user entry jump vector, and the re-directed interrupt vector table
- Project.prm—Part-specific linker command file used to store bootloader routines in protected flash, and application code in unprotected flash. Links bootloader's S19 file into the application project using HEXFILE command.

4 Using the Bootloader

The following sections describe how to use the bootloader. They include how to demonstrate the bootloader on a development board, and how to integrate with other applications.

4.1 Prevent Cross-Calling

It is important to understand that good bootloaders must only interact with the application through absolute addresses. In this case, the user entry vector resides at an absolute address, and the vector points to the entry point of the application. The bootloader knows this absolute address, when it determines to enter the application mode it jumps to the user entry vector which then jumps to the beginning of the application. This is important because every time the application is re-compiled, the linker might change the address of the entry point. The bootloader does not need to know the changed address because the user entry vector is always at the same location and always points to the application entry point. It is up to the user to ensure

that the user entry vector properly points to the application. This process is described later in this document.

Cross-calling is when the bootloader or application calls a function in another section without an absolute address. This must always be avoided. This bootloader has projects that combine the bootloader and application in a single project. Using a single project has the advantage of programming the device only once in production for both the bootloader and application. It also makes debugging easier. However, having a single project also has the potential for cross-calling and can be hazardous.

Consider this scenario:

A bootloader is used with an ANSI library and compiled in the same project with an application that uses the same library. RevA of the firmware is compiled and a library function called MyFunc() is linked into the bootloader section at address 0x100. The application also uses MyFunc() and cross-calls into the bootloader while executing it. RevA is released for production and everything works fine. After some time passes, the firmware is updated to RevB. While re-compiling the project, the linker moves MyFunc() to address 0x105. The bootloader and application for RevB are linked together, so this version works fine as is. Now the firmware update for RevB is uploaded into a device with the RevA bootloader. The application is updated to RevB, but the bootloader is still RevA. When the application executes and cross-calls MyFunc(), it jumps to address 0x105, but MyFunc() in the RevA bootloader is at address 0x100, and the code will runaway.

This example demonstrates why cross-calling between the application and the bootloader must always be avoided except through absolute addresses.

The only interaction this bootloader has with the application is when the bootloader launches the application that uses the absolute address of the user entry vector. The MC9S08 version of this bootloader integrates with the application differently than the ColdFire versions. The ColdFire versions include the bootloader source or library with the application project. The MC9S08 version of the bootloader uses the ANSI library ansiis.lib. To prevent a cross-call scenario, the MC9S08 bootloader is compiled as a separate project. The S19 file from the bootloader project is included with the application project using the HEXFILE command in the PRM linker file. With this method, the linker is not aware of the library functions in the bootloader while compiling the application and there is no cross-calling. The application project continues to include the bootloader code and continues with the benefit of programming only once in production.

When compiling the bootloader with the application, cross-calling must be tested before a production release of the firmware. An easy method to test that the bootloader does not cross-call application functions is to erase the application and test that the bootloader continues to work properly. This bootloader is self-sufficient and works when no application is present. To use this test method, program the device with the combined bootloader and application project. Then, erase the application sectors of flash. These sectors can be erased manually or the debugger can be used. With the debugger, place a breakpoint immediately after the bootloader erases the application during an update. Exercise the bootloader to update the firmware and when the breakpoint is reached, force a reset. The bootloader should continue to function normally. If it does not, the bootloader has a cross-call into the application. Use the debugger to find where the cross-call occurs, and then modify the project to remove the cross-call.

Test that the application does not cross-call into the bootloader. One method to do this is to carefully read the .MAP file after compiling. Read all the functions stored in the bootloader portion of the flash and make

sure they are functions used only by the bootloader, not the application. Ensure no libraries are included in the bootloader. Check any start-up routines used, because those function names might be used by both the bootloader and the application. Another method to test for cross-calling is to erase the bootloader and test the application. This method can be tricky because the reset and interrupt vectors are also located in the bootloader section.

4.2 Using Bootloader on the Demo Board

The included CodeWarrior projects have application examples to test with the M52259DEMO and the DEMOJM development boards. Choose a core to test with the corresponding board. Please refer to the documentation included with these boards to get started and learn how to program the boards with CodeWarrior. After getting familiar with the boards and CodeWarrior, follow these steps:

1. The hardware must be setup for USB device operation. Use the default jumper settings. Refer to the board documentation for more details. If using the DEMOJM board, ensure that the proper MC9S08 or ColdFire V1 module is plugged into the board. Plug both USB cables into the board and the computer.
2. Open the corresponding CodeWarrior and open one of the .MCP project files for one of the application examples. With a ColdFire V2 project, make sure the target “52259 Example Flash” is selected. Compile the project and program the flash.
3. Reset the board. One of the LEDs flashes on the board. This shows the application is running.
4. Force the bootloader mode by holding down a button on the board, then press and release the **Reset** button. For the DEMOJM board, use button **PTG0**. For the M52259DEMO board, use button **SW1**. The device reboots in bootloader mode and enumerates a new drive in the host computer. Figure 8 shows the bootloader drive in Windows XP. Notice the volume name of the drive is BOOTLOADER, and the status file is READY.TXT. The bootloader is now ready to receive an S19 file.

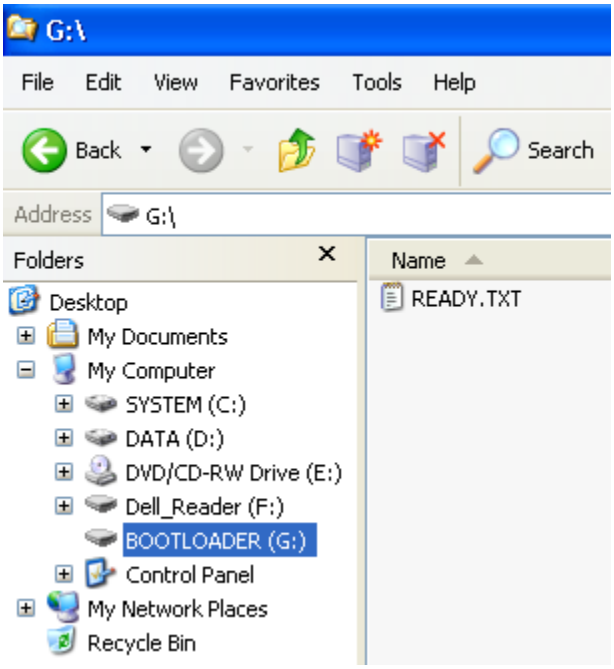

Figure 8. Bootloader drive after enumeration

5. Go to the directory “S19 Files”, copy one of the S19 files for that core and paste it into the bootloader drive. Dragging and dropping the file also works. In the directory **S19 Files**, choose either an S19 file that blinks another LED or choose the CMX Example S19 file that loads the HID example.
6. Some operating systems like Linux Fedora 8 do not transfer the file to the drive until the drive is unmounted. If using an operating system like this, unmount the bootloader drive.
7. Check the bootloader file status. After the file transfer is complete, the bootloader drive disappears. After a few seconds, the drive re-enumerates and re-appears. Click on the bootloader drive to verify the status file is SUCCESS.TXT. The firmware update is then complete.

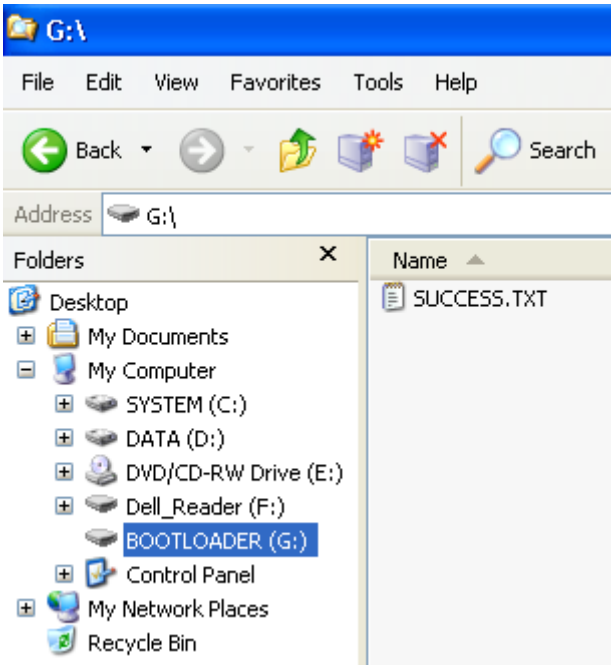

Figure 9. Bootloader drive with successful status file

8. Reset the board using the **Reset** button. The new application is then executed. If the CMX example S19 file was used, the mouse pointer wiggles back and forth on the host computer. If one of the other S19 files were used, a LED blinks on the board. To repeat the process and load another S19 file, go back to step 4.

4.3 Creating a New Project with the Bootloader

The bootloader is intended to be integrated with other applications. The provided application examples are templates that can be used to get started. Starting a new project with the bootloader is simple because the application examples can be used as a starting point. Integrating the bootloader with existing applications is more difficult and requires several modifications. These steps are listed in the following sections.

When starting a new project with the bootloader, begin the process by modifying one of the application examples with the new application. The application example already has the bootloader integrated and has the start-up functions provided. You can start writing the new application functionality in the main() in main.c. The only change that needs to be made to the application example is the interrupt vector table. The re-directed interrupt vector table is different for each core and is discussed in greater detail in the following sections.

4.3.1 Modifying the ColdFire V2 Interrupt Vector Table

The ColdFire V2 core has a feature to change the base address of the interrupt vector table. The register VBR stores the base address of the vector table and can be changed as you go along. Out of reset, the VBR is 0x0, which places the vector table in the protected flash sector. Therefore, the reset vector is at address 0x4 and is also protected and controlled by the bootloader. After the application starts, the application can

change the VBR to place the vector table in the RAM. Then the application can load its interrupt vectors into the RAM vector table.

```
void main (void)
{

    // Set the interrupt handlers in the vector table
    mcf5xxx_wr_vbr((uint32)__VECTOR_RAM);
    mcf5xxx_set_handler(64 + 55, (long)pit0_isr);
```

Figure 10. ColdFire V2 re-directing interrupt vectors

Figure 10 shows the two lines of code used in the ColdFire V2 application example to redirect the interrupt vector table to RAM and load the vector table. The first line uses the function `mcf5xxx_wr_vbr()` to place the vector table in the RAM. The second line uses the function `mcf5xxx_set_handler()` to load a vector into the vector table. In this example, the interrupt is for the Periodic Interrupt Timer0 and the ISR associated with this interrupt is `pit0_isr()`. In the MCF52259, the PIT0 interrupt vector is the 55th vector for Interrupt Controller0. Therefore, the vector number is 64 + 55.

When changing the vector table for the new application, use the `mcf5xxx_set_handler()` to set the new vectors to point to the appropriate ISRs.

4.3.2 Modifying ColdFire V1 Interrupt Vector Table for a New Application

The ColdFire V1 core has a feature to change the base address of the interrupt vector table. The register VBR stores the base address of the vector table and can be changed as you go along. Out of reset, the VBR is 0x0, this places the vector table in the protected flash sector. Therefore, the reset vector is at address 0x4 and is also protected and controlled by the bootloader. After the application starts, the application can change VBR to place the vector table in the RAM. Then the application can load its interrupt vectors into the RAM vector table.

The ColdFire V1 application stores its own vector table in the flash. Figure 11 shows a portion of the vector table called RAM_vector stored in the application flash. The whole table can be seen in Appendix F, “hid_main.c for the ColdFire V1 CMX Example,” on page 61. Because this vector table is stored in the application flash, the bootloader can update with a new firmware update.

```
void  (* const RAM_vector[])()@REDIRECT_VECTORS= {

    (pFun)&dummy_ISR,                // vector_0  INITSP
    (pFun)&dummy_ISR,               // vector_1  INITPC
    (pFun)&dummy_ISR,               // vector_2  Vaccerr

    .
    .
    .

    (pFun)&dummy_ISR,               // vector_75 Vtpm1ch4
    (pFun)&dummy_ISR,               // vector_76 Vtpm1ch5
    (pFun)&Timer_Overflow,               // vector_77 Vtpm1ovf
    (pFun)&dummy_ISR,               // vector_78 Vtpm2ch0
    (pFun)&dummy_ISR,               // vector_79 Vtpm2ch1

    .
    .
    .
};
```

Figure 11. ColdFire V1 application interrupt vector table

```
void main(void) {

  /* !! This section needs to be here to redirect interrupt vectors !! */
  dword *pdst,*psrc;
  byte i;

  asm (move.l  #0x00800000,d0);
  asm (movec   d0,vbr);

  pdst=(dword*)0x00800000;
  psrc=(dword*)&RAM_vector;

  for (i=0;i<111;i++,pdst++,psrc++)
  {
    *pdst=*psrc;
  }

  /* !! Start application code below here !! */
```

Figure 12. ColdFire V1 interrupt re-direction

When the application starts, the first two lines load the VBR with the RAM vector table location that re-directs the interrupt vector table to the RAM. Then the loop copies the application vector table from the flash to the RAM.

With a new application, load the RAM_vector array with the ISR functions at the appropriate vector location.

4.3.3 Modifying the MC9S08 Interrupt Vector Table for the New Application

The MC9S08 core cannot re-direct the vector table to the RAM like ColdFire. Instead, the bootloader points to the interrupt vectors to a re-directed table in the application flash. The re-directed vector table is stored at a specific address. In Figure 13, the array UserJumpVectors in main.c is the re-directed vector

table, and it starts at address VectorAddressTableAddress, which is 0xEBA6. Notice in the table, each ISR address is preceded by the value 0xCC. This value is the JMP op-code.

```
// User Interrupt Jump Vector Table
volatile const JumpVect UserJumpVectors[InterruptVectorsNum]@ VectorAddressTableAddress = {
    { 0xCC, Dummy_ISR},                 // 29 - RTC
    { 0xCC, Dummy_ISR},                 // 28 - IIC
    { 0xCC, Dummy_ISR},                 // 27 - ACMP
    { 0xCC, Dummy_ISR},                 // 26 - ADC Conversion
    { 0xCC, Dummy_ISR},                 // 25 - KBI
    { 0xCC, Dummy_ISR},                 // 24 - SCI2 Transmit
    { 0xCC, Dummy_ISR},                 // 23 - SCI2 Receive
    { 0xCC, Dummy_ISR},                 // 22 - SCI2 Error
    { 0xCC, Dummy_ISR},                 // 21 - SCI1 Transmit
    { 0xCC, Dummy_ISR},                 // 20 - SCI1 Receive
    { 0xCC, Dummy_ISR},                 // 19 - SCI1 Error
    { 0xCC, Dummy_ISR},                 // 18 - TPM2 Overflow
    { 0xCC, Dummy_ISR},                 // 17 - TPM2 Channel1
    { 0xCC, Dummy_ISR},                 // 16 - TPM2 Channel0
    { 0xCC, Timer_Overflow},            // 15 - TPM1 Overflow
    { 0xCC, Dummy_ISR},                 // 14 - TPM1 Channel5
    { 0xCC, Dummy_ISR},                 // 13 - TPM1 Channel4
    { 0xCC, Dummy_ISR},                 // 12 - TPM1 Channel3
    { 0xCC, Dummy_ISR},                 // 11 - TPM1 Channel2
    { 0xCC, Dummy_ISR},                 // 10 - TPM1 Channel1
    { 0xCC, Dummy_ISR},                 //  9 - TPM1 Channel0
    { 0xCC, Dummy_ISR},                 //  8 - Reserved
    { 0xCC, Dummy_ISR},                 //  7 - USB Status
    { 0xCC, Dummy_ISR},                 //  6 - SPI2
    { 0xCC, Dummy_ISR},                 //  5 - SPI1
    { 0xCC, Dummy_ISR},                 //  4 - MCG Loss of Lock
    { 0xCC, Dummy_ISR},                 //  3 - Low Voltage Detect
    { 0xCC, Dummy_ISR},                 //  2 - IRQ
    { 0xCC, Dummy_ISR},                 //  1 - SWI
};
```

Figure 13. MC9S08 Application re-directed interrupt vector table

The bootloader uses the array BootIntVectors in the file Redirect_Vectors_S08.c to load the interrupt vector table in the bootloader flash (see Figure 14 on page 24). These vectors point to the re-directed application vectors in Figure 13. When an interrupt occurs, the interrupt controller reads the vector from the BootIntVectors table in the bootloader flash. It changes the program counter (PC) to the loaded address, that is the re-directed vector in the UserJumpVectors table in the application flash. The core executes the 0xCC JMP op-code, followed by the ISR's address, and then PC jumps to the ISR.

```
// Bootloader Redirected Interrupt Vectors
const unsigned int BootIntVectors[InterruptVectorsNum]@BootVectorTableAddress = {
    VectorAddressTableAddress,
    VectorAddressTableAddress+3,
    VectorAddressTableAddress+6,
    VectorAddressTableAddress+9,
    VectorAddressTableAddress+12,
    VectorAddressTableAddress+15,
    VectorAddressTableAddress+18,
    VectorAddressTableAddress+21,
    VectorAddressTableAddress+24,
    VectorAddressTableAddress+27,
    VectorAddressTableAddress+30,
    VectorAddressTableAddress+33,
    VectorAddressTableAddress+36,
    VectorAddressTableAddress+39,
    VectorAddressTableAddress+42,
    VectorAddressTableAddress+45,
    VectorAddressTableAddress+48,
    VectorAddressTableAddress+51,
    VectorAddressTableAddress+54,
    VectorAddressTableAddress+57,
    VectorAddressTableAddress+60,
    VectorAddressTableAddress+63,
    VectorAddressTableAddress+66,
    VectorAddressTableAddress+69,
    VectorAddressTableAddress+72,
    VectorAddressTableAddress+75,
    VectorAddressTableAddress+78,
    VectorAddressTableAddress+81,
    VectorAddressTableAddress+84,
};
```

Figure 14. MC9S08 bootloader interrupt vector table

For a new application, load the array UserJumpVectors in main.c with the proper application ISRs.

4.4 Adding the Bootloader to the Existing ColdFireV2 Project

This section gives step-by-step instructions on how to take an existing ColdFire V2 CodeWarrior project and add the bootloader. This example uses the HID-DEMO from CMX for the ColdFire V2 with the complimentary CMX USB stack. For licensing reasons, the CMX source code is not included with this application note's firmware. However, the following steps give instructions on how to take the CMX example and add the bootloader. The CMX software is included in the DVD that comes with the MCF52259 board. The document titled *MCF52259 CMX USB LITE Lab Tutorial* data sheet (document M52259CMXUSBLAB) describes how to use the CMX example with the board.

The following steps reference directories and files for both the bootloader and CMX example. The bootloader files and paths are referenced from the root directory of this application note's firmware. The CMX example files and paths are referenced from the root directory of the CMX software installation.

1. Remove any references to the reset and interrupt vectors in the application. The bootloader uses the reset vector to point to its start-up function that determines if the bootloader or application mode must start. If the application keeps a reset vector reference, this can cause an error because two vectors attempt to use the same location.

   A common method for the reset vector is to provide a constant vector table in a project file, for example the mcf5222x_vectors.s. This vector table is forced to the proper flash location for the vectors either by using a fixed address in the vector table declaration, or by using a segment

declared in the LCF file. For the bootloader, this vector table must be removed. The interrupts are re-directed in the steps below.

For this CMX example, it uses the same vector table as the bootloader in mcf5222x_vectors.s. To integrate the bootloader with the CMX example, the bootloader version of the vector table is used. The CMX example version needs to be changed to keep the start-up function. Modify the following file with the steps below:

\usb-peripheral\projects\CodeWarrior\mcf52223\mcf5222x_vectors.s

a) Remove the VECTOR_TABLE with all the interrupt vectors. This table is replaced by the bootloader's vector table.

b) Remove the following non-volatile register settings. These register settings are replaced by the bootloader.

```
    .org 0x400
KEY_UPPER:  .long   0x00000000
KEY_LOWER:  .long   0x00000000
CFMPROT:    .long   0x00000000
CFMSACC:    .long   0x00000000
CFMDACC:    .long   0x00000000
CFMSEC:     .long   0x00000000
```

c) Modify the `start` function. This modified example needs to reference the `start` function in assembly from C. Change the declaration of the function by adding an underscore before the label so that C functions can reference it:

```
_start:
```

d) Add another global declaration for the start function so that C files can reference it. Add this line with the other global declarations:

```
.global _start
```

e) Because many modifications were made to mcf5222x_vectors.s, the entire source code for this file is given in Appendix A, "mcf5222x_vectors.s for the Coldfire V2 CMX Example," on page 43.

2. Copy the following files from the bootloader project into the application project using the directories below. After copying, add these files to the CodeWarrior project. To do this, in CodeWarrior right-click in the project window, and select **Add Files** navigate to the files below, select them all and click **Open**. Then select **OK**.

   Bootloader_V2.h
   mcf5225x_vectors.s
   usr_entry_V2.c
   V2_Bootloader.lib

Directory

From:	\Shared Source\V2_Source
To:	\usb-peripheral\src\mcf5222x\hid-demo

3. The bootloader files should now be visible in the CodeWarrior Project window shown in Figure 15.

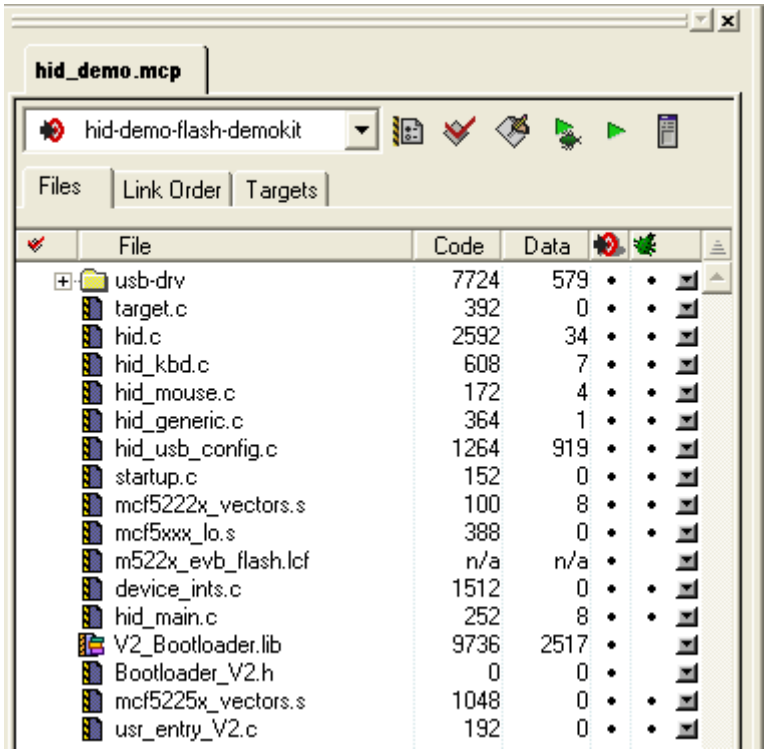

Figure 15. ColdFire V2 CMX example project window files

4. Copy/Edit the LCF linker file. The LCF file is critical when integrating the bootloader with an application. The bootloader LCF file must be used as the template for the new application LCF file. However, modifications may be required based on the application. The existing application's LCF file needs to be well understood. For more information on linker files, please read the *CodeWarrior Build Tools Reference Manual*. Pay particular attention to the memory sections, and stack settings of the existing LCF file. For this CMX example, the following modifications are required.
   a) Use the provided bootloader LCF file as the starting template. Copy and rename the file below overwriting the original LCF file with the CMX example.

Directory

From:	\Shared Source\V2_Source\mcf52259_bl_flash.lcf
To:	\usb-peripheral\projects\CodeWarrior\mcf52223\m522x_evb_flash.lcf

   b) Add the USB buffer descriptor table (BDT) settings to the LCF file. This CMX example uses the LCF to determine where to place the USB buffer descriptor table. From the original LCF file with this CMX example, copy the code below and paste into the modified LCF after the following line in the `.bss` section: `__BSS_END = .;`

```
/* Buffer descriptor base address
   shall be aligned to 512 byte boundary.
   Size shall be 512 bytes. */
.               = ALIGN(512);
__BDT_BASE      = .;
.               = . + 512;
```

```
__BDT_END       = .;
```

c) Because these modifications were made to this LCF file, the entire source code for the LCF file is given in Appendix B, “LCF File for ColdFire V2 CMX Example,” on page 47.

5. Modify the main application file to integrate with the bootloader. For this CMX example, modify hid_main.c with the following:

a) Include the Bootloader_V2.h header file in the main application file. Add the following line to hid_main.c:

```
#include "Bootloader_V2.h"
```

b) Add the user entry jump vector to the main application file. For this CMX example, the user entry point is `start`. Add the following code to hid_main.c:

```
extern asm void start(void);
void usb_it_handler(void);
const byte _UserEntry[] @ USER_ENTRY_ADDRESS = {
  0x4E,
  0x71,
  0x4E,
  0xF9   //asm NOP(0x4E71), asm JMP(0x4EF9)
};

void  (* const _UserEntry2[])()@(USER_ENTRY_ADDRESS+4)= {
  start,
};
```

c) Add the re-directed interrupt vectors to the RAM. In this CMX example, only the USB interrupt is used. The USB interrupt is vector 53 and needs to point to the ISR `usb_it_handler()`. Add the following code to the start of the `main` function before `hw_init()` is called:

```
__VECTOR_RAM[64 + 53] = (hcc_u32) usb_it_handler;
```

d) Re-direct the interrupt vectors to RAM. Add the line below after `hw_init()` is called. It is important to add this line after `hw_init()` because `hw_init()` sets VBR to point to the flash:

```
mcf5xxx_wr_vbr((hcc_u32) __VECTOR_RAM);
```

e) Because several changes were made to the hid_main.c file, Appendix C, “hid_main.c for ColdFire V2 CMX Example,” on page 51 provides the source code for the entire file.

6. Compiling at this point can cause errors because the CMX example uses different header files used for the ColdFire registers than for the bootloader. Modify Bootloader_V2.h and usr_entry_V2.c by changing the code below to include the other header file:

```
#include "support_common.h"
```

to

```
#include "mcf5222x_reg.h"
```

7. Compiling at this point can cause errors because usr_entry_V2.c references the start-up function for the bootloader example. There are two references in this file to `asm_startmeup`. Replace both with `start`. Refer to Appendix D, "usr_entry_V2.c for ColdFire V2 CMX Example," on page 54 for the full source file.
8. Change the entry point for the linker. The linker needs to know what function is the entry point. This needs to be modified to point to the bootloader `_Entry` function. Change the project properties by clicking on the **Edit Menu -> hid_demo-flash-demokit Settings**, select **ColdFire Linker** in the **Target Settings Panel** on the left. Change the entry point to `__Entry`.

NOTE

`__Entry` has 2 underscores.

Then click OK.

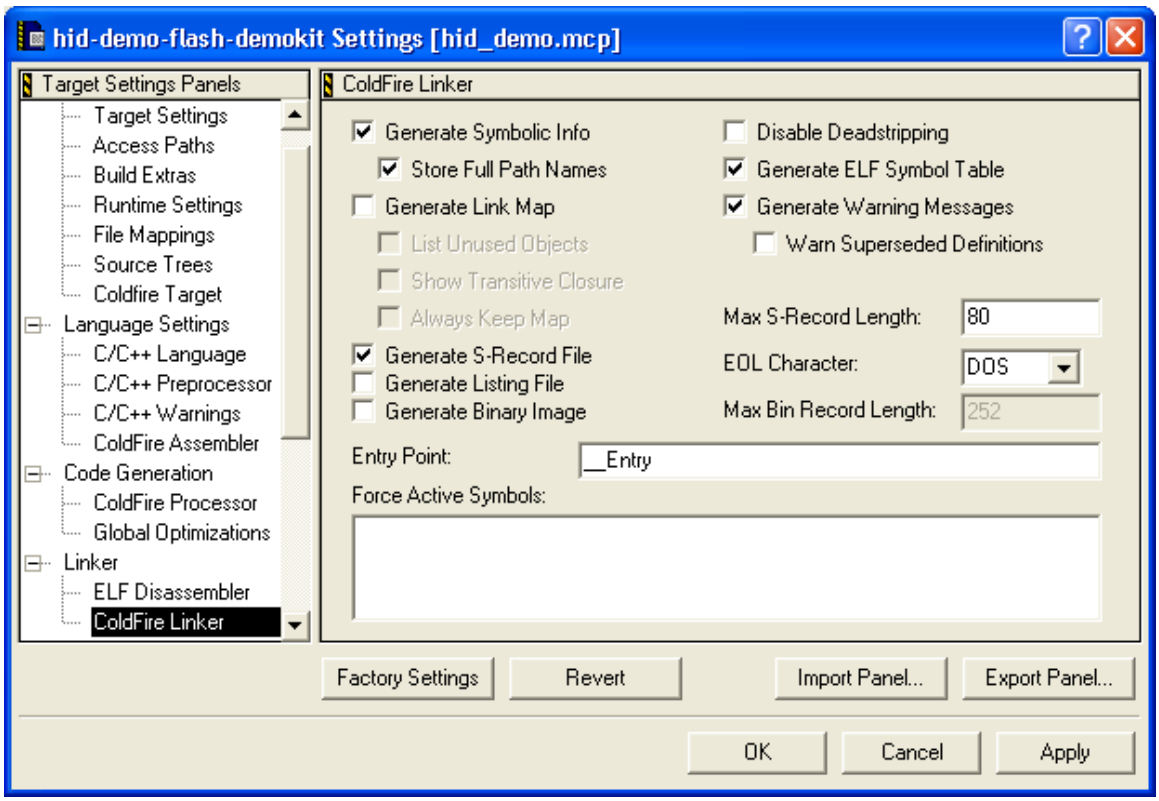

Figure 16. ColdFire V2 CMX example linker settings

9. Compile and download the project into the board. The CMX example works as described in the *MCF52259 CMX USB LITE Lab tutorial* document. Hold down the SW1 button while pressing and releasing the Reset button to enter bootloader mode. The bootloader drive appears in the PC. Copy another application's S19 file onto the drive to see the application change. For example, use either the ColdFire V2 S19 file in the bootloader directory \S19 Files. Enter bootloader mode again and copy the CMX example S19 file back to the board using the file **\usb-peripheral\projects\CodeWarrior\mcf52223\hid-demo\hid-demo.elf.S19.**

4.5 Adding Bootloader to the Existing ColdFire V1 Project

This section gives step-by-step instructions on how to take an existing ColdFire V1 CodeWarrior project and add the bootloader. This example uses the HID-DEMO from CMX for the ColdFire V1 with the complimentary CMX USB stack. For licensing reasons, the CMX source code is not included with this application note's firmware. However, the steps below give instructions on how to take the CMX example and add the bootloader. The CMX software can be downloaded from Freescale's website at http://www.freescale.com/usb.

The steps below reference directories and files for both the bootloader and CMX example. The bootloader files and paths are referenced from the root directory of this application note's firmware. The CMX example files and paths are referenced from the root directory of the CMX software installation.

1. Copy the following files from the bootloader project into the application project using the directories below. After copying, add these files to the CodeWarrior project. To do this, in CodeWarrior right-click on the hid-demo file group in the project window, and select **Add File** Navigate to the files below, select them all and click **Open**. Then select **OK**.

   Bootloader_V1.c
   Bootloader_V1.h
   exceptions.c
   exceptions.h
   usr_entry_V1.c
   Bootloader_Headers.h

	Directory
From:	\Shared Source\V1_Source\
To:	\usb-peripheral\projects\CodeWarrior-6.x\mcf51xx\hid-demo\Sources\

2. Copy the following files from the bootloader project into the application project using the directories below. After copying, add these files to the CodeWarrior project just like the previous step.

   FAT16.c
   FAT16.h
   ParseS19.hc
   ParseS19.h
   SCSI_Process.c
   SCSI_Process.h

Directory	
From:	\Shared Source\Common_Across_Cores\
To:	\usb-peripheral\projects\CodeWarrior-6.x\mcf51xx\hid-demo\Sources\

3. The bootloader files are now visible in the CodeWarrior Project window as shown in Figure 17:

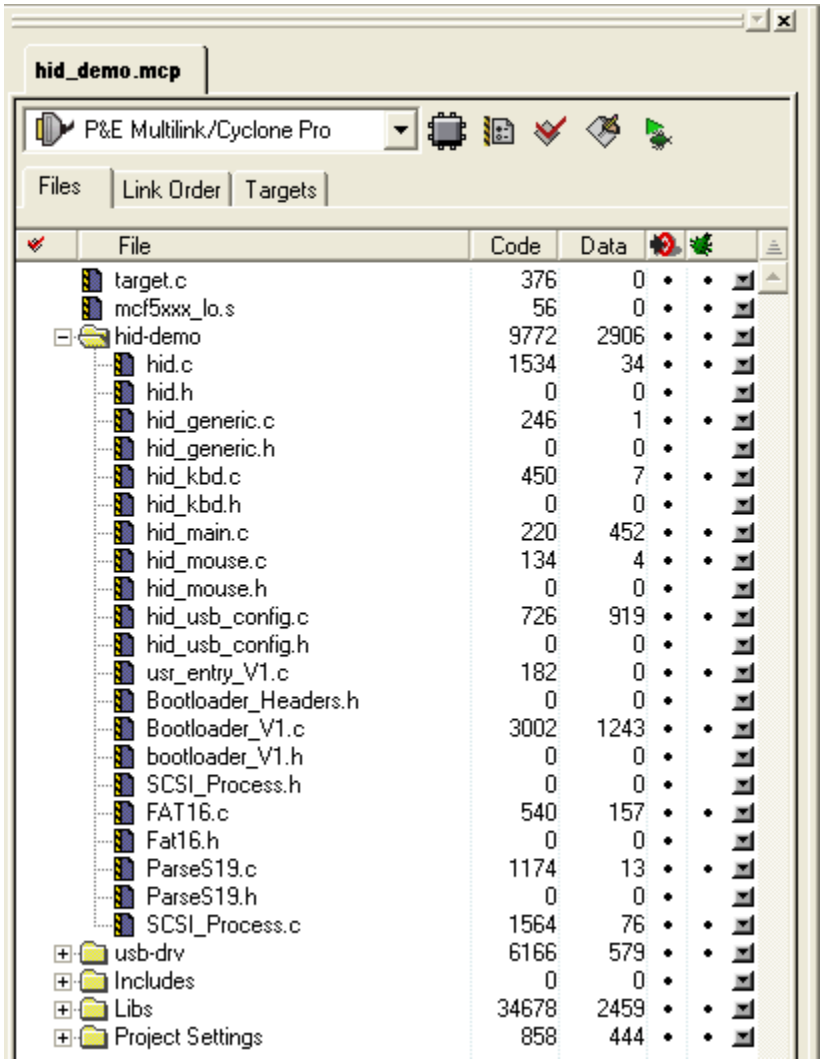

Figure 17. ColdFire V1 CMX Example project window files

4. Copy/Edit the LCF linker file. The LCF file is critical when integrating the bootloader with an application. The bootloader LCF file must be used as the template for the new application LCF file. However, modifications may be required based on the application. The existing application's LCF file needs to be well understood. For more information on linker files, please read the *CodeWarrior Build Tools Reference Manual*. Pay attention to the **Memory** sections, and **Stack** settings of the existing LCF file. For this CMX example, several modifications are required. Open the CMX example LCF file below and make the following changes:

   `\usb-peripheral\projects\CodeWarrior-6.x\mcf51xx\hid-demo\prm\Project_flash.lcf`

   a) The MEMORY section needs to be modified to segment the memory into the bootloader and application partitions. Overwrite the MEMORY section in the LCF file with the MEMORY section used in the bootloader example LCF file. Overwrite with the text below:

```
MEMORY {
   bootcode   (RX)   : ORIGIN = 0x000410, LENGTH = 0x00001BF0
   code       (RX)   : ORIGIN = 0x002200, LENGTH = 0x0001DE00
   #code      (RX)   : ORIGIN = 0x000410, LENGTH = 0x0001FBF0
   #userram   (RWX)  : ORIGIN = 0x800000, LENGTH = 0x00004000
   vectorram  (RWX)  : ORIGIN = 0x800000, LENGTH = 0x00000200
   userram    (RWX)  : ORIGIN = 0x800200, LENGTH = 0x00003E00
   bootsram   (RWX)  : ORIGIN = 0x8006E0, LENGTH = 0x00003B60
}
```

   b) The stack size needs to be increased. The bootloader and application share the stack, it must be the same size and location of the bootloader. Change the stack size by changing the line below:

```
___stack_size     = 0x400;
```

   c) The bootloader sections need to be added to the LCF file to tell the linker where to place the bootloader code and RAM. Copy the following sections from the bootloader example LCF file into the CMX example LCF file, and add below the `.code` section:

```
.bootcode:
    {
      ___Boot_START    = .;
        Bootloader_V1.c  (.text)
        Bootloader_V1.c  (.rodata)
        usr_entry_V1.c   (.text)
        ParseS19.c       (.text)
        ParseS19.c       (.rodata)
        SCSI_Process.c   (.text)
        SCSI_Process.c   (.rodata)
        FAT16.c          (.text)
        FAT16.c          (.rodata)
        exceptions.c     (.text)

        . = ALIGN (0x4);
        ___Boot_END      =.;
  } > bootcode

  .bootsram:
    {
      ___Boot_RAM_START    = .;
        Bootloader_V1.c                (.bss)
        Bootloader_V1.c                (.sbss)
```

```
        usr_entry_V1.c                  (.bss)
        usr_entry_V1.c                  (.sbss)
        ParseS19.c                        (.bss)
        ParseS19.c                        (.sbss)
        SCSI_Process.c    (.bss)
        SCSI_Process.c    (.sbss)
        FAT16.c                                                   (.bss)
        FAT16.c                                                  (.sbss)

        . = ALIGN (0x4);
        ___Boot_RAM_END     =.;
    } > bootsram
```

d) The stack location needs to be changed in the `.custom` section of the LCF file. The bootloader and application share the stack, it must be the same size and location of the bootloader. Use the bootloader example LCF file to see where the stack should be located, and change the `.custom` section as shown below:

```
.custom :
  {
    ___HEAP_START = .;
    ___heap_addr  = ___HEAP_START;
    ___HEAP_END   = ___HEAP_START + ___heap_size;
    . = ___HEAP_END;
    .       = ALIGN(512);
    __BDT_BASE    = .;
    .             = . + 512;
    __BDT_END     = .;

    ___SP_INIT    = ___RAM_ADDRESS + ___RAM_SIZE;
    ___SP_END     = ___SP_INIT - ___stack_size;
  } >> userram
```

e) Because several modifications were made to this LCF file, the entire source code for the LCF file is given in Appendix E, “LCF File for ColdFire V1 CMX Example,” on page 57.

5. Modify the main application file to integrate with the bootloader. For this CMX example, modify hid_main.c with the following:

a) Include the Bootloader_V1.h header file in the main application file. Add the following line to hid_main.c:

```
#include "Bootloader_V1.h"
```

b) Add the user entry jump vector to the main application file. For this CMX example, the user entry point is _startup(). Add the following lines to hid_main.c:

```
extern asm void _startup(void);
const byte _UserEntry[] @ USER_ENTRY_ADDRESS = {
  0x4E,
  0x71,
  0x4E,
  0xF9          //asm NOP(0x4E71), asm JMP(0x4EF9)
};

void  (* const _UserEntry2[])() @(USER_ENTRY_ADDRESS+4)=
{
  _startup,
};
```

c) Remove any references to a reset vector in the application. The bootloader uses the reset vector to point to its start-up function that determines if the bootloader or application mode start. If the application keeps a reset vector reference, this can cause an error because two vectors attempt to use the same location.

A common method for the reset vector is to provide a constant vector table in a project file, for example exceptions.c. This vector table is forced to the proper flash location for the vectors either by using a fixed address in the vector table declaration, or by using a segment declared in the LCF file. For the bootloader, this vector table must be removed. Interrupts are re-directed in the steps below.

For this CMX example, the existing application used a vector table in exceptions.c to reference `_startup()`. Overwriting the exceptions.c file has already implemented this change.

d) The re-directed interrupt vector table is added in the next step. To prepare for it, copy the following lines to the hid_main.c:

```
__interrupt void dummy_ISR(void) {}

typedef void (* pFun)(void);
```

e) Add the re-directed interrupt vector table, and point the re-directed vectors to the application's interrupt service routines. In this CMX example, only the USB interrupt is used. The vector table is too large to display in its entirety here. Copy the entire `RAM_vector[]` table from the bootloader application example file **\V1 USB Bootloader Projects\USB Bootloader V1 Blinks PTE2\Sources\main.c** to the hid_main.c file. Then, change the ISR name for USB vector 69 to `usb_it_handler`. The bootloader application also used a timer interrupt. Change that vector 77 to `dummy_ISR`. Below gives a snapshot of the modified CMX Example vector table:

```
void  (* const RAM_vector[])()@REDIRECT_VECTORS= {
    (pFun)&dummy_ISR,     // vector_0  INITSP
    (pFun)&dummy_ISR,     // vector_1  INITPC
    .
    .
    .
    (pFun)&dummy_ISR,     // vector_68 Vspi2
    (pFun)&usb_it_handler, // vector_69 Vusb
    (pFun)&dummy_ISR,     // vector_70 VReserved70
    (pFun)&dummy_ISR,     // vector_71 Vtpm1ch0
```

```
    (pFun)&dummy_ISR,     // vector_72 Vtpm1ch1
    (pFun)&dummy_ISR,     // vector_73 Vtpm1ch2
    (pFun)&dummy_ISR,     // vector_74 Vtpm1ch3
    (pFun)&dummy_ISR,     // vector_75 Vtpm1ch4
    (pFun)&dummy_ISR,     // vector_76 Vtpm1ch5
    (pFun)&dummy_ISR,     // vector_77 Vtpm1ovf
    (pFun)&dummy_ISR,     // vector_78 Vtpm2ch0
    .
    .
    .
    (pFun)&dummy_ISR,     // vector_110 VL1swi
};
```

f) The application needs to re-direct the interrupts to RAM and needs to copy the interrupt vectors from the flash to RAM. For this CMX example, copy the following to the beginning of `main()` in hid_main.c:

```
/* !! This section needs to be here to redirect
  interrupt vectors !! */
  dword *pdst,*psrc;
  byte i;

  asm (move.l  #0x00800000,d0);
  asm (movec   d0,vbr);

  pdst=(dword*)0x00800000;
  psrc=(dword*)&RAM_vector;

  for (i=0;i<111;i++,pdst++,psrc++)
  {
    *pdst=*psrc;
  }

  /* !! Start application code below here !! */
```

g) Because several changes were made to the hid_main.c file, Appendix F, “hid_main.c for the ColdFire V1 CMX Example,” on page 61 provides the source code for the entire file.

6. Change the existing interrupt vector table. The existing application places the interrupt vector table in the protected bootloader flash memory. These vectors need to be re-directed. The first step is to change references in the application to the existing interrupt service routines.

   If an interrupt vector number is used when the ISR is declared, CodeWarrior loads that interrupt vector with the address of the declared ISR. With the bootloader, these interrupt vector numbers need to be removed from the ISR declaration.

   Another common method for vector tables is to provide a constant vector table in a project file, for example exceptions.c. This vector table is forced to the proper flash location for the vectors either by using a fixed address in the vector table declaration, or by using a segment declared in the LCF file. For the bootloader, this vector table must be removed.

For this CMX example, the only interrupt used is the USB interrupt. The interrupt vector number of the usb_it_handler ISR declaration needs to be removed. Change the line below in usb.c to the following:

```
interrupt  VectorNumber_Vusb void usb_it_handler(void)
```

to

```
interrupt  void usb_it_handler(void)
```

7. Change the compiler optimization. The CMX example turns off all optimizations, so the bootloader code does not fit into the protected sectors. To change this, open the **Menu Edit -> DEMOJM Flash Settings,** then select **Global Optimizations**. Change the optimization to Level 1.

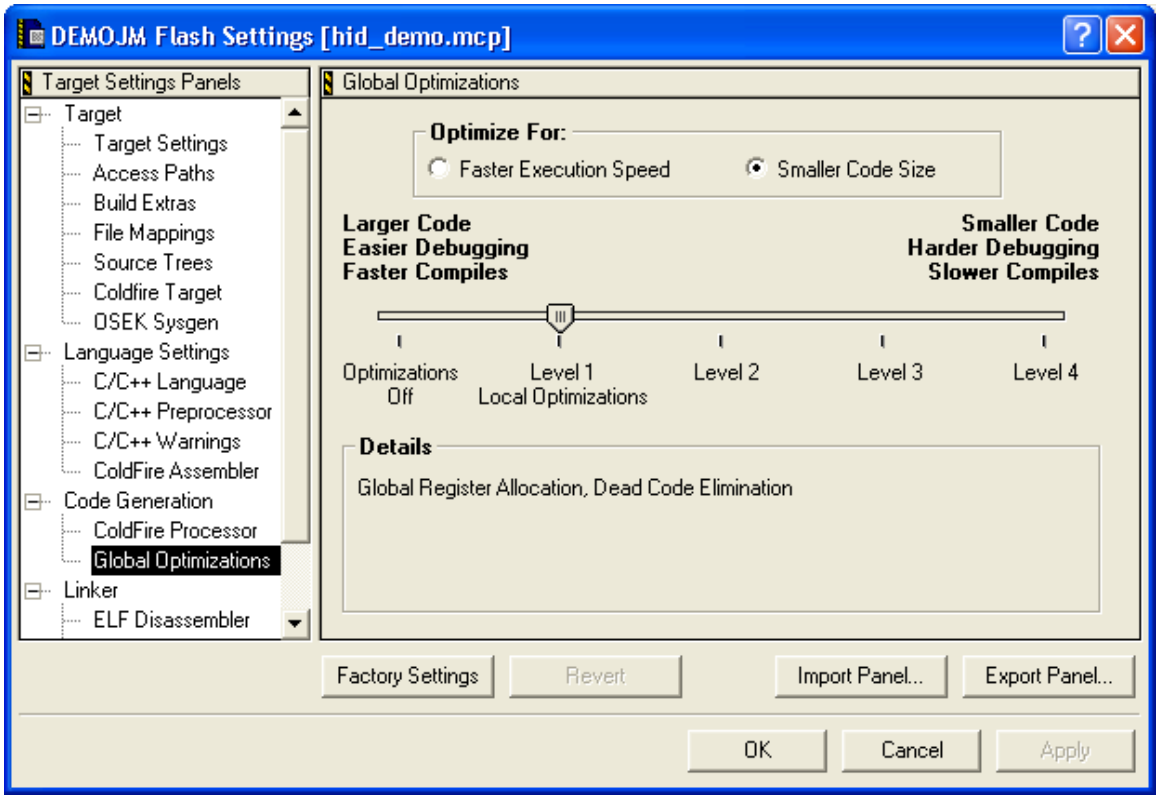

Figure 18. Compiler optimization for the ColdFire V1 CMX Example

8. Compile and download the project into the board. The CMX example works as described in the DEMOJM Lab Supplement for the 32-bit Flexis JM128 document. Hold down the **PTG0** button while pressing and releasing the **Reset** button to enter bootloader mode. The bootloader drive appears in the PC. Copy another application's S19 file onto the drive to see the application change. For example, use either the ColdFire V1 S19 file in the Bootloader directory \S19 Files. Enter bootloader mode again, and copy the CMX example S19 file back to the board using the file \usb-peripheral\projects\CodeWarrior-6.x\mcf51xx\hid-demo\bin\ hid_demo.abs.S19.

4.6 Adding the Bootloader to the Existing MC9S08 Project

This section gives step-by-step instructions on how to take an existing MC9S08 CodeWarrior project and add the bootloader. This example uses the HID-DEMO from CMX for the MC9S08 with the complimentary CMX USB stack. For licensing reasons, the CMX source code is not included with this application note's firmware. However, the steps below give instructions on how to take the CMX example and add the bootloader. The CMX software can be downloaded from Freescale's website at http://www.freescale.com/usb.

The steps below reference directories and files for both the bootloader and CMX example. The bootloader files and paths are referenced from the root directory of this application note's firmware. The CMX example files and paths are referenced from the root directory of the CMX software installation.

1. Copy the following files from the bootloader project into the application project:

File	
From:	\Shared Source\S08_Source\S08_Bootloader.abs.s19
To:	\usb-peripheral\projects\CodeWarrior\hc9S08jm60\hid-demo\Sources
From:	\Shared Source\S08_Source\Bootloader_S08.h
To:	\usb-peripheral\projects\CodeWarrior\hc9S08jm60\hid-demo\Sources\

2. Copy/Edit the PRM linker file. The PRM file is critical when integrating the bootloader with an application. The bootloader PRM file must be used as the template for the new application PRM file. However, modifications may be required based on the application. The existing application's PRM file needs to be well understood. For more information on linker files, please read the *CodeWarrior Build Tools Utilities Manual*. Pay attention to the SEGMENTS and PLACEMENT sections, and STACK settings of the existing PRM file. For this CMX example, do the following steps:
   a) Copy the PRM file from the bootloader application project into the application project overwriting the existing file:

File	
From:	\S08 USB Bootloader Projects \Application with Bootloader S08 Blinks PTE2\prm\Project.prm
To:	\usb-peripheral\projects\CodeWarrior\hc9S08jm60\hid-demo\prm\Project.prm

   b) Modify the stack size. The bootloader was written with a stack size of 80 (0x50) bytes, which is the default stack size for a CodeWarrior project. However, the CMX example requires 144 (0x90) bytes. Change the following lines in the PRM file:

```
MY_STACK        =  READ_WRITE   0x1020 TO 0x10AF;
Application_RAM =  READ_WRITE   0x0100 TO 0x101F;
```

c) The full source of the new PRM file is in Appendix G, "PRM File for the MC9S08 CMX Example," on page 66.

3. Modify the main application file to integrate with the bootloader. For this CMX example, modify hid_main.c with the following:

a) Include the Bootloader_S08.h header file in the main application file. Add the following line to hid_main.c:

```
#include "Bootloader_S08.h"
```

b) Add the user entry jump vector to the main application file. For this CMX example, the user entry point is _Startup(). Add the following lines to hid_main.c:

```
void _Startup(void);
// User Application code entry
volatile const JumpVect UsrEntry@ USER_ENTRY_ADDRESS = {
  0xCC,     // op-code for JMP
  _Startup
};
```

c) Add the re-directed interrupt vector table, and point the re-directed vectors to the application's interrupt service routines. In this CMX example, only the USB interrupt is used. Add the following lines to the hid_main.c file:

```
interrupt void Dummy_ISR(void) {}

// User Interrupt Jump Vector Table
volatile const JumpVect UserJumpVectors [InterruptVectorsNum] @
VectorAddressTableAddress = {
    { 0xCC, Dummy_ISR},    // 29 - RTC
    { 0xCC, Dummy_ISR},    // 28 - IIC
    { 0xCC, Dummy_ISR},    // 27 - ACMP
    { 0xCC, Dummy_ISR},    // 26 - ADC Conversion
    { 0xCC, Dummy_ISR},    // 25 - KBI
    { 0xCC, Dummy_ISR},    // 24 - SCI2 Transmit
    { 0xCC, Dummy_ISR},    // 23 - SCI2 Receive
    { 0xCC, Dummy_ISR},    // 22 - SCI2 Error
    { 0xCC, Dummy_ISR},    // 21 - SCI1 Transmit
    { 0xCC, Dummy_ISR},    // 20 - SCI1 Receive
    { 0xCC, Dummy_ISR},    // 19 - SCI1 Error
    { 0xCC, Dummy_ISR},    // 18 - TPM2 Overflow
    { 0xCC, Dummy_ISR},    // 17 - TPM2 Channel1
    { 0xCC, Dummy_ISR},    // 16 - TPM2 Channel0
    { 0xCC, Dummy_ISR},    // 15 - TPM1 Overflow
    { 0xCC, Dummy_ISR},    // 14 - TPM1 Channel5
    { 0xCC, Dummy_ISR},    // 13 - TPM1 Channel4
    { 0xCC, Dummy_ISR},    // 12 - TPM1 Channel3
    { 0xCC, Dummy_ISR},    // 11 - TPM1 Channel2
    { 0xCC, Dummy_ISR},    // 10 - TPM1 Channel1
    { 0xCC, Dummy_ISR},    //  9 - TPM1 Channel0
    { 0xCC, Dummy_ISR},    //  8 - Reserved
    { 0xCC, usb_it_handler}, //  7 - USB Status
    { 0xCC, Dummy_ISR},    //  6 - SPI2
    { 0xCC, Dummy_ISR},    //  5 - SPI1
    { 0xCC, Dummy_ISR},    //  4 - MCG Loss Lock
    { 0xCC, Dummy_ISR},    //  3 - Low VoltDetect
    { 0xCC, Dummy_ISR},    //  2 - IRQ
```

```
    { 0xCC, Dummy_ISR},     //  1 - SWI
};
```

d) Because several changes were made to the hid_main.c file, Appendix H, “hid_main.c for the MC9S08 CMX Example,” on page 69 provides the source code for the entire file.

4. Remove any references to a reset vector in the application. The bootloader uses the reset vector to point to its start-up function that determines if bootloader or application mode must start. If the application keeps a reset vector reference, this can cause an error because two vectors attempt to use the same location.

   A common method in CodeWarrior to set the reset vector is to declare the vector in the PRM file. The PRM file shows a reference like `VECTOR 0 _Startup`. This reference must be removed.

   Another common method for the reset vector is to provide a constant vector table in a project file, for example Vectors.c. This vector table is forced to the proper flash location for the vectors either by using a fixed address in the vector table declaration, or by using a segment declared in the PRM file. For the bootloader, this vector table must be removed. Interrupts are re-directed in the steps below.

   For this CMX example, the existing application used the PRM reset vector reference `VECTOR 0 _Startup`. Overwriting the PRM file has already implemented this change.

5. Change the existing interrupt vector table. The existing application places the interrupt vector table in the protected bootloader flash memory. These vectors need to be re-directed. The first step is to change references in the application to the existing interrupt service routines.

   If an interrupt vector number is used when the ISR is declared, CodeWarrior loads that interrupt vector with the address of the declared ISR. With the bootloader, these interrupt vector numbers need to be removed from the ISR declaration.

   Another common method for vector tables is to provide a constant vector table in a project file, for example Vectors.c. This vector table is forced to the proper flash location for the vectors either by using a fixed address in the vector table declaration, or by using a segment declared in the PRM file. For the bootloader, this vector table must be removed.

   For this CMX example, the only interrupt used is the USB interrupt. The interrupt vector number of the usb_it_handler ISR declaration needs to be removed. Change the line below in usb.c to the following

   ```
   interrupt 7 void usb_it_handler(void)
   ```

   to

   ```
   interrupt void usb_it_handler(void)
   ```

6. Compile and download the project into the board. The CMX example works as described in the DEMOJM Lab Supplement for the 8-bit Flexis JM60 document. Hold down the **PTG0** button while pressing and releasing the **Reset** button to enter bootloader mode. The bootloader drive appears in the PC. Copy another application’s S19 file onto the drive to see the application change. For example, use either MC9S08 S19 file in the Bootloader directory \S19 Files. Enter the bootloader mode again, and copy the CMX example S19 file back to the board using the file \usb-peripheral\projects\CodeWarrior\hc9S08jm60\hid-demo\bin\ Project.abs.s19.

4.7 Porting the Bootloader to Another Freescale Device

This bootloader was written for the three specified parts because they share the same USB peripheral registers and functionality for USB device. At the time this application note was written, the Freescale microcontrollers that share this peripheral include the Flexis JM family of MC9S08 and ColdFire V1 devices, the ColdFire V2 devices with part numbers MCF522xx, and the MC9S08JS family. Future Freescale devices are also planned with this same USB peripheral.

This application note provides the firmware for the three specified devices. However, the firmware can easily be ported to any of these devices sharing the same USB peripheral. When porting to these devices, pay particular attention to the items below:

- Change the macros defining the memory map in Bootloader_xx.h (i.e. Bootloader_V2.h). These macros define the flash and RAM address range of the part, where the flash protection range starts, where the User Entry jump vector is located, the flash page size, and address range for the USB buffers.
- Change the Linker file for the new part. Read the linker manual to understand the linker file. Then modify the linker file for the new memory map.
- Check the interrupts for the new part. Adjust the true interrupt vector table and the re-directed interrupt vector table for the new part.
- Verify part initialization in Bootloader_xx.c and in usr_entry_xx.c, including clock initialization, and flash clock initialization.
- As needed, change the forced entry method into bootloader mode in usr_entry_xx.c. For example, change the pin that is used during reset to enter bootloader mode.
- Verify the Flash Protection registers. For the MC9S08 and ColdFire V1, change the NVPROT register value in Bootloader_xx.c. For the ColdFire V2, change the CFMPROT register in mcf5225x_vectors.s.

5 Troubleshooting

The firmware provided has been thoroughly tested using the combination of devices, boards, and operating systems listed in Section 1, “Introduction,” on page 1. However, if the bootloader is modified or ported to another device, below is a list of common issues:

5.1 Reset Vector

If neither the bootloader nor the application start, the reset vector is likely to be responsible. To test it, use the debugger and place a breakpoint at the beginning of _Entry() in usr_entry_xx.c. Then reset the device holding the button to force bootloader mode. If the debugger hits this breakpoint, the reset vector is valid. If it does not hit this point, verify the reset vector points to _Entry(). The reset vector must be controlled by the bootloader.

5.2 User Entry Vector

If the bootloader executes as expected but the application never starts, the User Entry Vector is likely to be responsible. To test this, use the debugger and place a breakpoint in _Entry() in usr_entry_xx.c where

the bootloader jumps to the User Entry vector. Hold the button and reset to force the bootloader mode. Run to the breakpoint, then single-step through to see if the bootloader successfully jumps to the application. After the breakpoint, the bootloader must jump to the User Entry vector located at a specific absolute address in the flash. At that address is another JMP instruction that must jump to the application entry point.

If the bootloader initially jumps to the wrong absolute address, ensure the macro USER_ENTRY_ADDRESS in bootloader_xx.h is the correct address. If the vector is located at the correct absolute address and it jumps to the wrong address for the application entry, ensure the application loads the User Entry vector properly. In the provided bootloader examples, this is executed in main.c. The vector needs to be loaded properly in the application load. The bootloader code does not load this vector.

5.3 Interrupts

Re-directing the interrupt vectors is critical for the application to work properly. If the application starts properly, but does not function properly, interrupts are likely to be responsible. This is especially evident if the application worked fine before integrating with the bootloader. Symptoms of interrupt issues include the application code starting correctly but running away intermittently. To verify, disable interrupts and test if the application executes without running away.

If it is an interrupt issue, there are two aspects of the re-directed interrupt vectors to verify. First, the application is responsible for loading the interrupt vectors in the re-directed interrupt vector table. Please read Section Section 4.3, “Creating a New Project with the Bootloader,” on page 20 to understand how the application loads the re-directed interrupt vector table.

The second issue is to verify that the interrupt vectors have been re-directed. For devices that re-direct to RAM, ensure the VBR register is set in the application to point to RAM. For the devices that re-direct to the flash, ensure the vectors in the default interrupt vector location point to the re-directed vector. In the MC9S08, this re-direction is executed in the file Redirect_Vectors_S08.c.

5.4 Bootloader Erases Part of Itself

When modifying the bootloader or porting to a different device, it is possible the bootloader erases part of itself during a firmware update. Symptoms of this issue include the bootloader fails and runs away after the flash erase routine, or the bootloader runs successfully the first time but fails a second time.

There are two possible causes for this issue. The first is the flash protection range does not adequately protect the bootloader code in flash. Please read Section 2.8, “Flash Protection,” on page 6 on Flash Protection to ensure the bootloader is properly protected.

The second cause is that cross-calls are occurring, and the bootloader is calling functions in the application space. The application space gets erased during a firmware update, so cross-calling causes an issue. Cross-calling must be avoided. Please read Section 4.1, “Prevent Cross-Calling,” on page 16 on cross-calling and how to prevent it.

5.5 Linker Overlap Errors

When modifying the bootloader or porting to a different device, it is possible to get linker overlap errors when linking the project. Here are some possible reasons and solutions:

- Linker file has overlapping segments. When modifying the linker file for a new memory map, verify the segments do not overlap.
- Application, bootloader code, or RAM has outgrown its segment. If the bootloader code size exceeds its segment size, increase the segment size to a flash protection range large enough to protect the bootloader code. Then update for the new memory map as described in Section 4.7, “Porting the Bootloader to Another Freescale Device,” on page 39. For other outgrown issues, modify the memory map or move to a larger memory device as described in Section 4.7, “Porting the Bootloader to Another Freescale Device.
- Application interrupt vectors are still declared. If the application still declares an interrupt vector in the default vector table, it conflicts with the bootloader’s vector that should be at that location. The application interrupt vector needs to be located in the re-directed vector table. To fix this linker issue, load the application interrupt in the re-directed vector table as described in Section 4.3, “Creating a New Project with the Bootloader,” on page 20. Also, ensure the application does not reference the interrupt service routine in the default interrupt vector table that is controlled by the bootloader. These references include declaring the interrupt service routine with the interrupt vector number as shown below. A vector table is also included with the application that loads the default interrupt vector table and points to the ISRs.

  ```
  interrupt VectorNumber_Vftm2ovf void timer(void) {
  ```

  The code above is an example of how to declare an interrupt service routine in CodeWarrior. This declaration tells the linker to load the default vector table at the vector number with the vector to the ISR. In this case, VectorNumber_Vftm2ovf is a macro specifying the vector number and timer is the ISR function name. These vector numbers in the ISR declaration need to be removed because they overlap with the bootloader vector table.
- Application reset vector is still declared. If the application still declares the reset vector, it conflicts with the bootloader’s vector that should be at that location. To fix this linker issue, ensure the application does not reference the application entry point in the reset vector that is controlled by the bootloader. These references include specifying the application entry point in the linker file as shown below. A vector table is also included with the application that loads the reset vector and points to the application.

  ```
  VECTOR 0 _Startup
  ```

  The code above is an example of how to declare the reset vector in an MC9S08 PRM file in CodeWarrior. This command tells the linker to load the reset vector with the vector `_Startup`. In this case, `_Startup()` is the application entry point function. This reset vector declaration needs to be removed because it overlaps with the bootloader reset vector.
- Compiler optimization settings. It is possible when integrating an application using the bootloader library that the bootloader compile into binary code larger than the linker segment for the bootloader code. This can occur because the application project uses less optimization for the compiler than the original bootloader project. Change the compiler optimization settings to force the bootloader to fit in the protected flash segment, or change the memory map as described in Section 4.7, “Porting the Bootloader to Another Freescale Device,” on page 39.

5.6 USB Drive does not Appear in the Host

If the USB cable is connected between the host computer and the device the USB drive must then be visible in the host a few seconds after entering bootloader mode. If not, ensure the device is properly powered and connected. Also, ensure the hardware is configured for USB device operation similar to the default jumper settings of the demo board. Refresh the host application to check for the new drive. In Windows, check the Device Manager to see if the USB device has enumerated properly. Check that the bootloader starts properly as described in Section 5.1, “Reset Vector,” on page 39. Check that the bootloader has not corrupted itself as described in Section Section 5.4, “Bootloader Erases Part of Itself,” on page 40. Reboot the host computer. If none of these solutions appear to fix the issue, use the debugger or a USB analyzer to debug the USB enumeration to see where the issue occurs.

5.7 Bootloader Status Error before a File Transfer

After the bootloader drive enumerates in the host computer, the drive shows the READY.TXT file. Ensure this file is visible before transferring an S19 file to the drive. If there is already an error status file in the drive before transferring any file, this can be caused by settings within the host operating system. For example, Macintosh computers can try to write a file to removable drives after the drive enumerates. The bootloader assumes any file written to the root directory is an S19 file, and starts parsing the file. If a file other than an S-record file is written to the bootloader drive, it will cause errors. To fix this change the operating system settings to prevent it from writing files itself to the drive.

5.8 Flash Erase/Program Generates Errors

If the status file FFAILED.TXT is present in the bootloader drive after a firmware update attempt, this signals there was an error in erasing or programming part of the flash. This error is usually caused because an invalid flash address is erased/programmed, or the flash clock is not in the specified frequency range. Verify the memory map as described in Section 4.7, “Porting the Bootloader to Another Freescale Device,” on page 39. Also, verify the flash clock divider in Bootloader_Main() in Bootloader_xx.c. If the issue continues to persist, use the debugger with breakpoints to see at what point the error occurs, and verify the flash address received by the bootloader.

5.9 Bootloader and Application Cross-Call Issues

Cross-calling between the bootloader and application is hazardous. It can cause symptoms including the bootloader corrupting itself after a firmware update, or a new application fails to work when updated on a device with an older revision of the bootloader. Please refer to Section 4.1, “Prevent Cross-Calling,” on page 16 to prevent cross-calling.

6 Conclusion

Freescale’s USB Mass Storage Device bootloader allows for very easy firmware updates. It does not require a driver, does not require any software on the host computer, and can be executed by anyone. This bootloader also works with a broad selection of Freescale microcontrollers. The information in this application note allows any application to be integrated with the bootloader.

Appendix A mcf5222x_vectors.s for the Coldfire V2 CMX Example

Section 4.4, "Adding the Bootloader to the Existing ColdFireV2 Project," on page 24 gives detailed steps to start with the ColdFire V2 CMX Example and add the bootloader. The mcf5222x_vectors.s file required several modifications for this project. Below is the text used for the new mcf5222x_vectors.s file for this example with the bootloader. This file is located at the following path under the CMX Installation root directory:

\usb-peripheral\projects\CodeWarrior\mcf52223\mcf5222x_vectors.s

```
/*****************************************************************************
 This file contains the vector table and the startup code executed out of
 reset.
 *****************************************************************************/
        .global VECTOR_TABLE
        .global _VECTOR_TABLE
        .global start
        .global _start
        .global d0_reset
        .global _d0_reset
        .global d1_reset
        .global _d1_reset

        .extern __IPSBAR
        .extern __SRAM
        .extern __FLASH
        .extern __SP_INIT

    .extern _irq_handler02
    .extern _irq_handler03
    .extern _irq_handler04
    .extern _irq_handler05
    .extern _irq_handler06
    .extern _irq_handler07
    .extern _irq_handler08
    .extern _irq_handler09
    .extern _irq_handler0a
    .extern _irq_handler0b
    .extern _irq_handler0c
    .extern _irq_handler0d
    .extern _irq_handler0e
    .extern _irq_handler0f
    .extern _irq_handler10
    .extern _irq_handler11
    .extern _irq_handler12
    .extern _irq_handler13
    .extern _irq_handler14
    .extern _irq_handler15
    .extern _irq_handler16
    .extern _irq_handler17
    .extern _irq_handler18
    .extern _irq_handler19
    .extern _irq_handler1a
    .extern _irq_handler1b
    .extern _irq_handler1c
```

```
    .extern _irq_handler1d
    .extern _irq_handler1e
    .extern _irq_handler1f
    .extern _irq_handler20
    .extern _irq_handler21
    .extern _irq_handler22
    .extern _irq_handler23
    .extern _irq_handler24
    .extern _irq_handler25
    .extern _irq_handler26
    .extern _irq_handler27
    .extern _irq_handler28
    .extern _irq_handler29
    .extern _irq_handler2a
    .extern _irq_handler2b
    .extern _irq_handler2c
    .extern _irq_handler2d
    .extern _irq_handler2e
    .extern _irq_handler2f
    .extern _irq_handler30
    .extern _irq_handler31
    .extern _irq_handler32
    .extern _irq_handler33
    .extern _irq_handler34
    .extern _irq_handler35
    .extern _irq_handler36
    .extern _irq_handler37
    .extern _irq_handler38
    .extern _irq_handler39
    .extern _irq_handler3a
    .extern _irq_handler3b
    .extern _irq_handler3c
    .extern _irq_handler3d
    .extern _irq_handler3e
    .extern _irq_handler3f
    .extern _irq_handler40
    .extern _irq_handler41
    .extern _irq_handler42
    .extern _irq_handler43
    .extern _irq_handler44
    .extern _irq_handler45
    .extern _irq_handler46
    .extern _irq_handler47
    .extern _irq_handler48
    .extern _irq_handler49
    .extern _irq_handler4a
    .extern _irq_handler4b
    .extern _irq_handler4c
    .extern _irq_handler4d
    .extern _irq_handler4e
    .extern _irq_handler4f
    .extern _irq_handler50
    .extern _irq_handler51
    .extern _irq_handler52
    .extern _irq_handler53
    .extern _irq_handler54
    .extern _irq_handler55
```

```
    .extern _irq_handler56
    .extern _irq_handler57
    .extern _irq_handler58
    .extern _irq_handler59
    .extern _irq_handler5a
    .extern _irq_handler5b
    .extern _irq_handler5c
    .extern _irq_handler5d
    .extern _irq_handler5e
    .extern _irq_handler5f
    .extern _irq_handler60
    .extern _irq_handler61
    .extern _irq_handler62
    .extern _irq_handler63
    .extern _irq_handler64
    .extern _irq_handler65
    .extern _irq_handler66
    .extern _irq_handler67
    .extern _irq_handler68
    .extern _irq_handler69
    .extern _irq_handler6a
    .extern _irq_handler6b
    .extern _irq_handler6c
    .extern _irq_handler6d
    .extern _irq_handler6e
    .extern _irq_handler6f
    .extern _irq_handler70
    .extern _irq_handler71
    .extern _irq_handler72
    .extern _irq_handler73
    .extern _irq_handler74
    .extern _irq_handler75
    .extern _irq_handler76
    .extern _irq_handler77
    .extern _irq_handler78
    .extern _irq_handler79
    .extern _irq_handler7a
    .extern _irq_handler7b
    .extern _irq_handler7c
    .extern _irq_handler7d
    .extern _irq_handler7e
    .extern _irq_handler7f
    .extern _low_level_init
    .extern _usb_it_handler
    .extern _uart_it_handler
    .extern _main
        .bss
d0_reset:
_d0_reset:.long   0
d1_reset:
_d1_reset:.long   0

        .text

/*********************************************************************/
_start:
        /* Save off reset values of D0 and D1 */
```

```
        move.l   d0,d6
        move.l   d1,d7

        /* Initialize RAMBAR1: locate SRAM and validate it */
        move.l   #__SRAM,d0
        add.l    #0x21,d0
        movec    d0,RAMBAR1

        /* Locate Stack Pointer */
        move.l   #__SP_INIT,sp

        /* Initialize IPSBAR */
        move.l   #__IPSBAR,d0
        add.l    #0x1,d0
        move.l   d0,0x40000000

        /* Initialize FLASHBAR */
        /* Flash at address 0x0. */
        move.l    #0,d0
        add.l    #0x61,d0
        movec    d0,RAMBAR0

        /* Locate Stack Pointer */
        move.l   #__SP_INIT,sp

        /* Save off intial D0 and D1 to RAM */
        move.l   d6,d0_reset
        move.l   d7,d1_reset

        /* Common startup code */
        jsr      _low_level_init

        /* Jump to the main process */
        jsr              _main

        bra              .
        nop
        nop
        halt

        .end
```

Appendix B LCF File for ColdFire V2 CMX Example

Section 4.4, “Adding the Bootloader to the Existing ColdFireV2 Project,” on page 24 gives detailed steps to start with the ColdFire V2 CMX Example and add the bootloader. The LCF file for this project required several modifications. Below is the text used for this example for the new LCF file with the bootloader. This LCF file is located at the following path under the CMX Installation root directory:

\usb-peripheral\projects\CodeWarrior\mcf52223\m522x_evb_flash.lcf

```
#/*****************************************************************************
#*
#*  (c) copyright Freescale Semiconductor 2008
#*  ALL RIGHTS RESERVED
#*
#*  File Name: mcf52259_bl_flash.lcf
#*
#*  Purpose: This file is for a USB Mass-Storage Device bootloader.  This file
#*           is the linker command file for the flash target.
#*
#*  Assembler:  Codewarrior for ColdFire V7.1
#*
#*  Version:  1.1
#*
#*
#*  Author: Derek Snell
#*
#*  Location: Indianapolis, IN. USA
#*
#* UPDATED HISTORY:
#*
#* REV   YYYY.MM.DD  AUTHOR        DESCRIPTION OF CHANGE
#* ---   ----------  ------        ---------------------
#* 1.0   2008.06.10  Derek SnellInitial version
#* 1.1   2009.06.01  Derek SnellImproved for easier application integration
#*
#*
#*****************************************************************************/

#/* Freescale  is  not  obligated  to  provide  any  support, upgrades or new */
#/* releases  of  the Software. Freescale may make changes to the Software at */
#/* any time, without any obligation to notify or provide updated versions of */
#/* the  Software  to you. Freescale expressly disclaims any warranty for the */
#/* Software.  The  Software is provided as is, without warranty of any kind, */
#/* either  express  or  implied,  including, without limitation, the implied */
#/* warranties  of  merchantability,  fitness  for  a  particular purpose, or */
#/* non-infringement.  You  assume  the entire risk arising out of the use or */
#/* performance of the Software, or any systems you design using the software */
#/* (if  any).  Nothing  may  be construed as a warranty or representation by */
#/* Freescale  that  the  Software  or  any derivative work developed with or */
#/* incorporating  the  Software  will  be  free  from  infringement  of  the */
#/* intellectual property rights of third parties. In no event will Freescale */
#/* be  liable,  whether in contract, tort, or otherwise, for any incidental, */
#/* special,  indirect, consequential or punitive damages, including, but not */
#/* limited  to,  damages  for  any loss of use, loss of time, inconvenience, */
#/* commercial loss, or lost profits, savings, or revenues to the full extent */
#/* such  may be disclaimed by law. The Software is not fault tolerant and is */
#/* not  designed,  manufactured  or  intended by Freescale for incorporation */
```

```
#/* into  products intended for use or resale in on-line control equipment in */
#/* hazardous, dangerous to life or potentially life-threatening environments */
#/* requiring  fail-safe  performance,  such  as  in the operation of nuclear */
#/* facilities,  aircraft  navigation  or  communication systems, air traffic */
#/* control,  direct  life  support machines or weapons systems, in which the */
#/* failure  of  products  could  lead  directly to death, personal injury or */
#/* severe  physical  or  environmental  damage  (High  Risk Activities). You */
#/* specifically  represent and warrant that you will not use the Software or */
#/* any  derivative  work of the Software for High Risk Activities.           */
#/* Freescale  and the Freescale logos are registered trademarks of Freescale */
#/* Semiconductor Inc.                                                        */
#****************************************************************************/

MEMORY
{
         vectorflash(RX) : ORIGIN = 0x00000000, LENGTH = 0x00000418
         bootcode    (RX): ORIGIN = 0x00000420, LENGTH = 0x00003BE0
         #flash    (RX)    : ORIGIN = 0x00000420, LENGTH = 0x0007FC00
         flash    (RX)    : ORIGIN = 0x00004010, LENGTH = 0x0007BFF0
         vectorram(RWX) : ORIGIN = 0x20000000, LENGTH = 0x00000400
         ram               (RWX)   : ORIGIN = 0x20000400, LENGTH = 0x0000FC00
         bootsram(RWX)   : ORIGIN = 0x20000600, LENGTH = 0x0000FA00
         ipsbar   (RWX)   : ORIGIN = 0x40000000, LENGTH = 0x0
}

SECTIONS
{
         __SRAM_SIZE       = SIZEOF(ram);
         ___SRAM_SIZE      = __SRAM_SIZE;

         __FLASH_SIZE      = SIZEOF(flash);
         ___FLASH_SIZE     = __FLASH_SIZE;
         ___FLASHBAR_SIZE  = 0x00080000;
         ___RAMBAR_SIZE    = 0x0000FFFF;

         .ipsbar :
         {
                  __IPSBAR          = .;
                  ___IPSBAR         = .;
         } > ipsbar

         .vectorflash :
         {
                  ___FLASHBAR       = .;
         __FLASH                    = .;
         ___FLASH          = .;
                 mcf5225x_vectors.s(.text)
         } > vectorflash

         .vectorram :
         {
                  ___RAMBAR         = .;
                  __SRAM            = .;
                  ___SRAM                    = __SRAM;
                  ___VECTOR_RAM = .;
         } > vectorram
```

```
    .bootcode: {
            ___Boot_START= .;
            usr_entry_V2.c    (.text)
            usr_entry_V2.c    (.rodata)
            V2_Bootloader.lib (.text)
            V2_Bootloader.lib (.rodata)
            . = ALIGN (0x4);
            ___Boot_END       = .;
    } > bootcode

    .bootsram: {
            ___Boot_RAM_START= .;
            usr_entry_V2.c    (.bss)
            V2_Bootloader.lib(.bss)
            . = ALIGN (0x4);
            ___Boot_RAM_END= .;
    } > bootsram

/* Constant objects. */
    .flash :
    {
            *(.text)
            *(.rodata)
            .                           = ALIGN(0x10);
            ___COPY_START     = .;
            ___DATA_ROM       = .;
            __DATA_ROM        = .;
            ___ROM_AT         = .;
    } > flash

/* Inicialised data. This needs to be copied to RAM runtime. It will be
   allocated after .flash. */
    .data : AT(___COPY_START)
    {
            .               = ALIGN(0x10);
            ___COPY_DST       = .;
            ___DATA_RAM     = .;
            __DATA_RAM        = .;
            ___sinit__        = .;
            STATICINIT
            *(.data)
    *(.relocate_code)
            .             = ALIGN (0x10);
            ___DATA_END     = .;

            __START_SDATA  = .;
            *(.sdata)
            ___COPY_END       = .;
            ___DATA_END       = .;
            __SDA_BASE        = .;
            .                           = ALIGN(512);
    } > ram

    .bss :
    {
            .               = ALIGN(4);
            ___BSS_START      = .;
```

```
                __BSS_START       = .;
                __START_SBSS      = .;
                ___ZERO_START     = .;
                *(.sbss)
                *(SCOMMON)
                __END_SBSS        = .;
                __START_BSS       = .;
                *(.bss)
                *(COMMON)
                ___ZERO_END       = .;
                __END_BSS         = .;
                ___BSS_END        = .;
                __BSS_END         = .;

        /* Buffer descriptor base address
           shall be aligned to 512 byte boundary.
           Size shall be 512 bytes. */
        .                 = ALIGN(512);
        __BDT_BASE= .;
        .                 = . + 512;
        __BDT_END         = .;

                ___SP_SIZE      = 0x0800;
                __SP_SIZE         = ___SP_SIZE;
                ___SP_END         = ___RAMBAR + ___RAMBAR_SIZE + 1 - ___SP_SIZE;
                __SP_END          = ___SP_END;
                ___SP_INIT        = ___SP_END + ___SP_SIZE;
                __SP_INIT         = ___SP_INIT;
                ___SP_AFTER_RESET = ___RAMBAR + ___RAMBAR_SIZE - 4;
        } >> ram

        _romp_at = ___ROM_AT + SIZEOF(.data);
        .romp : AT(_romp_at)
        {
                __S_romp = _romp_at;
                WRITEW(___ROM_AT);
                WRITEW(ADDR(.data));
                WRITEW(SIZEOF(.data));
                WRITEW(0);
                WRITEW(0);
                WRITEW(0);
        }
}
```

Appendix C hid_main.c for ColdFire V2 CMX Example

Section 4.4, “Adding the Bootloader to the Existing ColdFireV2 Project,” on page 24 gives detailed steps to start with the ColdFire V2 CMX Example and add the bootloader. The hid_main.c file for this project required several modifications. Below is the text used for the new hid_main.c file for this example with the bootloader. This file is located at the following path under the CMX Installation root directory:

\usb-peripheral\src\mcf5222x\hid-demo\hid_main.c

```
/*****************************************************************************
 *
 *            Copyright (c) 2006-2007 by CMX Systems, Inc.
 *
 * This software is copyrighted by and is the sole property of
 * CMX.  All rights, title, ownership, or other interests
 * in the software remain the property of CMX.  This
 * software may only be used in accordance with the corresponding
 * license agreement.  Any unauthorized use, duplication, transmission,
 * distribution, or disclosure of this software is expressly forbidden.
 *
 * This Copyright notice may not be removed or modified without prior
 * written consent of CMX.
 *
 * CMX reserves the right to modify this software without notice.
 *
 * CMX Systems, Inc.
 * 12276 San Jose Blvd. #511
 * Jacksonville, FL 32223
 * USA
 *
 * Tel:  (904) 880-1840
 * Fax:  (904) 880-1632
 * http: www.cmx.com
 * email: cmx@cmx.com
 *
 *****************************************************************************/
#include "usb-drv/usb.h"
#include "target.h"
#include "hid_mouse.h"
#include "hid_kbd.h"
#include "hid_generic.h"
/*****************************************************************************
 ************************* Macro definitions *********************************
 *****************************************************************************/
/* none */

/*****************************************************************************
 ************************* Function predefinitions. **************************
 *****************************************************************************/
/* none */

/*****************************************************************************
 ************************* Global variables **********************************
 *****************************************************************************/
/* none */

/*****************************************************************************
```

```
 ************************* Module variables ******************************
 *************************************************************************/
/* none */

/*************************************************************************
 ************************* Function definitions **************************
 *************************************************************************/

// Added for Bootloader
#include "Bootloader_V2.h"

extern asm void start(void);
void usb_it_handler(void);

const byte _UserEntry[] @ USER_ENTRY_ADDRESS = {
  0x4E,
  0x71,
  0x4E,
  0xF9            //asm NOP(0x4E71), asm JMP(0x4EF9)
};

void  (* const _UserEntry2[])()@(USER_ENTRY_ADDRESS+4)= {
  start,
};


int main()
{
  hcc_u8 tmp;

  // Add redirected USB interrupt vector to RAM
  __VECTOR_RAM[64 + 53] = (hcc_u32) usb_it_handler;

  hw_init();

  // Re-Direct interrupt vectors to RAM
  mcf5xxx_wr_vbr((hcc_u32) __VECTOR_RAM);

  Usb_Vbus_Off();

  /* See in what mode should we run. */
  tmp = (hcc_u8)(SW1_ACTIVE() ? 1 : 0);
  tmp |= (hcc_u8)(SW2_ACTIVE() ? 2 : 0);

  /* Start the right HID application. */
  switch(tmp)
  {
  case 0:
  default:
    usb_cfg_init();
    set_mode(dm_mouse);
    (void)hid_mouse();
    break;
  case 1:
    usb_cfg_init();
    set_mode(dm_kbd);
    hid_kbd();
```

```
    break;
  case 2:
    usb_cfg_init();
    set_mode(dm_generic);
    hid_generic();
    break;
  }
  /* We will never get here. */
  return 0;
}
/****************************** END OF FILE **********************************/
```

Appendix D usr_entry_V2.c for ColdFire V2 CMX Example

Section 4.4, “Adding the Bootloader to the Existing ColdFireV2 Project,” on page 24 gives detailed steps to start with the ColdFire V2 CMX Example and add the bootloader. The usr_entry_V2.c file for this project required several modifications. Below is the text used for the new usr_entry_V2.c file for this example with the bootloader. This file is located at the following path under the CMX Installation root directory:

\usb-peripheral\src\mcf5222x\hid-demo\usr_entry_V2.c

```
/*****************************************************************************
*
*  (c) copyright Freescale Semiconductor 2008
*  ALL RIGHTS RESERVED
*
*  File Name: usr_entry.c
*
*  Purpose: This file is for a USB Mass-Storage Device bootloader.  This file
*           is the initial entry point for the bootloader
*
*  Assembler:  Codewarrior for ColdFire V7.0
*
*  Version:  1.3
*
*
*  Author: Derek Snell
*
*  Location: Indianapolis, IN. USA
*
* UPDATED HISTORY:
*
* REV   YYYY.MM.DD  AUTHOR        DESCRIPTION OF CHANGE
* ---   ----------  ------        ---------------------
* 1.0   2008.06.10  Derek Snell   Initial version
* 1.1   2008.06.30  Derek SnellPorted from JM128
* 1.2    2008.09.25David Seymour Fixed jump vector in RAM target
* 1.3    2008.09.26Derek SnellAdded check for User entry vector in flash
*
*
*****************************************************************************/

/* Freescale  is  not  obligated  to  provide  any  support, upgrades or new */
/* releases  of  the Software. Freescale may make changes to the Software at */
/* any time, without any obligation to notify or provide updated versions of */
/* the  Software  to you. Freescale expressly disclaims any warranty for the */
/* Software.  The  Software is provided as is, without warranty of any kind, */
/* either  express  or  implied,  including, without limitation, the implied */
/* warranties  of  merchantability,  fitness  for  a  particular purpose, or */
/* non-infringement.  You  assume  the entire risk arising out of the use or */
/* performance of the Software, or any systems you design using the software */
/* (if  any).  Nothing  may  be construed as a warranty or representation by */
/* Freescale  that  the  Software  or  any derivative work developed with or */
/* incorporating  the  Software  will  be  free  from  infringement  of  the */
/* intellectual property rights of third parties. In no event will Freescale */
/* be  liable,  whether in contract, tort, or otherwise, for any incidental, */
/* special,  indirect, consequential or punitive damages, including, but not */
```

```
/* limited  to,  damages  for  any loss of use, loss of time, inconvenience, */
/* commercial loss, or lost profits, savings, or revenues to the full extent */
/* such  may be disclaimed by law. The Software is not fault tolerant and is */
/* not  designed,  manufactured  or  intended by Freescale for incorporation */
/* into  products intended for use or resale in on-line control equipment in */
/* hazardous, dangerous to life or potentially life-threatening environments */
/* requiring  fail-safe  performance,  such  as  in the operation of nuclear */
/* facilities,  aircraft  navigation  or  communication systems, air traffic */
/* control,  direct  life  support machines or weapons systems, in which the */
/* failure  of  products  could  lead  directly to death, personal injury or */
/* severe  physical  or  environmental  damage  (High  Risk Activities). You */
/* specifically  represent and warrant that you will not use the Software or */
/* any  derivative  work of the Software for High Risk Activities.           */
/* Freescale  and the Freescale logos are registered trademarks of Freescale */
/* Semiconductor Inc.                                                        */
/*****************************************************************************/

#include "Bootloader_V2.h"
#include "mcf5222x_reg.h"

extern asm void start(void);
extern byte __SRAM[];


void _Entry(void)
{
  byte i;
  dword* UserEntryCheck;

  MCF_GPIO_DDRNQ &= ~MCF_GPIO_DDRNQ_DDRNQ5;// PNQ5 as Input
  MCF_GPIO_PNQPAR &= ~(MCF_GPIO_PNQPAR_PNQPAR5(3));// set PNQ5 as GPIO

  // delay some time for GPIO stable
  for(i=0;i<3;i++) {

    asm(nop);
  }

  // If SW1 on PORTNQ5 not pressed, jump to user application
  if(MCF_GPIO_SETNQ & MCF_GPIO_SETNQ_SETNQ5)
  {
        if ((dword)_Entry >= (dword)(__SRAM)) // Test to see if running in RAM or Flash
{
asm (JMP start);      // jump to user entry
}
else {
                UserEntryCheck = (dword*)USER_ENTRY_ADDRESS;
                if(*UserEntryCheck == 0x4E714EF9) // check there is a valid jump op-code
                {
                        UserEntryCheck++;// increment pointer to next long word
                        if(*UserEntryCheck != 0xFFFFFFFF) // check there is a valid vector
                        {
                                asm (JMP USER_ENTRY_ADDRESS);      // jump to user entry
                        }
                }
        }
  }
```

```
  // Disable Software Watchdog Timer
  MCF_SCM_CWCR = 0;


  // Enable debug
  MCF_GPIO_PDDPAR = 0x0F;

  MCF_CLOCK_OCLR = 0xC0;   //turn on crystal

  // delay some time for oscillator stable
  for(i=0;i<255;i++) {

                asm(nop);
  }
  MCF_CLOCK_CCLR = 0x00;    //switch to crystal
  MCF_CLOCK_OCHR = 0x00; //turn off relaxation osc

  // Enable on-chip modules to access internal SRAM
  MCF_SCM_RAMBAR = (0
         | MCF_SCM_RAMBAR_BA(MIN_RAM1_ADDRESS)
         | MCF_SCM_RAMBAR_BDE);

  Bootloader_Main();
}
```

Appendix E LCF File for ColdFire V1 CMX Example

Section 4.5, “Adding Bootloader to the Existing ColdFire V1 Project,” on page 29 gives detailed steps to start with the ColdFire V1 CMX Example and add the bootloader. The LCF file for this project required several modifications. Below is the text used for the new LCF file for this example with the bootloader. This LCF file is located at the following path under the CMX Installation root directory:

\usb-peripheral\projects\CodeWarrior-6.x\mcf51xx\hid-demo\prm\Project_flash.lcf

```
# Sample Linker Command File for CodeWarrior for ColdFire MCF51JM128

# Memory ranges

MEMORY {
   bootcode  (RX)   : ORIGIN = 0x000410, LENGTH = 0x00001BF0
   code      (RX)   : ORIGIN = 0x002200, LENGTH = 0x0001DE00
   #code     (RX)   : ORIGIN = 0x000410, LENGTH = 0x0001FBF0
   #userram  (RWX)  : ORIGIN = 0x800000, LENGTH = 0x00004000
   vectorram (RWX)  : ORIGIN = 0x800000, LENGTH = 0x00000200
   userram   (RWX)  : ORIGIN = 0x800200, LENGTH = 0x00003E00
   bootsram  (RWX)  : ORIGIN = 0x8006E0, LENGTH = 0x00003B60
}

SECTIONS {

   __IPSBAR  =  0xFFFF8000;
    __VECTOR_TABLE_BASE = 0;


# Heap and Stack sizes definition
  ___heap_size     = 0x0000;
  ___stack_size    = 0x400;

# MCF51JM128 Derivative Memory map definitions from linker command files:
# ___RAM_ADDRESS, ___RAM_SIZE, ___FLASH_ADDRESS, ___FLASH_SIZE linker
# symbols must be defined in the linker command file.

# 16 Kbytes Internal SRAM
   ___RAM_ADDRESS = 0x00800000;
   ___RAM_SIZE    = 0x00004000;

# 128 KByte Internal Flash Memory
   ___FLASH_ADDRESS  = 0x00000000;
   ___FLASH_SIZE     = 0x00020000;

  .userram     : {} > userram
  .code     : {} > code

  .bootcode:
        {
                ___Boot_START   = .;
                Bootloader_V1.c (.text)
                Bootloader_V1.c (.rodata)
                usr_entry_V1.c  (.text)
                ParseS19.c      (.text)
                ParseS19.c      (.rodata)
                SCSI_Process.c  (.text)
```

```
                SCSI_Process.c  (.rodata)
                FAT16.c         (.text)
                FAT16.c         (.rodata)
                exceptions.c    (.text)

                . = ALIGN (0x4);
                ___Boot_END     =.;
} > bootcode

.bootsram:
       {
                ___Boot_RAM_START   = .;
                Bootloader_V1.c (.bss)
                Bootloader_V1.c (.sbss)
                usr_entry_V1.c (.bss)
                usr_entry_V1.c (.sbss)
                ParseS19.c        (.bss)
                ParseS19.c        (.sbss)
                SCSI_Process.c    (.bss)
                SCSI_Process.c    (.sbss)
                FAT16.c           (.bss)
                FAT16.c           (.sbss)

                . = ALIGN (0x4);
                ___Boot_RAM_END     =.;
} > bootsram


.text :
{
  *(.text)
  . = ALIGN (0x4);
  *(.rodata)
  . = ALIGN (0x4);
  ___ROM_AT = .;
  ___DATA_ROM = .;
} >> code

.data : AT(___ROM_AT)
{
  ___DATA_RAM = .;
  . = ALIGN(0x4);
  *(.exception)
  . = ALIGN(0x4);
  __exception_table_start__ = .;
  EXCEPTION
  __exception_table_end__ = .;

  ___sinit__ = .;
    STATICINIT
  __START_DATA = .;

  *(.data)
  . = ALIGN (0x4);
  __END_DATA = .;

  __START_SDATA = .;
```

```
  *(.sdata)
  . = ALIGN (0x4);
  __END_SDATA = .;

  ___DATA_END = .;
  __SDA_BASE = .;
  . = ALIGN (0x4);
} >> userram

.bss :
{
  ___BSS_START = .;
  __START_SBSS = .;
  *(.sbss)
  . = ALIGN (0x4);
  *(SCOMMON)
  __END_SBSS = .;

  __START_BSS = .;
  *(.bss)
  . = ALIGN (0x4);
  *(COMMON)
  __END_BSS = .;
  ___BSS_END = .;

  . = ALIGN(0x4);
} >> userram

.custom :
{
  ___HEAP_START       = .;
  ___heap_addr        = ___HEAP_START;
  ___HEAP_END         = ___HEAP_START + ___heap_size;
  . = ___HEAP_END;
     /* Buffer descriptor base address
       shall be aligned to 512 byte boundary.
       Size shall be 512 bytes. */
  .                   = ALIGN(512);
  __BDT_BASE    = .;
  .               = . + 512;
  __BDT_END       = .;

  ___SP_INIT        = ___RAM_ADDRESS + ___RAM_SIZE;
  ___SP_END         = ___SP_INIT - ___stack_size;

} >> userram

__SP_INIT             = ___SP_INIT;

___SP_AFTER_RESET     = __SP_INIT;

_romp_at = ___ROM_AT + SIZEOF(.data);
.romp : AT(_romp_at)
{
  __S_romp = _romp_at;
  WRITEW(___ROM_AT);
  WRITEW(ADDR(.data));
```

```
    WRITEW(SIZEOF(.data));
    WRITEW(0);
    WRITEW(0);
    WRITEW(0);
  }
}
```

Appendix F hid_main.c for the ColdFire V1 CMX Example

Section 4.5, “Adding Bootloader to the Existing ColdFire V1 Project,” on page 29 gives detailed steps to start with the ColdFire V1 CMX example and to add the bootloader. The hid_main.c file for this project required several modifications. Below is the text used for the new hid_main.c file for this example with the bootloader. This file is located at the following path under the CMX Installation root directory:

\usb-peripheral\src\mcf51xx\hid-demo\hid_main.c

```
/*****************************************************************************
 *
 *            Copyright (c) 2006-2007 by CMX Systems, Inc.
 *
 * This software is copyrighted by and is the sole property of
 * CMX.  All rights, title, ownership, or other interests
 * in the software remain the property of CMX.  This
 * software may only be used in accordance with the corresponding
 * license agreement.  Any unauthorized use, duplication, transmission,
 * distribution, or disclosure of this software is expressly forbidden.
 *
 * This Copyright notice may not be removed or modified without prior
 * written consent of CMX.
 *
 * CMX reserves the right to modify this software without notice.
 *
 * CMX Systems, Inc.
 * 12276 San Jose Blvd. #511
 * Jacksonville, FL 32223
 * USA
 *
 * Tel:  (904) 880-1840
 * Fax:  (904) 880-1632
 * http: www.cmx.com
 * email: cmx@cmx.com
 *
 *****************************************************************************/
#include "mcf51xx_reg.h"
#include "../usb-drv/usb.h"
#include "target.h"
#include "hid_mouse.h"
#include "hid_kbd.h"
#include "hid_generic.h"

//#define DEBUG
//#define EVB
/*****************************************************************************
 ************************* Macro definitions *********************************
 *****************************************************************************/
/* none */

/*****************************************************************************
 ************************* Function predefinitions. **************************
 *****************************************************************************/
/* none */

/*****************************************************************************
 ************************* Global variables **********************************
```

```
 ****************************************************************************/
/* none */

/****************************************************************************
 ************************* Module variables ********************************
 ****************************************************************************/
/* none */

/****************************************************************************
 ************************* Function definitions ****************************
 ****************************************************************************/

// Added for Bootloader
#include "Bootloader_V1.h"

extern asm void _startup(void);

const byte _UserEntry[] @ USER_ENTRY_ADDRESS = {
  0x4E,
  0x71,
  0x4E,
  0xF9           //asm NOP(0x4E71), asm JMP(0x4EF9)
};

void  (* const _UserEntry2[])()@(USER_ENTRY_ADDRESS+4)=
{
  _startup,
};

__interrupt void dummy_ISR(void) {}

typedef void (* pFun)(void);

void  (* const RAM_vector[])()@REDIRECT_VECTORS= {

    (pFun)&dummy_ISR,              // vector_0  INITSP
    (pFun)&dummy_ISR,             // vector_1  INITPC
    (pFun)&dummy_ISR,             // vector_2  Vaccerr
    (pFun)&dummy_ISR,             // vector_3  Vadderr
    (pFun)&dummy_ISR,             // vector_4  Viinstr
    (pFun)&dummy_ISR,             // vector_5  VReserved5
    (pFun)&dummy_ISR,             // vector_6  VReserved6
    (pFun)&dummy_ISR,             // vector_7  VReserved7
    (pFun)&dummy_ISR,             // vector_8  Vprviol
    (pFun)&dummy_ISR,             // vector_9  Vtrace
    (pFun)&dummy_ISR,             // vector_10 Vunilaop
    (pFun)&dummy_ISR,             // vector_11 Vunilfop
    (pFun)&dummy_ISR,             // vector_12 Vdbgi
    (pFun)&dummy_ISR,             // vector_13 VReserved13
    (pFun)&dummy_ISR,             // vector_14 Vferror
    (pFun)&dummy_ISR,             // vector_15 VReserved15
    (pFun)&dummy_ISR,             // vector_16 VReserved16
    (pFun)&dummy_ISR,             // vector_17 VReserved17
    (pFun)&dummy_ISR,             // vector_18 VReserved18
    (pFun)&dummy_ISR,             // vector_19 VReserved19
    (pFun)&dummy_ISR,             // vector_20 VReserved20
    (pFun)&dummy_ISR,             // vector_21 VReserved21
```

```
(pFun)&dummy_ISR,              // vector_22 VReserved22
(pFun)&dummy_ISR,              // vector_23 VReserved23
(pFun)&dummy_ISR,              // vector_24 Vspuri
(pFun)&dummy_ISR,              // vector_25 VReserved25
(pFun)&dummy_ISR,              // vector_26 VReserved26
(pFun)&dummy_ISR,              // vector_27 VReserved27
(pFun)&dummy_ISR,              // vector_28 VReserved28
(pFun)&dummy_ISR,              // vector_29 VReserved29
(pFun)&dummy_ISR,              // vector_30 VReserved30
(pFun)&dummy_ISR,              // vector_31 VReserved31
(pFun)&dummy_ISR,              // vector_32 Vtrap0
(pFun)&dummy_ISR,              // vector_33 Vtrap1
(pFun)&dummy_ISR,              // vector_34 Vtrap2
(pFun)&dummy_ISR,              // vector_35 Vtrap3
(pFun)&dummy_ISR,              // vector_36 Vtrap4
(pFun)&dummy_ISR,              // vector_37 Vtrap5
(pFun)&dummy_ISR,              // vector_38 Vtrap6
(pFun)&dummy_ISR,              // vector_39 Vtrap7
(pFun)&dummy_ISR,              // vector_40 Vtrap8
(pFun)&dummy_ISR,              // vector_41 Vtrap9
(pFun)&dummy_ISR,              // vector_42 Vtrap10
(pFun)&dummy_ISR,              // vector_43 Vtrap11
(pFun)&dummy_ISR,              // vector_44 Vtrap12
(pFun)&dummy_ISR,              // vector_45 Vtrap13
(pFun)&dummy_ISR,              // vector_46 Vtrap14
(pFun)&dummy_ISR,              // vector_47 Vtrap15
(pFun)&dummy_ISR,              // vector_48 VReserved48
(pFun)&dummy_ISR,              // vector_49 VReserved49
(pFun)&dummy_ISR,              // vector_50 VReserved50
(pFun)&dummy_ISR,              // vector_51 VReserved51
(pFun)&dummy_ISR,              // vector_52 VReserved52
(pFun)&dummy_ISR,              // vector_53 VReserved53
(pFun)&dummy_ISR,              // vector_54 VReserved54
(pFun)&dummy_ISR,              // vector_55 VReserved55
(pFun)&dummy_ISR,              // vector_56 VReserved56
(pFun)&dummy_ISR,              // vector_57 VReserved57
(pFun)&dummy_ISR,              // vector_58 VReserved58
(pFun)&dummy_ISR,              // vector_59 VReserved59
(pFun)&dummy_ISR,              // vector_60 VReserved60
(pFun)&dummy_ISR,              // vector_61 Vunsinstr
(pFun)&dummy_ISR,              // vector_62 VReserved62
(pFun)&dummy_ISR,              // vector_63 VReserved63
(pFun)&dummy_ISR,              // vector_64 Virq
(pFun)&dummy_ISR,              // vector_65 Vlvd
(pFun)&dummy_ISR,              // vector_66 Vlol
(pFun)&dummy_ISR,              // vector_67 Vspi1
(pFun)&dummy_ISR,              // vector_68 Vspi2
(pFun)&usb_it_handler,         // vector_69 Vusb
(pFun)&dummy_ISR,              // vector_70 VReserved70
(pFun)&dummy_ISR,              // vector_71 Vtpm1ch0
(pFun)&dummy_ISR,              // vector_72 Vtpm1ch1
(pFun)&dummy_ISR,              // vector_73 Vtpm1ch2
(pFun)&dummy_ISR,              // vector_74 Vtpm1ch3
(pFun)&dummy_ISR,              // vector_75 Vtpm1ch4
(pFun)&dummy_ISR,              // vector_76 Vtpm1ch5
(pFun)&dummy_ISR,              // vector_77 Vtpm1ovf
(pFun)&dummy_ISR,              // vector_78 Vtpm2ch0
```

```
    (pFun)&dummy_ISR,              // vector_79 Vtpm2ch1
    (pFun)&dummy_ISR,              // vector_80 Vtpm2ovf
    (pFun)&dummy_ISR,              // vector_81 Vsci1err
    (pFun)&dummy_ISR,              // vector_82 Vsci1rx
    (pFun)&dummy_ISR,              // vector_83 Vsci1tx
    (pFun)&dummy_ISR,              // vector_84 Vsci2err
    (pFun)&dummy_ISR,              // vector_85 Vsci2rx
    (pFun)&dummy_ISR,              // vector_86 Vsci2tx
    (pFun)&dummy_ISR,              // vector_87 Vkeyboard
    (pFun)&dummy_ISR,              // vector_88 Vadc
    (pFun)&dummy_ISR,              // vector_89 Vacmpx
    (pFun)&dummy_ISR,              // vector_90 Viic1x
    (pFun)&dummy_ISR,              // vector_91 Vrtc
    (pFun)&dummy_ISR,              // vector_92 Viic2x
    (pFun)&dummy_ISR,              // vector_93 Vcmt
    (pFun)&dummy_ISR,              // vector_94 Vcanwu
    (pFun)&dummy_ISR,              // vector_95 Vcanerr
    (pFun)&dummy_ISR,              // vector_96 Vcanrx
    (pFun)&dummy_ISR,              // vector_97 Vcantx
    (pFun)&dummy_ISR,              // vector_98 Vrnga
    (pFun)&dummy_ISR,              // vector_99 VReserved99
    (pFun)&dummy_ISR,              // vector_100 VReserved100
    (pFun)&dummy_ISR,              // vector_101 VReserved101
    (pFun)&dummy_ISR,              // vector_102 VReserved102
    (pFun)&dummy_ISR,              // vector_103 VReserved103
    (pFun)&dummy_ISR,              // vector_104 VL7swi
    (pFun)&dummy_ISR,              // vector_105 VL6swi
    (pFun)&dummy_ISR,              // vector_106 VL5swi
    (pFun)&dummy_ISR,              // vector_107 VL4swi
    (pFun)&dummy_ISR,              // vector_108 VL3swi
    (pFun)&dummy_ISR,              // vector_109 VL2swi
    (pFun)&dummy_ISR,              // vector_110 VL1swi
};

#ifdef ON_THE_GO
/* If on-the-go is used pull-up control is done by the on-the-go driver.
   To avoid having trouble this callback must be empty. */
void enable_usb_pull_up(void)
{
}
#else
void enable_usb_pull_up(void)
{
 USB_OTG_CONTROL |= USB_OTG_CONTROL_DPPULLUP_NONOTG_MASK;
 // enable pullup in non-otg mode by setting bit 4
 USBTRC0_USBPU = 1;
}
#endif

int main()
{
  hcc_u8 tmp;

  /* !! This section needs to be here to redirect  interrupt vectors !! */
  dword *pdst,*psrc;
  byte i;
```

```
  asm (move.l  #0x00800000,d0);
  asm (movec   d0,vbr);

  pdst=(dword*)0x00800000;
  psrc=(dword*)&RAM_vector;

  for (i=0;i<111;i++,pdst++,psrc++)
  {
    *pdst=*psrc;
  }

  /* !! Start application code below here !! */

  stack_init(0x88);

  hw_init();
  usb_init();  /* select MCGPLLSCK clock as the source clock

  /* See in what mode should we run. */
  tmp = (hcc_u8)(SW1_ACTIVE() ? 1 : 0);
  tmp |= (hcc_u8)(SW2_ACTIVE() ? 2 : 0);

  /* Start the right HID application. */
  switch(tmp)
  {
  case 0:
  default:
    set_mode(dm_mouse);
    hid_mouse();
    break;
  case 1:
    set_mode(dm_kbd);
    hid_kbd();
    break;
  case 2:
    set_mode(dm_generic);
    hid_generic();
    break;
  }
  /* We will never get here. */
  return 0;
}
/***************************** END OF FILE *********************************/
```

Appendix G PRM File for the MC9S08 CMX Example

Section 4.6, “Adding the Bootloader to the Existing MC9S08 Project,” on page 36 gives detailed steps to start with the MC9S08 CMX Example and add the bootloader. The PRM file for this project required some modifications. Below is the text used for the new PRM file for this example with the bootloader. This file is located at the following path under the CMX Installation root directory:

\usb-peripheral\projects\CodeWarrior\hc9S08jm60\hid-demo\prm\Project.prm

```
/*****************************************************************************
*
*  (c) copyright Freescale Semiconductor 2008
*  ALL RIGHTS RESERVED
*
*  File Name: Project.prm
*
*  Purpose: This file is for a USB Mass-Storage Device bootloader.  This file
*           is the Linker Command File for the bootloader, and defines the
*           memory locations for both bootloader and application code
*
*  Assembler:  Codewarrior for Microcontrollers V6.2
*
*  Version:  1.2
*
*
*  Author: Derek Snell
*
*  Location: Indianapolis, IN. USA
*
* UPDATED HISTORY:
*
* REV   YYYY.MM.DD  AUTHOR        DESCRIPTION OF CHANGE
* ---   ----------  ------        ---------------------
* 1.2   2009.01.12  Derek Snell   Ported to S08 from V1
* 1.1   2008.10.09  Derek Snell   Overlapped RAM of Bootloader and Application
* 1.0   2008.06.10  Derek Snell   Initial version
*
*
*****************************************************************************/

/* Freescale  is  not  obligated  to  provide  any  support, upgrades or new */
/* releases  of  the Software. Freescale may make changes to the Software at */
/* any time, without any obligation to notify or provide updated versions of */
/* the  Software  to you. Freescale expressly disclaims any warranty for the */
/* Software.  The  Software is provided as is, without warranty of any kind, */
/* either  express  or  implied,  including, without limitation, the implied */
/* warranties  of  merchantability,  fitness  for  a  particular purpose, or */
/* non-infringement.  You  assume  the entire risk arising out of the use or */
/* performance of the Software, or any systems you design using the software */
/* (if  any).  Nothing  may  be construed as a warranty or representation by */
/* Freescale  that  the  Software  or  any derivative work developed with or */
/* incorporating  the  Software  will  be  free  from  infringement  of  the */
/* intellectual property rights of third parties. In no event will Freescale */
/* be  liable,  whether in contract, tort, or otherwise, for any incidental, */
/* special,  indirect, consequential or punitive damages, including, but not */
/* limited  to,  damages  for  any loss of use, loss of time, inconvenience, */
/* commercial loss, or lost profits, savings, or revenues to the full extent */
```

```
/* such  may be disclaimed by law. The Software is not fault tolerant and is */
/* not  designed,  manufactured  or  intended by Freescale for incorporation */
/* into  products intended for use or resale in on-line control equipment in */
/* hazardous, dangerous to life or potentially life-threatening environments */
/* requiring  fail-safe  performance,  such  as  in the operation of nuclear */
/* facilities,  aircraft  navigation  or  communication systems, air traffic */
/* control,  direct  life  support machines or weapons systems, in which the */
/* failure  of  products  could  lead  directly to death, personal injury or */
/* severe  physical  or  environmental  damage  (High  Risk Activities). You */
/* specifically  represent and warrant that you will not use the Software or */
/* any  derivative  work of the Software for High Risk Activities.           */
/* Freescale  and the Freescale logos are registered trademarks of Freescale */
/* Semiconductor Inc.                                                        */
/*****************************************************************************/

/* This is a linker parameter file for the mc9s08jm60 */

NAMES END /* CodeWarrior will pass all the needed files to the linker by command line. But here
you may add your own files too. */

HEXFILE S08_Bootloader.abs.s19

SEGMENTS /* Here all RAM/ROM areas of the device are listed. Used in PLACEMENT below. */

    // Bootloader Segments
    Bootloader               =  READ_ONLY    0xEC00 TO 0xFFAD;
    Bootloader_RAM           =  READ_WRITE   0x0100 TO 0x105F;
    Bootloader_USB_RAM       =  READ_WRITE   0x1860 TO 0x195F;
 /* INTVECTS                =  READ_ONLY    0xFFC4 TO 0xFFFF; Reserved for Interrupt Vectors */

    // Application Segments
    ROM                      =  READ_ONLY    0x1960 TO 0xEBA5;
    Z_RAM                    =  READ_WRITE   0x00B0 TO 0x00FF;
    Application_RAM          =  READ_WRITE   0x0100 TO 0x101F;
    Application_USB_RAM      =  READ_WRITE   0x1860 TO 0x195F;
    ROM1                     =  READ_ONLY    0x10B0 TO 0x17FF;
    ROM2                     =  READ_ONLY    0xFFC0 TO 0xFFC3;

    MY_STACK                 =  READ_WRITE   0x1020 TO 0x10AF;
END

PLACEMENT /* Here all predefined and user segments are placed into the SEGMENTS defined above. */


    DEFAULT_RAM                          INTO  Application_RAM,
                                               Application_USB_RAM;

//    _PRESTART,                           /* startup code */
    .init,
    STARTUP,                             /* startup data structures */
    ROM_VAR,                             /* constant variables */
    STRINGS,                             /* string literals */
    VIRTUAL_TABLE_SEGMENT,               /* C++ virtual table segment */
    DEFAULT_ROM,
    COPY                                /* copy down information: how to initialize variables */
                                         INTO  ROM; /* ,ROM1,ROM2: To use "ROM1,ROM2" as well,
pass the option -OnB=b to the compiler */
```

```
    _DATA_ZEROPAGE, MY_ZEROPAGE          INTO  Z_RAM;

    SSTACK                               INTO  MY_STACK;

END

STACKTOP  0x10AF
```

Appendix H hid_main.c for the MC9S08 CMX Example

Section 4.6, “Adding the Bootloader to the Existing MC9S08 Project,” on page 36 gives detailed steps to start with the MC9S08 CMX Example and add the bootloader. The hid_main.c file for this project required several modifications. Below is the text used for the new hid_main.c file for this example with the bootloader. This file is located at the following path under the CMX Installation root directory:

\usb-peripheral\src\hc9s08\hid-demo\hid_main.c

```
/*****************************************************************************
 *
 *            Copyright (c) 2006-2007 by CMX Systems, Inc.
 *
 * This software is copyrighted by and is the sole property of
 * CMX.  All rights, title, ownership, or other interests
 * in the software remain the property of CMX.  This
 * software may only be used in accordance with the corresponding
 * license agreement.  Any unauthorized use, duplication, transmission,
 * distribution, or disclosure of this software is expressly forbidden.
 *
 * This Copyright notice may not be removed or modified without prior
 * written consent of CMX.
 *
 * CMX reserves the right to modify this software without notice.
 *
 * CMX Systems, Inc.
 * 12276 San Jose Blvd. #511
 * Jacksonville, FL 32223
 * USA
 *
 * Tel:  (904) 880-1840
 * Fax:  (904) 880-1632
 * http: www.cmx.com
 * email: cmx@cmx.com
 *
 ****************************************************************************/
#include "usb-drv/usb.h"
#include "target.h"
#include "hid_mouse.h"
#include "hid_kbd.h"
#include "hid_generic.h"
/*****************************************************************************
 ************************* Macro definitions *********************************
 ****************************************************************************/
/* none */

/*****************************************************************************
 ************************* Function predefinitions. **************************
 ****************************************************************************/
/* none */

/*****************************************************************************
 ************************* Global variables **********************************
 ****************************************************************************/
/* none */

/*****************************************************************************
```

```
 ************************* Module variables ******************************
 *************************************************************************/
/* none */

/*************************************************************************
 ************************* Function definitions **************************
 *************************************************************************/

  // Added for Bootloader
  #include "Bootloader_S08.h"

  void _Startup(void);
  // User Application code entry
  volatile const JumpVect UsrEntry@ USER_ENTRY_ADDRESS = {
    0xCC,     // op-code for JMP
    _Startup
  };

  interrupt void Dummy_ISR(void) {}

  // User Interrupt Jump Vector Table
 volatile const JumpVect UserJumpVectors [InterruptVectorsNum] @ VectorAddressTableAddress = {
      { 0xCC, Dummy_ISR},     // 29 - RTC
      { 0xCC, Dummy_ISR},     // 28 - IIC
      { 0xCC, Dummy_ISR},     // 27 - ACMP
      { 0xCC, Dummy_ISR},     // 26 - ADC Conversion
      { 0xCC, Dummy_ISR},     // 25 - KBI
      { 0xCC, Dummy_ISR},     // 24 - SCI2 Transmit
      { 0xCC, Dummy_ISR},     // 23 - SCI2 Receive
      { 0xCC, Dummy_ISR},     // 22 - SCI2 Error
      { 0xCC, Dummy_ISR},     // 21 - SCI1 Transmit
      { 0xCC, Dummy_ISR},     // 20 - SCI1 Receive
      { 0xCC, Dummy_ISR},     // 19 - SCI1 Error
      { 0xCC, Dummy_ISR},     // 18 - TPM2 Overflow
      { 0xCC, Dummy_ISR},     // 17 - TPM2 Channel1
      { 0xCC, Dummy_ISR},     // 16 - TPM2 Channel0
      { 0xCC, Dummy_ISR},     // 15 - TPM1 Overflow
      { 0xCC, Dummy_ISR},     // 14 - TPM1 Channel5
      { 0xCC, Dummy_ISR},     // 13 - TPM1 Channel4
      { 0xCC, Dummy_ISR},     // 12 - TPM1 Channel3
      { 0xCC, Dummy_ISR},     // 11 - TPM1 Channel2
      { 0xCC, Dummy_ISR},     // 10 - TPM1 Channel1
      { 0xCC, Dummy_ISR},     //  9 - TPM1 Channel0
      { 0xCC, Dummy_ISR},     //  8 - Reserved
      { 0xCC, usb_it_handler}, //  7 - USB Status
      { 0xCC, Dummy_ISR},     //  6 - SPI2
      { 0xCC, Dummy_ISR},     //  5 - SPI1
      { 0xCC, Dummy_ISR},     //  4 - MCG Loss Lock
      { 0xCC, Dummy_ISR},     //  3 - Low VoltDetect
      { 0xCC, Dummy_ISR},     //  2 - IRQ
      { 0xCC, Dummy_ISR},     //  1 - SWI
  };


int main()
{
  hcc_u8 tmp;
```

```
  hw_init();

  /* See in what mode should we run. */
  tmp = (hcc_u8)(SW1_ACTIVE() ? 1 : 0);
  tmp |= (hcc_u8)(SW2_ACTIVE() ? 2 : 0);

  /* Start the right HID application. */
  switch(tmp)
  {
  default:
  case 0:
    usb_cfg_init();
    set_mode(dm_mouse);
    (void)hid_mouse();
    break;
  case 1:
    usb_cfg_init();
    set_mode(dm_kbd);
    hid_kbd();
    break;
  case 2:
    usb_cfg_init();
    set_mode(dm_generic);
    hid_generic();
    break;
  }
  /* We will never get here. */
  return 0;
}
/****************************** END OF FILE **********************************/
```

How to Reach Us:

Home Page:
www.freescale.com

Web Support:
http://www.freescale.com/support

USA/Europe or Locations Not Listed:
Freescale Semiconductor, Inc.
Technical Information Center, EL516
2100 East Elliot Road
Tempe, Arizona 85284
+1-800-521-6274 or +1-480-768-2130
www.freescale.com/support

Europe, Middle East, and Africa:
Freescale Halbleiter Deutschland GmbH
Technical Information Center
Schatzbogen 7
81829 Muenchen, Germany
+44 1296 380 456 (English)
+46 8 52200080 (English)
+49 89 92103 559 (German)
+33 1 69 35 48 48 (French)
www.freescale.com/support

Japan:
Freescale Semiconductor Japan Ltd.
Headquarters
ARCO Tower 15F
1-8-1, Shimo-Meguro, Meguro-ku,
Tokyo 153-0064
Japan
0120 191014 or +81 3 5437 9125
support.japan@freescale.com

Asia/Pacific:
Freescale Semiconductor China Ltd.
Exchange Building 23F
No. 118 Jianguo Road
Chaoyang District
Beijing 100022
China
+86 10 5879 8000
support.asia@freescale.com

For Literature Requests Only:
Freescale Semiconductor Literature Distribution Center
1-800-441-2447 or 303-675-2140
Fax: 303-675-2150
LDCForFreescaleSemiconductor@hibbertgroup.com

Document Number: AN3927
Rev. 1
02/2012

freescale™
semiconductor