Document Number: AN4379
Rev. 0, October 2011

Freescale USB 大容量存储设备引导加载程序

Derek Snell
Freescale

目录

1 简介

Freescale 很多微控制器都集成了通用串行总线（USB）访问接口。带USB端口的产品可以极为方便地在现场更新固件。本应用笔记介绍一个大容量存储设备 (MSD) USB 引导加载程序，它能配合多个 Freescale USB 系列进行工作。具有该引导加载程序的设备连接到主机后，该引导加载程序就会被枚举为一个新的驱动器。在新固件复制到该驱动器上后，设备会使用新的固件对自己进行更新。

Freescale 还提供其他引导加载程序。例如，应用笔记 AN3561 （“适用于 MC9S08JM60 的 USB 引导加载程序”）介绍了为 Flexis JM 系列而编写的 USB 引导加载程序。本应用笔记中介绍的 MSD 引导加载程序作为另一种选择而提供，且具有以下这些优势：

- 无需在主机上安装驱动程序。
- 无需在主机上运行应用程序。
- 任何用户只需稍加培训都能使用。唯一需要的操作是将文件复制到驱动器上。
- 由于不需要主机软件或驱动程序，因此它能用于许多不同的主机操作系统。

该引导加载程序专门针对多个具有相似 USB 外设的 Freescale 微控制器系列而编写。这些系列包括但不限于：

- Flexis JM 系列 MCF51JM
- 集成 USB 的 ColdFire MCF522xx
- Kinetis

该引导加载程序只需略经修改便可支持上述所有器件；实例针对下列Freescale微控制器编写并经过测试：

- MCF52259 — 带 USB 、以太网、 CAN 和外部总线的 32 位 ColdFire V2
- MCF51JM128 — 带 USB 的 32 位 ColdFire V1 （属于 Flexis JM 系列）
- MK60N512 — 带USB 2.0全速OTG控制器和10/100 Mbps以太网MAC的ARM® Cortex™-M4架构

使用的 USB 协议栈：

- 支持 PHDC 的 Freescale USB 协议栈 v3.0

测试的开发板：

- TWR-MCF5225X-KIT — 适用于 MCF5225x 系列的低成本 Tower 套件
- DEMOJM — 适用于 Flexis JM 系列的低成本开发板
- TWR-K60N512-KIT — 适用于 Kinetis K60 系列的低成本 Tower 套件

测试的操作系统：

- Windows XP 专业版（ Service Pack 2 和 Service Pack 3 ）

测试的开发工具：

- CodeWarrior for Microcontrollers v10.1
- CodeWarrior for Microcontrollers v6.3
- CodeWarrior for ColdFire v7.2

2 功能说明

2.1 概述

本节概要说明 USB MSD 设备引导加载程序的架构及其软件流程。

2.1.1 USB MSD 设备引导加载程序架构

Freescale USB MSD 设备引导加载程序的架构如图 1 所示。

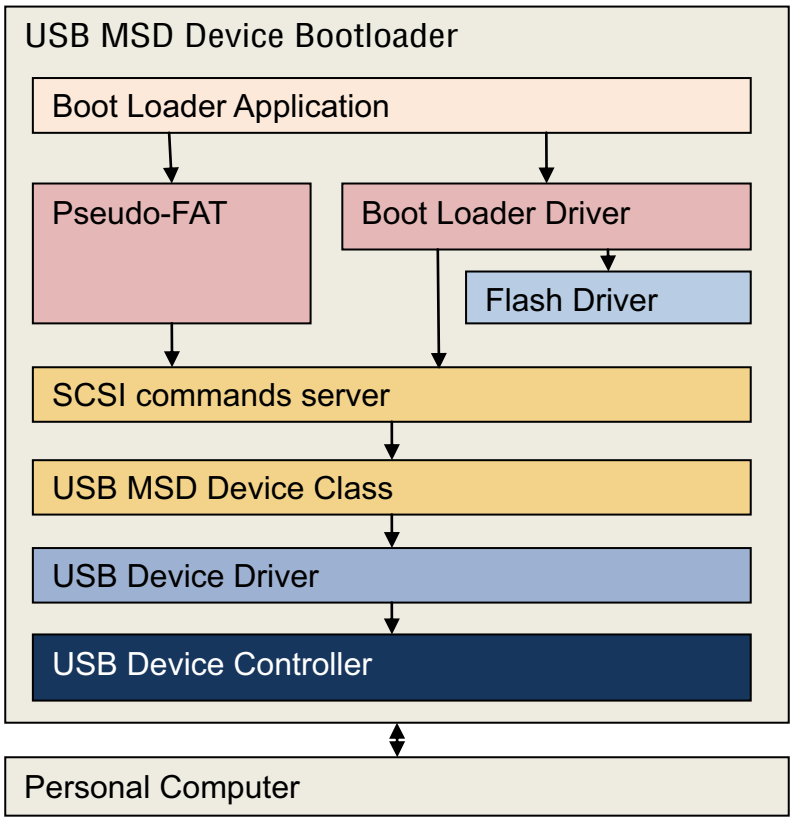

图 1. USB MSD 引导加载程序架构

USB MSD 设备引导加载程序包括：

- 引导加载程序的应用程序：控制整个加载过程并执行一般任务
- 伪 FAT 结构：包含用以响应主机 SCSI 命令的结构。
- 引导加载程序的驱动程序：利用 SCSI 命令服务器从主机接收数据，然后解析映像文件并将其下载到 Flash 存储器。引导加载程序驱动程序支持解析 CodeWarrior 二进制、 S-Record 和原始二进制文件格式的映像文件。
- Flash 驱动程序：支持对 Flash 存储器进行的擦除、读取和写入功能。
- SCSI 命令服务器：响应主机 SCSI 命令。
- USB MSD 设备类：提供 MSD 类中指定的 API 。
- USB 设备驱动程序和设备控制器：通过 USB 协议与 USB 主机通信。

2.1.2 引导加载程序流程

引导加载程序与执行产品主要功能的应用程序相集成。复位时，引导加载程序执行某些简单的检查以确定是执行应用程序还是进入引导加载程序模式。如果进入引导加载程序模式，它就会利用 USB 在主机中进行枚举。在枚举过程中，设备声明自身为 MSD 。主机随后在系统中创建一个新的驱动器。固件映像文件随后可被复制到该驱动器上，然后设备会使用新的固件对自己进行更新。引导加载程序支持采用 S-record 、 CodeWarrior 二进制和原始二进制文件格式的固件映像。

S-record 文件是常用的 ASCII 文件类型，用于指定程序数据在设备中存储的位置。编译工程时，Freescale 的软件工具链 CodeWarrior 自动生成 S-record 文件和 CodeWarrior 二进制文件。 S-record 文件的扩展名为 .S19， CodeWarrior 二进制文件的扩展名为 .bin 。原始二进制文件是 Flash 存储器的映像文件。

固件映像文件传输到设备且设备自身的重编程完成后，该设备在主机中重新枚举。驱动器中显示一个文件代表引导加载程序操作的状态。

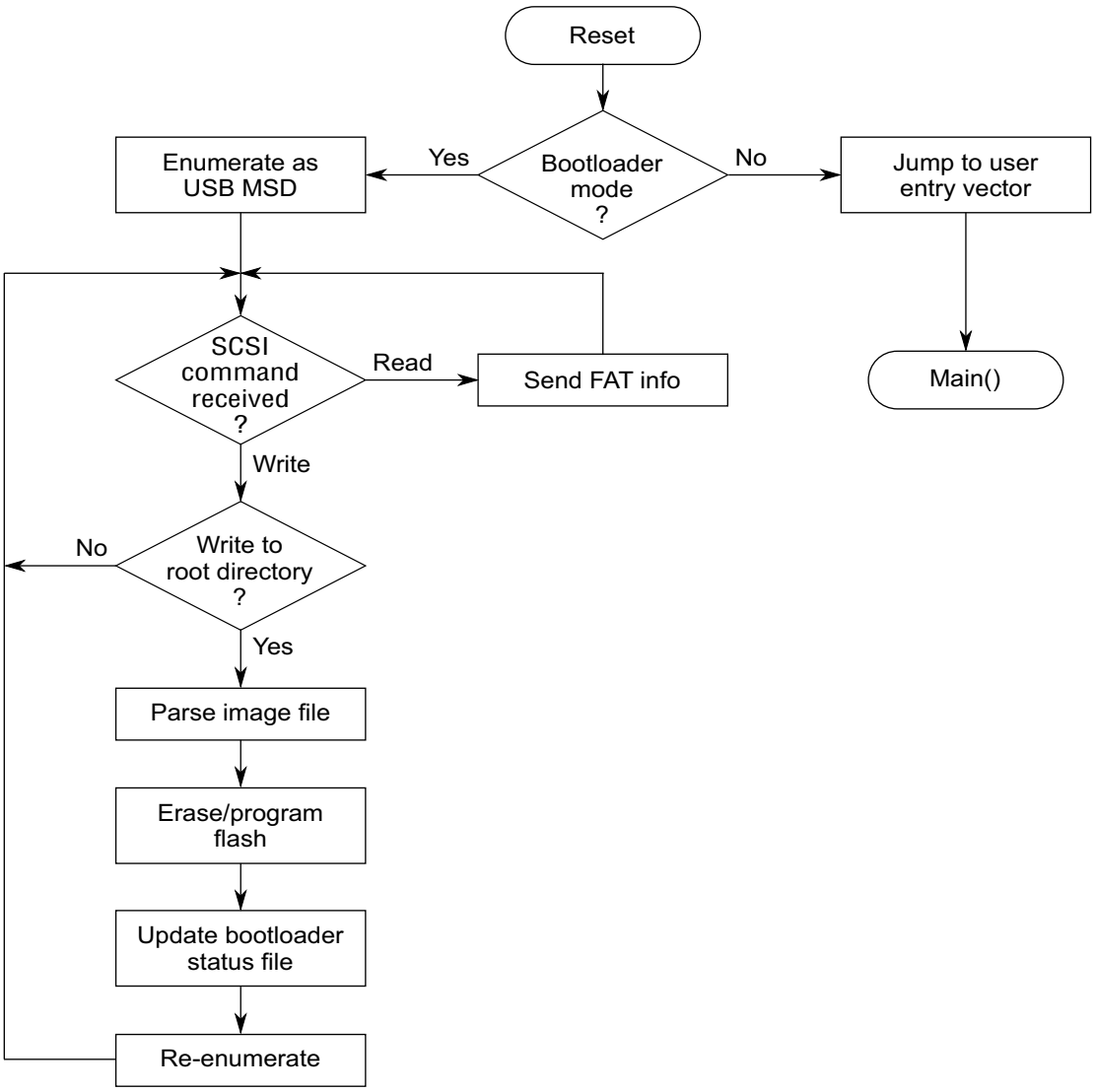

图 2. 引导加载程序功能流程图

2.2 引导加载程序模式与应用程序模式

引导加载程序在复位后立即就会开始执行并确定进入何种模式。复位后，引导加载程序必须处于主控状态，以便能在应用程序不存在、被擦除或损坏时运行。通常情况下，引导加载程序会尝试进入应用程序模式，这通过跳转到应用程序的复位向量而实现。

进入应用程序模式前，引导加载程序会执行某些简单的检查以确保应用程序存在。它查看应用程序的初始栈指针值和复位向量（固件映像的第一个四字节和第二个四字节），确保其未被擦除。如果它们有效，引导加载程序会跳转到应用程序 — 否则，就会进入引导加载程序模式以便更新应用程序。

复位期间也可以强制进入引导加载程序模式。文中的实例监控一个按钮，如果复位期间一直按住该按钮，无论应用程序的栈指针值和复位向量是否有效，都会进入引导加载程序模式。用于进入此模式的按钮取决于开发板；具体按钮参见在演示板上使用引导加载程序。许多应用需要更改用于强制进入引导加载程序模式的方法，这可以通过修改 Boot_loader_task.c 文件来实现。

2.3 USB 枚举

引导加载程序进入引导加载程序模式后， USB 外设就能与主机进行通信。这会启动 USB 枚举过程。（有关枚举的详细信息，请参阅文档 MEDUSBUG ： “带 PHDC 设备的 Freescale USB 协议栈用户指南”。）该引导加载程序使用的 USB 协议栈为带 PHDC 的 Freescale USB 协议栈 v3.0 。该设备向主机发送 USB 描述符，声明其为大容量存储设备。主机将该设备加载为新驱动器，并使用批量传输来与设备通信。这种批量传输发送 SCSI 命令以与移动存储设备（对主机而言）通信。

2.4 伪 FAT

主机将引导加载程序配置为存储驱动器后，就会尝试读取该驱动器的文件分配表 (FAT) ，并读取驱动器的内容。引导加载程序用 1 GB 驱动器之类的信息响应这些查询。该驱动器的卷名为 BOOTLOADER 。它看起来像是没有子目录，但根目录中有一个名为 READY.TXT 的文件。

引导加载程序并不使用真正的文件系统，并且只能对主机显示为驱动器。它将接受可写入其中的文件，这些文件随后会被传输到固件映像文件。然而，由于引导加载程序只是一个伪 FAT 系统，因此当主机试图对其作更改时，它不会更新 FAT 。 FAT 始终是固定的，唯一的变化是根目录中指定引导加载程序状态的文件名称。

当主机将数据写入该驱动器时，引导加载程序会忽略所有不是写入驱动器根目录的数据。将文件数据写入根目录时，引导加载程序会认为它是固件映像文件。它会擦除应用程序 Flash 并启动固件映像解析器以获得新的固件更新。复位之后，只能向该驱动器传输一个文件。传完一个文件后，引导加载程序需要重启才能接受另一个文件。

2.5 固件映像文件解析器

引导加载程序包含一个固件映像解析器，用于将固件映像文件原样送至引导加载程序。将某个文件传输到引导加载程序后，它假定该文件是固件映像文件，并开始解析该文件。它解析可存储一部分固件映像的某个缓冲区，如果成功，则继续解析下一部分的映像，直至整个映像解析完毕。

如果数据的地址有效，它就会将固件映像的所有数据写入 Flash 存储器。对于 S-Record 文件，写入数据前会验证每个 S-record 校验和。所有 Flash 操作都成功后，解析器会返回一条表示成功的消息。若在任何地方映像文件无效或 S-record 校验和未通过验证，则解析器返回错误。

附注

其他不使用USB的引导加载程序或应用程序也可以使用该解析器。如需其他类型的通信，程序只需将接收到的数据映像存储到某个缓冲区，然后将该缓冲区传送给解析器的函数（在文件 loader.c 中）以解析映像。

2.6 重新枚举

解析器完成解析后，设备在主机中重新枚举以更新驱动器的状态文件。状态文件在引导加载程序状态中讨论。在主机中，设备与USB总线断开连接后，驱动器就会随之消失。几秒钟后，设备重新枚举，驱动器再次出现。可以读取驱动器上的状态文件以查看引导加载程序的状态。

引导加载程序之所以要重新枚举，是因为有些主机操作系统将文件数据缓存在存储驱动器上且不刷新。其他一些操作系统则是执行延迟写入，直到驱动器卸载后才会将文件写入驱动器。卸载之后，设备接收固件映像文件，更新固件，并在随后重新枚举以显示状态。在 Windows XP 和 Vista 中，操作系统缓存驱动器的FAT而不对其重新读取，而且状态文件不会更新。重新枚举迫使主机重新读取 FAT 并显示引导加载程序状态文件。

2.7 引导加载程序状态

引导加载程序利用主机中的文件系统驱动器显示状态。这种方法使得引导加载程序的状态始终可见，无论设备硬件提供何种类型的显示器或 LED 。如果发生 S-record 错误，还会显示错误的地址。

状态以驱动器根目录中的文件名显示。这些文件是空文件，无数据。将文件传输到引导加载程序之前，用户必须检查状态文件 READY.TXT 。传输完毕且引导加载程序重新枚举后，用户必须检查状态文件以确定是否成功。状态文件名如下所列：

- READY.TXT — 表示引导加载程序已就绪，可接收映像文件。
- SUCCESS.TXT — 表示引导加载程序已成功完成固件更新。
- FFAILED.TXT — 表示引导加载程序发生Flash擦除或写入错误。如果Flash擦除或写入例程返回错误，就会触发该错误。触发该错误的原因通常是 Flash 时钟频率不合适或使用的地址无效。如果该地址不是Flash地址或受到保护，就会发生错误。若要诊断，请在Loader.c中生成该错误的地方设置一个断点，然后运行调试器找出出错的原因。
- SFxxxxxx.TXT — 表示引导加载程序S-record解析器在S-record文件中找到错误。解析器测试每个 S-record 以确保其格式有效且校验和相符。如果测试失败，就会生成错误。如果 S-record 指定的地址无效，也会发生该错误。 S-record 文件的错误地址显示在文件名中。例如，若 S-record 的地址 0x1234 无效，文件名将是 SF001234.TXT 。发生该错误时，请检查 S-record 的格式、校验和及地址是否正确。
- STARTED.TXT — 表示根目录已接收到含数据的文件，固件映像解析器已启动。通常情况下，此状态对用户应当是不可见的，因为驱动器只在成功或失败后才重新枚举。如果可见，请调试引导加载程序以诊断问题。

2.8 Flash 保护

引导加载程序必须始终受到保护，不会擦除自身或损坏。启用 Flash 保护后，固件更新不成功造成的最坏情况是应用程序被擦除或损坏，但引导加载程序仍旧完好无损。引导加载程序随后可以再次运行并重新加载新的应用程序。

该 MSD 引导加载程序使用 Freescale 的 Flash 保护特性。取决于所用的器件，微控制器有一个非易失性寄存器，可指定哪些 Flash 扇区受到保护。引导加载程序存储在受保护的扇区中，应用程序存储在未受保护的扇区中。这些非易失性寄存器设置是引导加载程序的一部分，应用程序无法更改。此外，该 Flash 保护还可保护复位和中断向量使其不被应用程序更改。

下面列出具有不同内核的Flash保护寄存器以及受保护的扇区。有关Flash存储器使用量的受影响情况，请参见表 4 。这些保护值可在 Bootloader.h 中修改。

- ColdFire V2 — 位于 0x408 的寄存器 CFMPROT 设置为 0x7 ， 保护 Flash 地址范围 0x0 至 0xBFFF 。
- ColdFire V1 — 位于 0x40D 的寄存器 NVPROT 设置为 0xD7 ， 保护 Flash 地址范围 0x0 至 0x9FFF 。
- Kinetis K60N512 — 位于 0x408 的寄存器 FPROT 设置为 0xFFFFFFF8 ， 保护 Flash 地址范围 0x0 至 0xBFFF 。

附注

如果修改引导加载程序，可以增加或减少这些 Flash 保护扇区，以便妥善保护引导加载程序。如果地址范围发生变化，则更改上述寄存器，并且还要更改链接器文件和 Bootloader.h 中的内存映射。

2.9 重定向中断向量

应用程序因 Flash 存储器受保护而无法更改微控制器的默认中断向量表。因此，应用程序需要将其中断向量表存储在应用程序Flash存储器中并在随后将这些向量移至RAM。通过这种方式的重定向，应用程序就能更新这些向量。有关各器件的更具体细节，请参见重定向中断向量。

2.10 栈和 RAM 使用

引导加载程序仅在启动应用程序时与其进行交互。应用程序绝不会调用引导加载程序中的函数。因此，引导加载程序和应用程序无需具有单独的 RAM 存储器，都可使用相同的物理 RAM 。就应用程序而言，引导加载程序本质上不使用 RAM ，因为应用程序能够访问设备的整个 RAM 。链接器文件设置引导加载程序与应用程序共享整个物理 RAM 存储器。

2.11 引导加载程序内存映射

下面几小节给出三个不同设备示例的内存映射。给出引导加载程序和应用程序的内存映射是为了使读者了解 RAM 的重叠情况。注意，应用程序能够访问所有 RAM 且与引导加载程序 RAM 重叠。二者的所有其他内存部分完全相同。

2.11.1 ColdFire V2 引导加载程序内存映射

表 1. MCF52259 引导加载程序内存映射

地址	引导加载程序	应用程序
0x0000_0000 至 0x0000_03FF	引导加载程序中断向量	引导加载程序中断向量
0x0000_0400 至 0x0000_041F	Flash 保护和安全寄存器	Flash 保护和安全寄存器
0x0000_0420 至 0x0000_BFFF	引导加载程序 Flash (48 KB)	引导加载程序 Flash (48 KB)
0x0000_C000 至 0x0000_C3FF	应用程序存储的中断向量	应用程序存储的中断向量
0x0000_C400 至 0x0000_C41F	应用程序 Flash 保护和安全（未使用）	应用程序 Flash 保护和安全（未使用）
0x0000_C500 至 0x0007_FFFF	应用程序 Flash 存储器 (463 KB)	应用程序 Flash 存储器 (463 KB)
0x0008_0000 至 0x1FFF_FFFF	保留	保留
0x2000_0000 至 0x2000_03FF	RAM 中的重定向中断向量表	RAM 中的重定向中断向量表
0x2000_0400 至 0x2000_FFFF	引导加载程序可用的 RAM	应用程序可用的 RAM

2.11.2 ColdFire V1 引导加载程序内存映射

表 2. MCF51JM128 引导加载程序内存映射

地址	引导加载程序	应用程序
0x(00)00_0000 至 0x(00)00_03FF	引导加载程序中断向量	引导加载程序中断向量
0x(00)00_0400 至 0x(00)00_040F	Flash 保护和安全寄存器	Flash 保护和安全寄存器
0x(00)00_0410 至 0x(00)00_9FFF	引导加载程序 Flash (39 KB)	引导加载程序 Flash (39 KB)
0x(00)00_A000 至 0x(00)00_A3FF	应用程序存储的中断向量表	应用程序存储的中断向量表
0x(00)00_A400 至 0x(00)00_A40F	应用程序 Flash 保护和安全（未使用）	应用程序 Flash 保护和安全（未使用）
0x(00)00_A410 至 0x(00)01_FFFF	应用程序 Flash 存储器 (87 KB)	应用程序 Flash 存储器 (87 KB)
0x(00)20_0000 至 0x(00)7F_FFFF	保留	保留
0x(00)80_0000 至 0x(00)80_01BB	RAM 中的重定向中断向量表	RAM 中的重定向中断向量表
0x(00)80_01BC 至 0x(00)80_3FFF	引导加载程序可用的 RAM	应用程序可用的 RAM

2.11.3 Kinetis 引导加载程序内存映射

表 3. K60N512 引导加载程序内存映射

地址	引导加载程序	应用程序
0x0000_0000 至 0x0000_03FF	引导加载程序中断向量	引导加载程序中断向量
0x0000_0400 至 0x0000_040F	Flash 保护和安全寄存器	Flash 保护和安全寄存器
0x0000_0410 至 0x0000_BFFF	引导加载程序 Flash (47 KB)	引导加载程序 Flash (47 KB)
0x0000_C000 至 0x0000_C3FF	应用程序存储的中断向量表	应用程序存储的中断向量表
0x0000_C400 至 0x0000_C410	应用程序 Flash 保护和安全（未使用）	应用程序 Flash 保护和安全（未使用）

下一页继续介绍此表...

表 3. K60N512 引导加载程序内存映射（续）

地址	引导加载程序	应用程序
0x0000_C410 至 0x0007_FFFF	应用程序 Flash 存储器 (463 KB)	应用程序 Flash 存储器 (463 KB)
0x0008_0000 至 0x1FFE_FFFF	保留	保留
0x1FFF_0000 至 0x1FFF_03FF	RAM 中的重定向中断向量表	RAM 中的重定向中断向量表
0x1FFF_0400 至 0x2000_FFFF	引导加载程序可用的 RAM	应用程序可用的 RAM

2.12 资源使用情况

表 4 显示引导加载程序在三个不同设备示例中的资源使用情况。如 Flash 保护所述，引导加载程序在 Flash 存储器中受到保护。选择 Flash 最小保护值来保护引导加载程序。因此，引导加载程序使用的 Flash 内存量由设备提供的最小保护值决定。

表 4. 引导加载程序资源使用情况

器件	Flash 使用量 (KB)	加密的 Flash (KB)	RAM 使用量（字节）
MCF52259	33	48	0
MCF51JM128	32	40	0
K60N512	33	48	0

有关引导加载程序为何不使用 RAM 的说明，请参见栈和 RAM 使用。

3 使用引导加载程序

3.1 引导加载程序文件结构

本节介绍针对引导加载程序和应用程序实例提供的固件源代码。本应用笔记提供的源代码包括引导加载程序和所有三个内核的实例。图 3 显示了所提供源代码的目录结构。

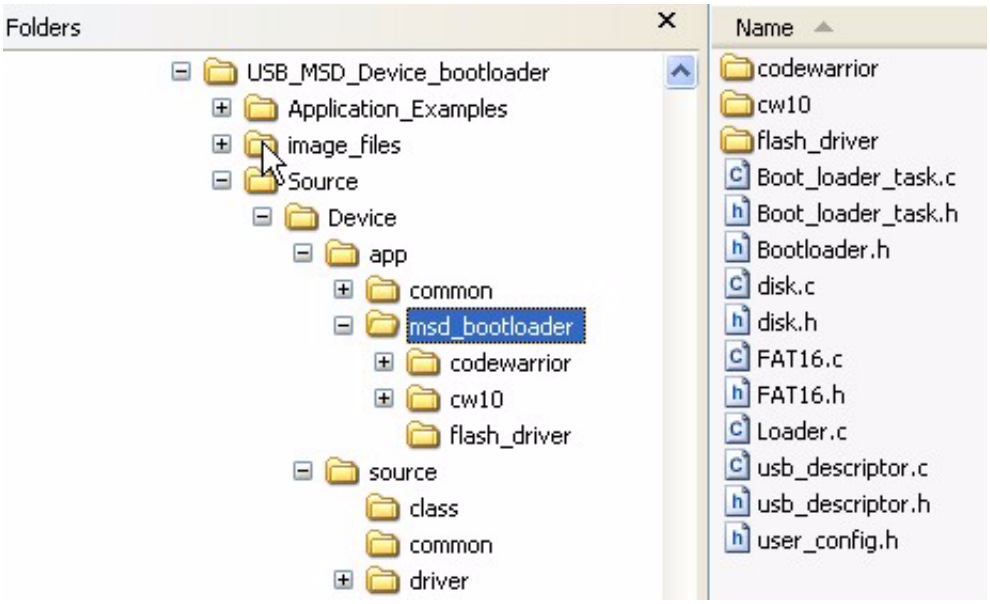

图 3. 引导加载程序文件结构

- Application_Examples ：包含应用程序实例项目，用以使应用程序与引导加载程序集成。更多详情，请参见应用示例。
- image_files ：包含应用程序的一些固件映像文件实例，用以结合引导加载程序进行测试。更多详情，请参见在演示板上使用引导加载程序。
- CodeWarrior：包含配合ColdFire V1使用的CodeWarrior for Microcontrollers v6.3项目和配合ColdFire V2 使用的 CodeWarrior for ColdFire v7.2 项目。
- cw10 ：包含用于 ColdFire 和 Kinetis 的 CodeWarrior v10.1 项目。
- Flash_driver ：包含适用于许多微控制器的 Flash 驱动程序源代码。
- Boot_loader_task.c ：引导加载程序通用任务的源文件。
- Boot_loader_task.h ：函数原型。
- Bootloader.h ：包含特定平台板的存储器映射定义。
- disk.c ：包含大容量存储磁盘函数。
- disk.h ：包含磁盘参数定义。
- FAT16.c ：包含伪 FAT 结构。
- FAT16.h ：包含 FAT 参数定义。
- Loader.c ：包含用以解析和加载固件映像到 Flash 存储器的函数。
- usb_descriptor.c ：包含 USB 描述符结构和函数。

- usb_descriptor.h ： 包含 USB 描述符参数。
- user_config.h ： 包含 USB 协议栈的用户配置。

3.2 在演示板上使用引导加载程序

本节介绍如何使用引导加载程序，包括如何在开发板上演示引导加载程序，以及如何将引导加载程序移植到其他平台。

所提供的 CodeWarrior 项目包含配合 TWR-MCF5225X-KIT 、 DEMOJM 和 TWR-K60N512-KIT 进行测试的应用程序实例和映像文件。选择一个引导加载程序项目以在对应的板上进行测试。有关入门指南及如何利用 CodeWarrior 对这些板进行编程，请参阅这些板随附的文档。熟悉这些板和 CodeWarrior 之后，可使用下面的步骤：

1. 硬件必须针对USB设备操作而设置。更多详情，请参阅演示板文档。如果使用的是DEMOJM板，则应确保将正确的 MCF51JM128 模块插入板中。用 USB 线连接演示版和计算机。
2. 打开对应的 CodeWarrior 项目，然后打开其中一个应用程序实例对应的引导加载程序项目文件。编译项目并对 Flash 进行编程。
3. 复位演示板。设备在引导加载程序模式下重启并在主机中枚举一个新的驱动器。
4. 图 4 显示了 Windows XP 中的引导加载程序驱动器。注意，该驱动器的卷名为 BOOTLOADER，状态文件为 READY.TXT 。引导加载程序现已准备好接收固件映像文件。

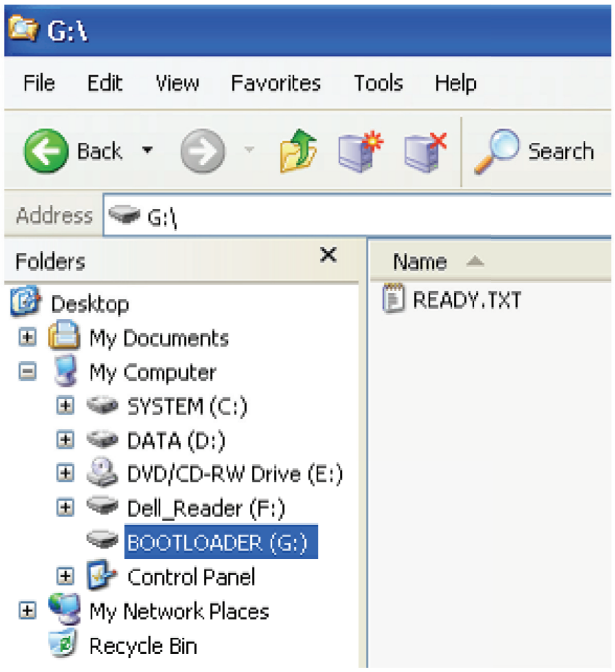

图 4. 枚举后的引导加载程序驱动器

5. 转到“ image_files ”目录，复制该板的一个映像文件，将其粘贴到引导加载程序驱动器。拖放文件同样有效。

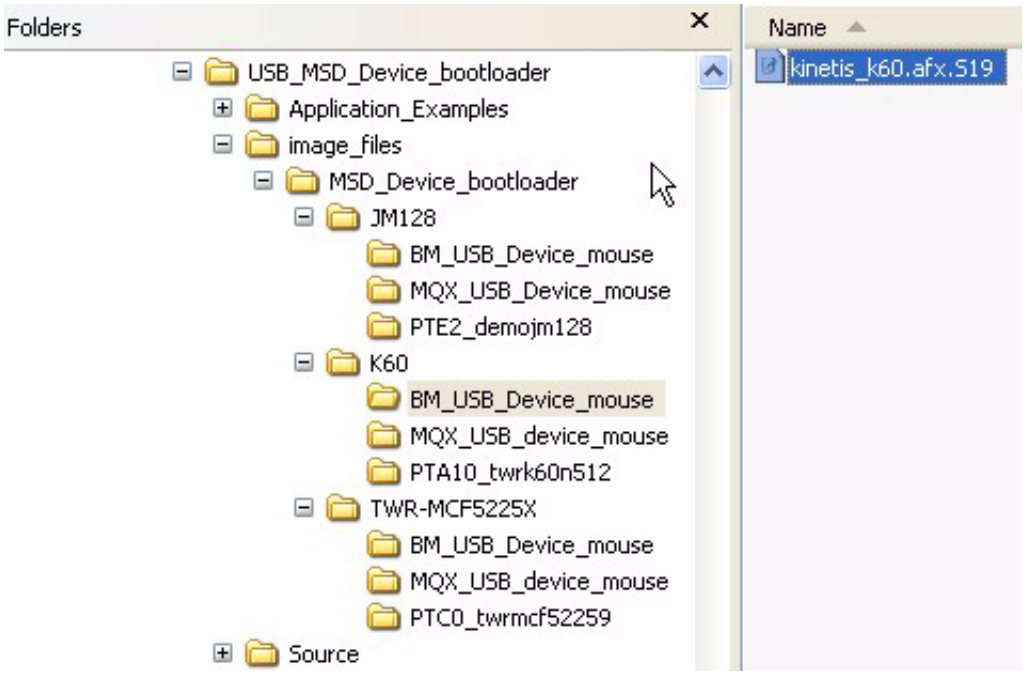

图 5. 选择要下载的固件映像文件

某些操作系统（如 Linux Fedora ）要等到驱动器卸载后才会将文件传输到驱动器。如果使用的是此类操作系统，则应卸载引导加载程序驱动器。

6. 检查引导加载程序文件状态。文件传输完毕后，引导加载程序驱动器会消失。几秒钟后，驱动器重新枚举并再次出现。单击引导加载程序驱动器，验证状态文件是否为 SUCCESS.TXT 。经过以上操作，固件更新即告完成。

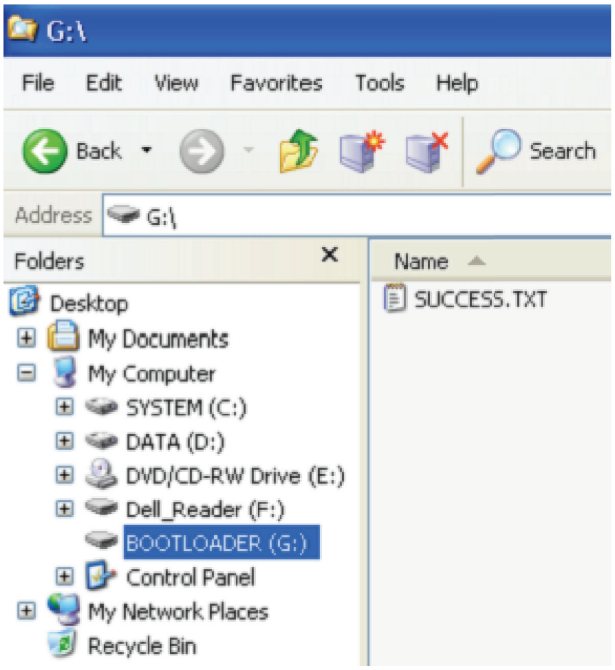

图 6. 状态文件显示成功的引导加载程序驱动器

7. 使用复位按钮复位演示板。随即执行新的应用程序。

加载某个应用程序后，在复位期间按住一个按钮便可强制进入引导加载程序模式。下面是要使用的按钮，具体取决于演示板：

- TWR-MCF5225X-KIT — SW1
- DEMOJM — PTG1
- TWR-K60N512-KIT — SW1

3.3 防止引导加载程序与应用程序链接

必须明白，良好的引导加载程序只应通过绝对地址与应用程序交互。这种情况下，应用程序复位向量位于某个绝对地址，该向量指向应用程序的入口点。引导加载程序知道该绝对地址，为了进入应用程序模式，它将利用应用程序复位向量跳转到应用程序的开始位置。这很重要，因为每次重新编译应用程序时，链接器可能会更改入口点的地址。引导加载程序无需知道改变后的地址，因为应用程序复位向量始终位于同一位置，并且总是指向应用程序入口点。

引导加载程序和应用程序使用单个项目的好处是生产中只需对设备编程一次，调试也更容易。然而，如果引导加载程序和应用程序链接在一起，那么使用单个项目也具有一些潜在风险。必须防止链接器让某一部分引用另一部分中的非绝对地址。

考虑这种情况：引导加载程序使用了一个 ANSI 库，应用程序也使用该库，二者在同一个项目中进行编译。固件的 RevA 已完成编译，被称为 MyFunc() 的一个库函数链接到引导加载程序位于地址 0x1000 的部分。应用程序也使用 MyFunc()，当执行该函数时便跳转到引导加载程序中。RevA 交付生产，一切正常。经过一段时间后，固件更新到 RevB 。重新编译该项目时，链接器将 MyFunc() 链接到地址 0x1005 。 RevB 的引导加载程序和应用程序链接在一起，因此该版本同样

能正常工作。现在， RevB 固件更新下载到一台含 RevA 引导加载程序的设备。应用程序更新到 RevB ，但引导加载程序仍旧是 RevA 。当应用程序执行并跳转到 MyFunc() 时，它跳转到地址 0x1005 。但是， RevA 引导加载程序中的 MyFunc() 位于地址 0x1000 ，因此代码将跑飞。

上例说明了应用程序和引导加载程序之间应当避免存在链接（除非通过绝对地址链接）的原因。该引导加载程序仅在其启动应用程序时与应用程序进行交互，这是利用应用程序复位向量的绝对地址完成的。如果应用程序项目中包括引导加载程序，那么链接器应当只链接到引导加载程序的二进制映像，而不能链接到源代码或库。采用这种方法，当编译应用程序时，链接器对引导加载程序中的函数将不知情。

4 将 USB MSD 设备引导加载程序移植到其他平台

本节高度概括了如何将 USB MSD 设备引导加载程序移植到其他 Freescale 设备。对要移植的设备做如下的假定：

- 该设备由一个带 MSD 类的 USB 协议栈提供支持。
- 该设备有一个简单的 Flash 驱动程序。

执行以下步骤将引导加载程序移植到其他平台：

1. 在 `\USB_MSD_Device_bootloader\Source\Device\app\msd_bootloader \codewarrior\ or \USB_MSD_Device_bootloader\Source\Device\app\msd_bootloader\cw10\` 目录下新建一个项目。

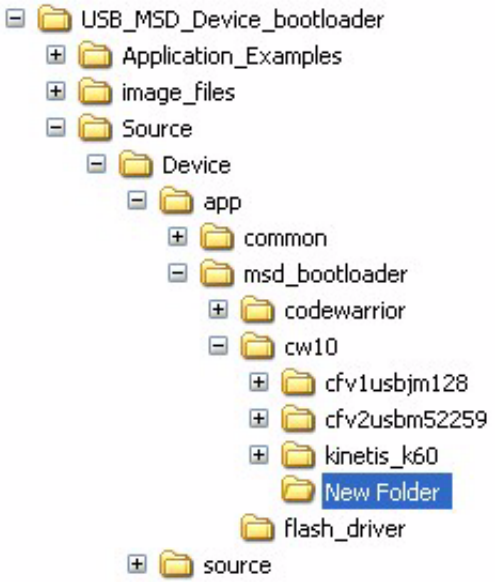

图 7. 创建新的项目文件夹

2. 创建文件结构与示例引导加载程序项目类似的一个项目。 CW10 M52259EVB 项目如下所示。

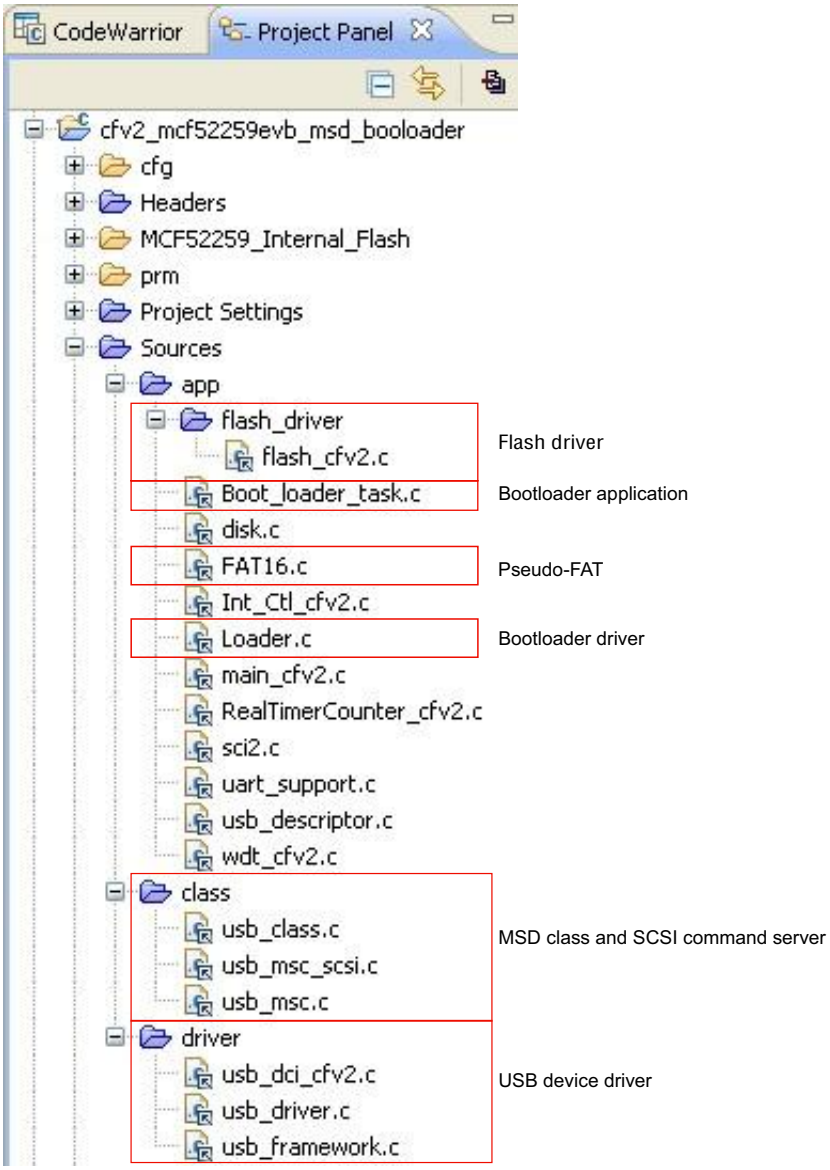

图 8. M52259EVB 引导加载程序项目

3. 向项目添加文件：
   - Flash 驱动程序源代码
   - MSD 类和 SCSI 命令服务器源代码
   - USB 设备驱动程序源代码
   - `disk.c`, `disk.h`, `Boot_loader_task.c`, `Boot_loader_task.h`, `Loader.c`, `Bootloader.h`, `FAT16.c`, `FAT16.h`, `usb_descriptor.c`, `usb_descriptor.h` 和特定于各板的其他文件。
4. 修改链接器命令文件以区分引导加载程序和应用程序，并允许引导加载程序位于受保护的 Flash 区域。
5. 针对平台板的情况修改 Boot_loader_task.c 文件。
6. 修改 Bootloader.h 文件中的内存映射。代码显示如下：

```
#if (defined __MCF52259_H__)
#define MIN_RAM1_ADDRESS        0x20000000
#define MAX_RAM1_ADDRESS        0x2000FFFF
#define MIN_FLASH1_ADDRESS      0x00000000
#define MAX_FLASH1_ADDRESS      0x0007FFFF
#define IMAGE_ADDR              ((uint_32_ptr)0xC000)
#define PROT_VALUE0x7   // Protects 0x0 - 0xBFFF
#define ERASE_SECTOR_SIZE       (0x1000) /* 4K bytes*/
#elif (defined _MCF51JM128_H)
#define MIN_RAM1_ADDRESS        0x00800000
#define MAX_RAM1_ADDRESS        0x00803FFF
#define MIN_FLASH1_ADDRESS      0x00000000
#define MAX_FLASH1_ADDRESS      0x0001FFFF
#define IMAGE_ADDR              ((uint_32_ptr)0x0A000)
#define PROT_VALUE0xD7   // Protects 0x0 - 0x9FFF
#define ERASE_SECTOR_SIZE       (0x0400) /* 1K bytes*/
#elif (defined MCU_MK60N512VMD100)
#define MIN_RAM1_ADDRESS        0x1FFF0000
#define MAX_RAM1_ADDRESS        0x20010000
#define MIN_FLASH1_ADDRESS      0x00000000
#define MAX_FLASH1_ADDRESS      0x0007FFFF
#define IMAGE_ADDR              ((uint_32_ptr)0xC000)
#define PROT_VALUE00xFF   // Protects 0x0 - 0xBFFF
#define PROT_VALUE10xFF   // Protects 0x0 - 0xBFFF
#define PROT_VALUE20xFF   // Protects 0x0 - 0xBFFF
#define PROT_VALUE30xF8   // Protects 0x0 - 0xBFFF
#define ERASE_SECTOR_SIZE       (0x800) /* 2K bytes*/
#endif
```

5 开发新的应用程序

本节介绍如何修改应用程序以配合 USB MSD 设备引导加载程序工作。

5.1 修改链接器命令文件

应用程序一般位于 Flash 存储器的开头。但是，使用引导加载程序时，它位于 Flash 存储器的开头。必须将应用程序置于引导加载程序之后。因此，必须修改应用程序链接器文件以将应用程序置于特定内存区域。

5.1.1 CFV1 链接器文件

原始应用程序链接器文件示例：

```
# Sample Linker Command File for CodeWarrior for ColdFire MCF51JM128
# Memory ranges
MEMORY {
  code     (RX)   : ORIGIN = 0x00000410, LENGTH = 0x0001FBF0
  userram  (RWX)  : ORIGIN = 0x00800000, LENGTH = 0x00004000
}
```

为了与引导加载程序兼容，应用程序代码应从受保护的 Flash 和应用程序存储的向量表之后开始，即从地址 0xA410 开始。此外，重定向向量表所用的 RAM 应从应用程序 RAM 中剔除。本例中， RAM 应从 0x8001BC 开始。修改后的链接器文件应当像如下所示：

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

# Memory ranges

MEMORY {
  code     (RX)   : ORIGIN = 0x0000A410, LENGTH = 0x0001BBF0
  userram  (RWX)  : ORIGIN = 0x008001BC, LENGTH = 0x00003E44
}
```

5.1.2 CFV2 链接器文件

原始应用程序链接器文件示例：

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

KEEP_SECTION {.vectortable}

# Memory ranges

MEMORY {
   vectorrom      (RX)   : ORIGIN = 0x00000000, LENGTH = 0x00000400
   cfmprotrom     (RX)   : ORIGIN = 0x00000400, LENGTH = 0x00000020
   code           (RX)   : ORIGIN = 0x00000500, LENGTH = 0x0007FB00
   vectorram      (RWX)  : ORIGIN = 0x20000000, LENGTH = 0x00000400
   userram        (RWX)  : ORIGIN = 0x20000400, LENGTH = 0x00005C00
}
```

为了与引导加载程序兼容，存储的应用程序中断向量 (vectorrom) 必须从受保护的 Flash 存储器之后开始，即从地址 0xC000 开始。应用程序代码应从应用程序存储的向量表之后开始，即从地址 0xC500 开始。修改后的链接器文件应当像如下所示：

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

KEEP_SECTION {.vectortable}

# Memory ranges

MEMORY {
   vectorrom      (RX)   : ORIGIN = 0x0000C000, LENGTH = 0x00000400
   cfmprotrom     (RX)   : ORIGIN = 0x0000C400, LENGTH = 0x00000020
   code           (RX)   : ORIGIN = 0x0000C500, LENGTH = 0x00073B00
   vectorram      (RWX)  : ORIGIN = 0x20000000, LENGTH = 0x00000400
   userram        (RWX)  : ORIGIN = 0x20000400, LENGTH = 0x00005C00
}
```

5.1.3 Kinetis 链接器文件

原始应用程序链接器文件示例：

```
# Default linker command file.
MEMORY {
m_interrupts      (RX) : ORIGIN = 0x00000000, LENGTH = 0x000001E0
m_text            (RX) : ORIGIN = 0x00000800, LENGTH = 0x00080000-0x00000800
m_data            (RW) : ORIGIN = 0x1FFF0000, LENGTH = 0x00020000
m_cfmprotrom      (RX) : ORIGIN = 0x00000400, LENGTH = 0x00000010
}
```

为了与引导加载程序兼容，存储的应用程序中断向量 (vectorrom) 必须从受保护的 Flash 存储器之后开始，即从地址 0xC000 开始。应用程序代码应从应用程序存储的向量表之后开始，即从地址

0xC410 开始。此外，重定向向量表所用的 RAM 应从应用程序 RAM 中剔除。本例中， RAM 应从 0x1FFF_0400 开始。修改后的链接器文件应当像如下所示：

```
# Default linker command file.
MEMORY {
m_interrupts    (RX) : ORIGIN = 0x0000C000, LENGTH = 0x00000400
m_cfmprotrom    (RX) : ORIGIN = 0x0000C400, LENGTH = 0x00000010
m_text          (RX) : ORIGIN = 0x0000C410, LENGTH = 0x00073BF0
m_data          (RW) : ORIGIN = 0x1FFF0400, LENGTH = 0x0001FC00
}
```

5.2 重定向中断向量

中断和异常向量的默认位置是引导加载程序保护的 Flash 内存区，因此在执行固件更新时，应用程序无法更新它们。为使应用程序能使用中断，需将中断向量重定向到 RAM 。

本节介绍如何利用 Freescale MQX 和 Freescale USB 协议栈将中断向量重定向到 RAM 。

5.2.1 Freescale MQX

Freescale MQX RTOS 能够将应用程序配置为在 RAM 或 ROM 中实现中断。将 user_config.h 中的宏 MQX_ROM_VECTORS 置 0， MQX 内核就会将这些向量重定向到 RAM 。向 user_config.h 添加下面的一行代码，然后重新构建 BSP 和 PSP 。有关操作详情，请参阅 MQX 文档。

将参数配置为 0 以在 RAM 中实现中断：

```
#define MQX_ROM_VECTORS    0 //1=ROM (default), 0=RAM vector
```

5.2.2 裸机应用程序

以下各节介绍如何将向量表重定向到 CFV1 、 CFV2 和 Kinetis 系列的 RAM 中。

5.2.2.1 CFV1 微控制器

为了易于使用， CodeWarrior 项目通常将向量表固定在 Flash 存储器中的默认位置（地址 0x0000 ）。因此，可以在应用程序中新建一个向量表，以便将中断向量存储在Flash存储器的应用程序部分。

下面是一个示例向量表，可将其包括在应用程序中。这里没有显示完整的向量表，不过可以在所提供的应用软件实例中找到它。该向量表应位于未受保护的 Flash 开始处，本例中为 0xA000 ，因为引导加载程序会在该地址寻找应用程序复位向量。

同默认向量表一样，向量 0 应具有初始栈指针值，向量 1 必须是应用程序复位向量。本例用空中断服务例程 dummy_ISR 填充其余向量。要添加新 ISR ，请将 dummy_ISR 的地址替换为新 ISR 的地址，如下面的 Timer_Overflow 所示。应用程序中的新向量表通过以下代码来声明：

```
// Declare the stored vector table for the application,
// and locate it at the beginning of unprotected flash
void (* const RAM_vector[])()@0xA000= {
    (pFun)&__SP_INIT,               // vector_0 INITSP
    (pFun)&_startup,                // 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_74 Vtpm1ch3
     (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
     (pFun)&dummy_ISR,              // vector_80 Vtpm2ovf

…
     (pFun)&dummy_ISR,              // vector_108 VL3swi
     (pFun)&dummy_ISR,              // vector_109 VL2swi
     (pFun)&dummy_ISR,              // vector_110 VL1swi

};
```

在应用程序中，应将此表复制到 RAM 的基地址。下面的代码将上表 (RAM_Vector) 复制到 RAM 中的向量表区域：

```
// Copy stored application interrupt vectors
// to re-directed vector table in RAM
pdst=(uint_32_ptr)0x00800000;
psrc=(uint_32_ptr)&RAM_vector;
   for (i=0;i<111;i++,pdst++,psrc++)
   {
      *pdst=*psrc;
   }
```

CFV1 有一个向量基寄存器 (VBR)，它包含异常向量表的基地址。利用该寄存器将异常向量表从 Flash 存储器中的默认位置（地址 0x0 ）重定向到 RAM 的基地址 (0x800000) 。

```
// Move vector table to start of RAM, 0x800000
asm (move.l #0x00800000,d0);
asm (movec d0,vbr);
```

5.2.2.2 CFV2 微控制器

5.2.2.2.1 Freescale USB 协议栈

对于 CFV2 版本， Freescale USB 协议栈的一个函数可将中断向量表复制到 RAM 中的指定区域。

```
void initialize_exceptions(void);
```

该函数将中断向量表复制到 RAM 中的 __VECTOR_RAM 地址。该地址在链接器命令文件中得到定义。它还通过更改 VBR 将向量表重定向到 RAM 。

此函数默认在启动时调用，因此应用程序无需调用此函数。

5.2.2.2.2 其他裸机应用程序

对于其他应用程序，应用程序需要将存储的向量表从 Flash 复制到 RAM ，并通过更改 VBR 来重定向向量表。下面的代码可用来复制向量表：

```
// Copy Application Stored Interrupt Vector table to RAM
pdst=(uint_32*)0x20000000;
psrc=(uint_32*)0xC000;
```

```
   for (i=0;i<0x100;i++,pdst++,psrc++)
   {
       *pdst=*psrc;
   }
```

CFV2 有一个向量基寄存器 (VBR)，它包含异常向量表的基地址。利用该寄存器将异常向量表从 Flash 存储器中的默认位置（地址 0x0 ）重定向到 RAM 的基地址 (0x2000_0000) 。

以下代码用于将向量表重定向到 RAM 地址 0x2000_0000 。

```
// Move vector table to RAM
asm(move.l#0x20000000, d0);
    asm(movec d0,VBR);
asm(nop);
```

5.2.2.3 Kinetis 微控制器

在 Kinetis 中， SCB_VTOR 寄存器包含异常向量表的基地址。为了重定向向量表，必须将向量表复制到 RAM ，然后将 SCB_VTOR 的值更改为复制的地址。以下步骤说明重定向 Kinetis 向量表的方法：

执行此代码，将中断向量表复制到 RAM ：

```
// Copy Application Stored Interrupt Vector table to RAM
pdst=(uint_32*)0x1FFF0000;
psrc=(uint_32*)0xC000;
   for (i=0;i<0x100;i++,pdst++,psrc++)
   {
      *pdst=*psrc;
   }
```

然后通过更改 SCB_VTOR 来将向量表重定向到 RAM ：

```
// Redirect the vector table to the new copy in RAM
  SCB_VTOR = (uint_32)0x1FFF0000;
```

5.3 应用示例

该软件包含一些简单的应用示例，用以显示如何创建一个集成应用程序和引导加载程序的项目。这些示例利用一个周期性中断触发板上的 LED ，并显示了如何重定向向量表。“ image_files ”目录中还提供了这些示例项目的二进制映像，评估引导加载程序时可以使用。

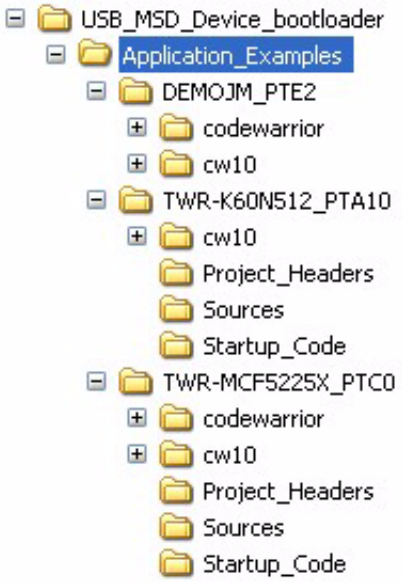

图 9. 应用示例项目

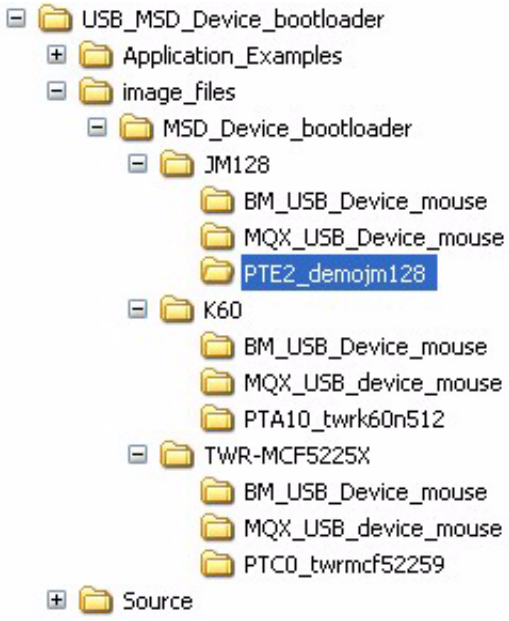

图 10. 应用示例映像文件

共有三个示例项目，支持的评估板各有一个。各示例都提供 CodeWarrior 项目。这些项目可在调试器中运行，但在不使用调试器的情况下引导时， Flash 中必须存在引导加载程序，因为应用程序向量表不是位于默认复位向量位置。示例应用程序包括：

- DEMOJM_PTE2 — 适用于运行在 DEMOJM 板上的 Flexis ColdFire V1 MCF51JM128 。翻转 PTE2 引脚来使 LED 闪烁。
- TWR-K60N512_PTA10 — 适用于运行在 TWR-K60N512-KIT 上的 Kinetis MK60N512 。翻转 PTA10 引脚来使 LED E4 闪烁。

- TWR-MCF5225X_PTC0 — 适用于运行在 TWR-MCF5225X-KIT 上的 ColdFire V2 MCF52259。翻转 PTC0 引脚来使 LED1 闪烁。

6 结语

利用Freescale USB大容量存储设备引导加载程序可以非常轻松地更新固件。无需驱动程序，主机上也不需要任何软件，任何人都可以执行。该引导加载程序还能配合各种各样的Freescale微控制器工作。借助本应用笔记中的信息，可实现任何应用程序与引导加载程序的集成。

How to Reach Us:

Home Page:
freescale.com

Web Support:
freescale.com/support

本文档中的信息仅供系统和软件实施方使用 Freescale 产品。 本文并未明示或者暗示授予利用本文档信息进行设计或者加工集成电路的版权许可。 Freescale 保留对此处任何产品进行更改的权利，恕不另行通知。

Freescale 对其产品在任何特定用途方面的适用性不做任何担保、表示或保证，也不承担因为应用程序或者使用产品或电路所产生的任何责任，明确拒绝承担包括但不局限于后果性的或附带性的损害在内的所有责任。 Freescale 的数据表和/或规格中所提供的“典型”参数在不同应用中可能并且确实不同，实际性能会随时间而有所变化。 所有运行参数，包括“经典值”在内，必须经由客户的技术专家对每个客户的应用程序进行验证。 Freescale 未转让与其专利权及其他权利相关的许可。 Freescale 销售产品时遵循以下网址中包含的标准销售条款和条件：
freescale.com/SalesTermsandConditions.

Document Number: AN4379
Rev. 0, October 2011