Freescale Semiconductor
应用笔记

Document Number: AN4370
Rev 1, 2012

用于 MCU 的 USB DFU 引导加载程序

作者: Paolo Alcantara

内容

1 简介

在不借助外部编程工具的情况下现场进行微控制器(MCU)固件升级是目前的一项必要功能。对于支持 USB 设备控制器的 Freescale MCU，USB 设备固件更新(DFU)类给出了解决方案。USB DFU 引导加载程序仅需一台 PC 和一条 USB 线缆。

本文档说明 DFU 如何装入嵌入式设备，并给出了使用 Windows OS PC 进行配置的示例。

1.1 受众

本文档面向所有软件开发工程师、测试工程师以及任何正在实现 USB DFU 类或者想将其用作解决方案的人们。

1.2 范围

本文档提供有关在 Freescale MCU 中 USB DFU 类实施的信息，例如 S08 (JM60)、ColdFire+(51JF)、ColdFire (MCF52259)以及 Kinetis K 和 L 系列（K20、K40、K60、K70、LK25）。本文档中包含下列详细信息：

- 运行 MQX RTOS 应用程序
- 运行裸机软件
- USB DFU 如何移植到其他平台

2 引导加载程序概述

USB DFU 引导加载程序可轻松、可靠地将新用户应用程序载入那些预先载入 USB DFU 引导加载程序的设备。

加载后，新用户应用程序便可在 MCU 中运行。USB DFU 引导加载程序需要一个在 PC 上运行的程序（上位机程序）。DFU PC 应用程序支持通过特定请求将固件载入设备，这些请求在 USB DFU 类的规范中说明。

支持以两种方式枚举 USB DFU 引导加载程序：

- USB 复合设备模式：亦称为运行时模式。由 DFU 设备加上另一个 USB 设备类组成。在这种实现中，将使用人机接口设备(HID)鼠标设备，以避免增加引导加载程序的存储容量。进入该模式前，MCU 必须满足下列条件：
  - MCU 不含有效固件映像，或者不含固件。
  - 向 MCU 施加一个外部操作，比如在复位事件期间按下按钮。这与 USB DFU 引导加载程序的实现有关。
- DFU 设备模式：在 USB DFU PC 端程序发出请求后，DFU 已经准备上载或下载固件镜像时使用。在进入此模式之前，MCU 应处于 USB 复合设备模式下。

2.1 引导加载程序示例概述：ColdFire V2

引导加载程序是一个小型应用程序，用于将新的用户应用程序载入设备。因此，引导加载程序需要能够在用户应用程序模式和引导加载程序模式下运行。例如，图 1 描述了 ColdFire V2 引导加载程序设置的存储器映射。

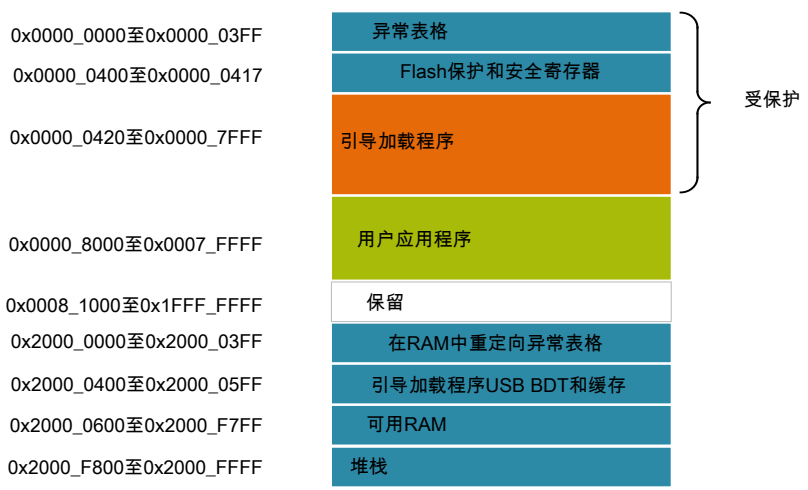

图 1. ColdFire V2 引导加载程序存储器映射

复位后，设备会尝试运行用户应用程序。若未找到用户应用程序，或者用户应用程序已损坏，则设备自动进入引导加载程序模式。如果应用程序有效，并且用户想运行引导加载程序，则需要进行外部干预，例如在复位时按下某个键以强制设备进入引导加载程序模式。引导加载程序异常表格位于 Flash 存储器区域，并在引导加载程序运行时使用。因此，加载新的用户应用程序时，引导加载程序无法更新异常表格。如果用户应用程序要求中断，则必须将用户应用程序异常表格重定向至 RAM。

引导加载程序解析用户应用程序映像，并将映像存入 Flash 存储器的用户应用程序区内，如图 1 所示。

如图 1 所示，引导加载程序保留 0x0000_0000 到 0x0000_7FFF (32KB)范围的 Flash 存储器区域。该 Flash 存储器区域需要进行编程保护，以防引导加载程序损坏。从 0x0000_8000 到 0x0007_FFFF (480 KB)的余下 Flash 存储器供用户应用程序使用。重定向至 RAM 后，中断和异常表格位于 RAM 存储器中 0x2000_0000 到 0x2000_03FF (1 KB)范围的区域内。

运行期间，用户应用程序可以使用整个 RAM 存储器，无论引导加载程序所需的 RAM 空间如何。RAM 中的异常表空间不应通过链接文件（LCF）配置成用户应用程序的数据空间，既不是.data 段也不是.bss 段。

下表显示不同 MCU 的 DFU 引导加载程序空间要求：

表 1. DFU 引导加载程序存储器空间占用量

MCU	需要引导加载程序 Flash 存储器
CFV1、ColdFire+	40KB
CFV2	36KB
Kinetis（L 和 K 系列）	40KB
S08	~21KB

3 引导加载程序架构和引导序列

下节概要说明 USB DFU 引导加载程序的架构及其软件流程。

3.1 架构概述

USB DFU 引导加载程序的架构如下图所示：

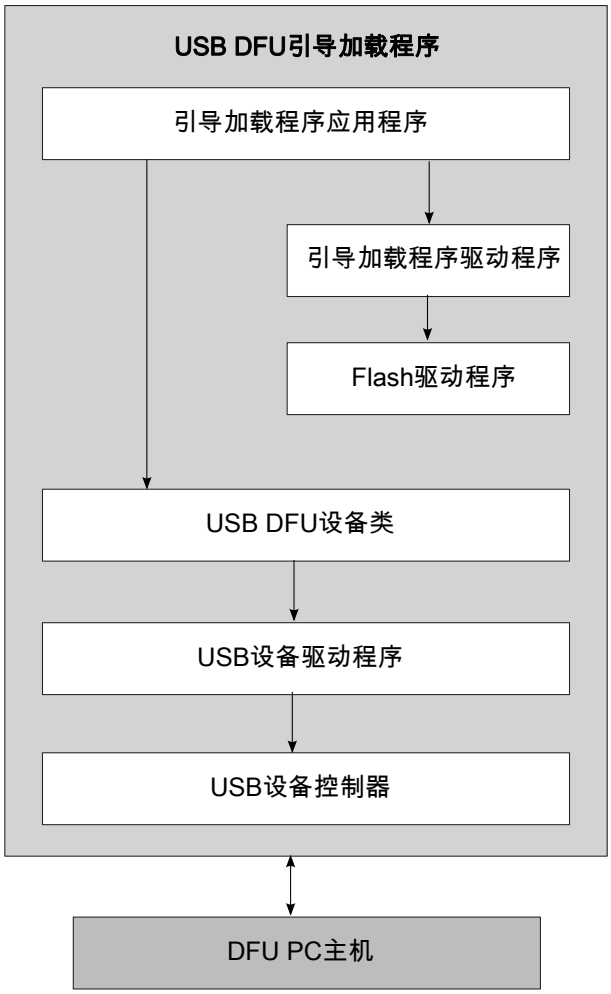

图 2. USB DFU 引导加载程序架构

USB DFU 引导加载程序架构包含下列功能块:

- 引导加载程序应用程序: 控制加载过程。它使用 DFU 类的特定请求来收发固件映像文件，然后通过引导加载程序驱动器将用户应用程序文件加载至设备的 flash 存储器，或者从设备的 flash 存储器加载用户应用程序文件。
- 引导加载程序驱动程序: 解析固件映像文件并将其存入 flash 存储器。引导加载程序驱动程序支持对 CodeWarrior 二进制、S19 和原始二进制文件格式的映像文件进行解析。
- Flash 驱动程序: 支持擦除、读取和写入 Flash 存储器等功能。
- USB DFU 设备类: 包含在 DFU 类中指定的 API。
- USB 设备驱动程序和设备控制器: 通过 USB 标准与 USB 主机(PC)通信。

USB DFU PC 应用程序支持将固件下载或上载至设备，或从设备下载或上载固件。

3.2 引导加载程序工作流程

引导加载程序用来加载执行产品主要功能的应用程序。复位时，引导加载程序执行某些简单的检查以确定是否能启动应用程序或引导加载程序模式。一旦处于 DFU 引导加载程序模式，引导加载程序便能接收来自 USB DFU PC 应用程序的请求。若收到的请求是要下载固件，则 DFU 引导加载程序将在缓存中积累数据。缓存满后，它开始解析缓存，并将其下载到用户应用程序区。详情参见图 1 。

USB DFU 引导加载程序的流程如以下流程图所示:

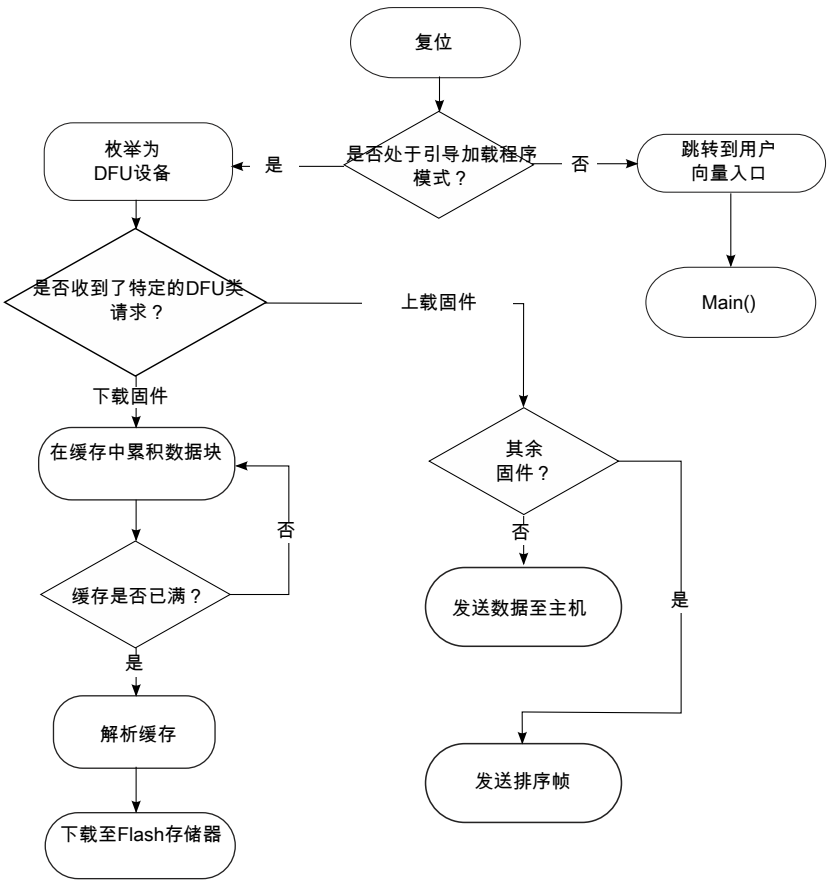

图 3. USB DFU 引导加载程序工作流程

4 开发带有引导加载程序的应用程序

下节介绍如何修改用户应用程序，以供 USB DFU 引导加载程序使用。

4.1 链接器文件修改

应用程序一般位于 Flash 存储器的开头。然而，引导加载程序需要使用 Flash 存储器空间，因此用户应用程序必须置于 Flash 存储器的其余位置。详情请参见图 1 。

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存储器区域。

下节解释针对 ColdFire V1、ColdFire+、ColdFire V2-4、Kinetis（K 和 L 系列）以及 S08 MCU 需要修改的链接器文件。

4.1.1 CFV1 链接器文件：ColdFire V1 和 ColdFire+

常见的 CFV1 链接器文件如下所示：

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

# Memory ranges

MEMORY {
   code         (RX)   : ORIGIN = 0x00000410, LENGTH = 0x0001FBF0
   userram      (RWX)  : ORIGIN = 0x00800000, LENGTH = 0x00004000
}
```

如需运行 USB DFU 引导加载程序，用户应用程序必须指定 flash 存储器区域于地址 0x0000_A000 处开始。修改后的链接器文件如下所示：

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

# Memory ranges

MEMORY {
   code         (RX)   : ORIGIN = 0x0000A410, LENGTH = 0x00017BF0
   userram      (RWX)  : ORIGIN = 0x00800000, LENGTH = 0x00004000
}
```

4.1.2 CFV2 链接器文件：ColdFire V2-4

常见的 CFV2 链接器文件如下所示：

```
# 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
}
```

如需运行 USB DFU 引导加载程序，用户应用程序必须指定 flash 存储器区域于地址 0x0000_9000 处开始。修改后的链接器文件如下所示：

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

KEEP_SECTION {.vectortable}

# Memory ranges

MEMORY {
   vectorrom    (RX)   : ORIGIN = 0x00009000, LENGTH = 0x00000400
   cfmprotrom   (RX)   : ORIGIN = 0x00009400, LENGTH = 0x00000020
   code         (RX)   : ORIGIN = 0x00009500, LENGTH = 0x00077B00
   vectorram    (RWX)  : ORIGIN = 0x20000000, LENGTH = 0x00000400
   userram      (RWX)  : ORIGIN = 0x20000400, LENGTH = 0x00005C00
}
```

4.1.3 Kinetis（K 和 L 系列）链接器文件

常见的 Kinetis 链接器文件如下所示：

```
MEMORY
{
```

```
   vectorrom    (RX): ORIGIN = 0x00000000, LENGTH = 0x00000400
   cfmprotrom   (RX): ORIGIN = 0x00000400, LENGTH = 0x00000020
   rom          (RX): ORIGIN = 0x00000420, LENGTH = 0x0001FBE0  # Code + Const data
   ram          (RW): ORIGIN = 0x00800000, LENGTH = 0x00004000  # SRAM - RW data
}
```

如需运行 USB DFU 引导加载程序，用户应用程序必须指定 flash 存储器区域于地址 0x0000_A000 处开始。修改后的链接器文件如下所示：

```
MEMORY
{
   vectorrom    (RX): ORIGIN = 0x0000A000, LENGTH = 0x00000400
   cfmprotrom   (RX): ORIGIN = 0x0000A400, LENGTH = 0x00000020
   rom          (RX): ORIGIN = 0x0000A420, LENGTH = 0x00017BE0  # Code + Const data
   ram          (RW): ORIGIN = 0x00800000, LENGTH = 0x00004000  # SRAM - RW data
}
```

4.1.4 S08 链接器文件

常见的 S08 链接器文件如下所示：

```
SEGMENTS /* Here all RAM/ROM areas of the device are listed. Used in PLACEMENT below. */
    Z_RAM                   =  READ_WRITE   0x00B0 TO 0x00FF;
    RAM                     =  READ_WRITE   0x0100 TO 0x10AF;
    RAM1                    =  READ_WRITE   0x1860 TO 0x195F;
    ROM                     =  READ_ONLY    0x1960 TO 0xFFAD;
    ROM1                    =  READ_ONLY    0x10B0 TO 0x17FF;
    ROM2                    =  READ_ONLY    0xFFC0 TO 0xFFC3;
```

如需运行 USB DFU 引导加载程序，用户应用程序必须指定 flash 存储器区域于地址 0xABA5 处结束。修改后的链接器文件如下所示：

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

    // Application Segments
    Z_RAM                   =  READ_WRITE   0x00B0 TO 0x00FF;
    RAM           =  READ_WRITE   0x0110 TO 0x10AF;
    RAM1      =  READ_WRITE   0x1860 TO 0x195F;
    ROM                     =  READ_ONLY    0x1960 TO 0xABA5;
    ROM1                    =  READ_ONLY    0x10B0 TO 0x17FF;
    ROM2                    =  READ_ONLY    0xFFC0 TO 0xFFC3;
```

注

对于 CFV1、CFV2、ColdFire+和 Kinetis（L 和 K 系列）链接器文件，用户应用程序数据空间的起始位置与 MCU RAM 的起始位置一致。异常表格重定位期间（如异常表格重定向 中所述），编译器会保留声明的 RAM 异常表格空间，而不会与其他数据（.data 或.bss）共享此空间。

4.2 异常表格重定向

默认情况下，异常向量位于 Flash 存储器中，并为引导加载程序所用，因此加载新的用户应用程序时，引导加载程序无法更新它。

如果用户应用程序要求中断，则必须将异常表格重定向至 RAM，但 S08 MCU 除外。

每个 MCU 重定向异常表格至 RAM 的步骤各有不同。

下节描述如何在 MQX 和裸机用户应用程序中重定向异常表格。

4.2.1 MQX 用户应用程序

借助 userconfig.h 中包含的 C 语言宏 MQX_ROM_VECTORS，MQX 实时操作系统可将异常表格重定向至 RAM。

下列源代码示例显示如何将数值 0 分配至 MQX_ROM_VECTORS 宏。

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

注

MQX 实时操作系统仅支持 ColdFire、ColdFire+和 Kinetis MCU。8 位 MCU 必须采用裸机应用程序。

4.2.2 裸机用户应用程序

以下各节介绍如何针对 ColdFire V1、ColdFire+、ColdFire V2-4、Kinetis 和 S08 MCU，将异常表格重定向至 RAM。

4.2.2.1 CFV1 MCU：ColdFire V1 和 ColdFire+

CFV1 MCU 有一个名为向量基寄存器(VBR)的 CPU 寄存器，其中包含异常向量表的基址。利用该寄存器可以将异常向量表从 Flash 存储器中的默认位置（地址 0x0000_0000）重定向到 RAM 的基址(0x0080_0000)。

使用引导加载程序时，在应用程序源代码中声明中断服务例程（ISR）是不同的。

异常表格重定向过程可总结如下：

1. 在用户应用程序代码区域内声明异常表格并在该空间内分配 ISR。
2. 在用户应用程序数据区域内保留异常表格空间。它必须位于 RAM 空间的起始部分。
3. 运行时，将已声明的异常表格复制到保留的异常表格空间。
4. 将已保留异常表格的地址写入 VBR，即 RAM 空间的起始位置。

新的异常表格必须如下一行所示进行声明。如需添加新的 ISR，dummy_ISR 的地址向量必须以新 ISR 的名称代替。这一新异常表格的地址必须是用户应用程序代码空间的一部分。该示例在地址 0x0000_A000 处声明。详情参见图 1 。用户应用程序中的新异常表格声明如下：

```
void  (* const RAM_Vector[])()@0x0000A000=
{
(pFun)&dummy_ISR,              // vector_0  INITSP
(pFun)&dummy_ISR,              // vector_1  INITPC
……
(pFun)&dummy_ISR,              // vector_67 Vspi1
(pFun)&dummy_ISR,              // vector_68 Vspi2
(pFun)&dummy_ISR,              // 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
……
}
```

接着，必须在运行时将声明的异常表格(RAM_Vector)复制到 RAM 基址。该任务由下列源代码执行：

```
pdst=(dword)&New_RAM_vector;//0x00800000;//RAM base address
psrc=(dword)&RAM_vector;

for (i=0;i<111;i++,pdst++,psrc++)//112 exceptions
  {
    *pdst=*psrc;
  }
```

最后，使用下列软件通过地址 0x0080_0000 将异常表格重定向至 RAM:

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

4.2.2.2 CFV2 MCU：ColdFire V2-4

与 CFV1 相似，异常表格必须在运行时从用户应用程序空间复制到 RAM。下列源代码显示 `initialize_exceptions` 函数，该函数从用户应用程序空间(FLASH)复制到 RAM 基址:

```
void initialize_exceptions(void)
{
/*
 * Memory map definitions from linker command files used by mcf5xxx_startup
 */

register uint32 n;

/*
     * Copy the vector table to RAM
     */
if (__VECTOR_RAM != (unsigned long*)_vect)
{
for (n = 0; n < 256; n++)
__VECTOR_RAM[n] = (unsigned long)_vect[n];
}
mcf5xxx_wr_vbr((unsigned long)__VECTOR_RAM);
}
```

使用 CFV2 版本，则装有 PHDC v3.0 的 Freescale USB 堆栈便同时支持 `initialize_exceptions` 函数，可将中断异常表格复制到 RAM 中的指定区域。

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

`initialize_exceptions` 函数将中断向量表复制到*__VECTOR_RAM* 地址处的 RAM 区域。需要在链接器文件中定义此地址。

如将装有 PHDC v3.0 的 USB 堆栈用作用户应用程序项目模板，则默认启动时调用 `initialize_exceptions` 函数。

4.2.2.3 Kinetis（L 和 K 系列）MCU

在 Kinetis MCU 中，SCB_VTOR 寄存器包含异常表格的基地址。如需重定向异常表格，则必须将异常表格复制到 RAM。然后，必须向 SCB_VTOR 中写入复制的地址值。

下列步骤详述如何必须在 Kinetis 中执行重定向。

1. 声明 ROM 区，用于存储异常表格（链接器文件)。

```
.interrupts :
{
  ___VECTOR_ROM = .;
  * (.vectortable)
  . = ALIGN (0x4);
} > interrupts
```

2. 将异常表格从默认的用户应用程序代码空间复制到 RAM 地址，并与 128 字节对齐。

```
extern uint_32 ___VECTOR_RAM[];
extern uint_32 ___VECTOR_ROM[];            //Get vector table in ROM

uint_32 i,n;
/* Copy the vector table to RAM */
if (___VECTOR_RAM != ___VECTOR_ROM)
{
```

```
    for (n = 0; n < 0x410/4; n++)
    ___VECTOR_RAM[n] = ___VECTOR_ROM[n];
    }
    /* Point the VTOR to the new copy of the vector table */
    SCB_VTOR = (uint_32)___VECTOR_RAM;
```

4.2.2.4 S08 MCU

MC9S08 内核无法像 ColdFire 或 Kinetis 那样将异常表格重定向至 RAM。相反，引导加载程序在用户应用程序空间的重定向异常表格处指向应用程序的异常表格。

重定向异常表格保存在特定地址处。用户应用程序必须在特定的地址处声明一个指向异常表格的函数指针，才能实现中断。

对于 DFU 引导加载程序，阵列 UserJumpVectors 表示指向异常表格的函数指针，起始地址为 VectorAddressTableAddress，即 0xABA6（根据 S08 规格）。

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

Addr 是函数指针类型，如下所示：

```
typedef void (* Addr)(void);
```

引导加载程序使用 Redirect_Vectors_S08.c 文件中的数组 BootIntVectors 加载引导加载程序 Flash 存储器中的中断向量表。

```
volatile const Addr BootISRTable[InterruptVectorsNum] = {
    Dummy_ISR,                  //  0 - Reset
    Dummy_ISR,                  //  1 - SWI
    Dummy_ISR,                  //  2 - IRQ
    Dummy_ISR,                  //  3 - Low Voltage Detect
    Dummy_ISR,                  //  4 - MCG Loss of Lock
    Dummy_ISR,                  //  5 - SPI1
    Dummy_ISR,                  //  6 - SPI2
```

```
    USB_ISR,                    //  7 - USB Status
    Dummy_ISR,                  //  8 - Reserved
    Dummy_ISR,                  //  9 - TPM1 Channel0
    Dummy_ISR,                  //  10 - TPM1 Channel1
    Dummy_ISR,                  //  11 - TPM1 Channel2
    Dummy_ISR,                  //  12 - TPM1 Channel3
    Dummy_ISR,                  //  13 - TPM1 Channel4
    Dummy_ISR,                  //  14 - TPM1 Channel5
    Dummy_ISR,                  //  15 - TPM1 Overflow
    Dummy_ISR,                  //  16 - TPM2 Channel0
    Dummy_ISR,                  //  17 - TPM2 Channel1
    Dummy_ISR,                  //  18 - TPM2 Overflow
    Dummy_ISR,                  //  19 - TPM1 SCI1 Error
    Dummy_ISR,                  //  20 - TPM1 SCI1 Receive
    Dummy_ISR,                  //  21 - TPM1 SCI1 Transmit
    Dummy_ISR,                  //  22 - TPM1 SCI2 Error
    Dummy_ISR,                  //  23 - TPM1 SCI2 Receive
    Dummy_ISR,                  //  24 - TPM1 SCI2 Transmit
    Dummy_ISR,                  //  25 - TPM1 KBI
    Dummy_ISR,                  //  26 - TPM1 ADC Conversion
    Dummy_ISR,                  //  27 - TPM1 ACMP
    Dummy_ISR,                  //  28 - IIC
    Dummy_ISR,                  //  29 - RTC
};
```

Redirect_Vectors_S08.c 文件包含一些函数，用于决定调用引导加载程序中断函数还是用户应用程序。中断发生时，将调用 Redirect_Vectors_S08.c 文件中相应的中断函数，该函数随后决定是调用引导加载程序的中断函数，或是调用用户应用程序。

```
extern uint_8 boot_mode;
/* VectorNumber_Vswi */
interrupt VectorNumber_Vswi vector1(void)
{
    if(boot_mode == BOOT_MODE)
    {
        BootISRTable[VectorNumber_Vswi]();
    }
    else
    {
        AppISRTable[VectorNumber_Vswi]();
    }
}
```

对于新的应用程序，必须将 Bootloader.h 和 Vectortable.c 两个文件加入应用程序项目中。然后，通过正确的应用程序 ISR 加载 Vectortable.c 中的数组 UserJumpVectors。

5 引导加载程序示例：引导 MQX

下节解释如何使用 USB DFU 引导加载程序，并提供一个 MQX 引导示例。示例采用 M52259EVB 板和 CodeWarrior 版本 7.2。

5.1 准备设置

DFU 引导加载程序需要进行软件和硬件配置。以下各节介绍在 MQX 中运行引导加载程序示例的步骤。

5.1.1 软件要求

运行 DFU 应用程序需使用下列软件：

- DFU PC 主机应用程序
- CodeWarrior 版本 7.2
- 串口终端

有关如何使用这些 PC 应用程序的详细信息，请参见后续章节。

5.1.2 硬件设置

硬件要求如下：

- 运行 Windows XP、Windows Vista 或 Windows 7（32 位或 64 位版本）的 PC
- M52259EVB 板和+5V 电源
- 两条 USB 线缆：
  - USB 2.0 A-B
  - USB 2.0 A 至 miniB
- DB9 线缆或 USB2SER 转换器

硬件配置方式如下：

1. 将电源连接到电路板。
2. 利用 USB 2.0 A-B 线缆将电路板的 USB 调试端口连接到 PC。
3. 利用 DB9 线缆或 USB2SER 转换器将 MCF52259EVB COM1 端口连接到 PC。
4. 为电路板上电。

5.2 准备固件镜像文件

必须遵循下列步骤生成有效的 MQX 映像，供 USB DFU 引导加载程序使用：

1. 在 user_config.h 文件中将 MQX_ROM_VECTORS 设为 0，以使用 RAM 中的异常表格

   ```
   #define MQX_ROM_VECTORS     0
   ```

2. 通过运行 Freescale MQX 3.7.0\config\m52259evb\cwcf72\build_m52259evb_libs.mcp 项目，编译 MQX 库。若使用 CW10.x，则应分别编译下图中列出的每个库（bsp_m52259evb、psp_m52259evb 等）。

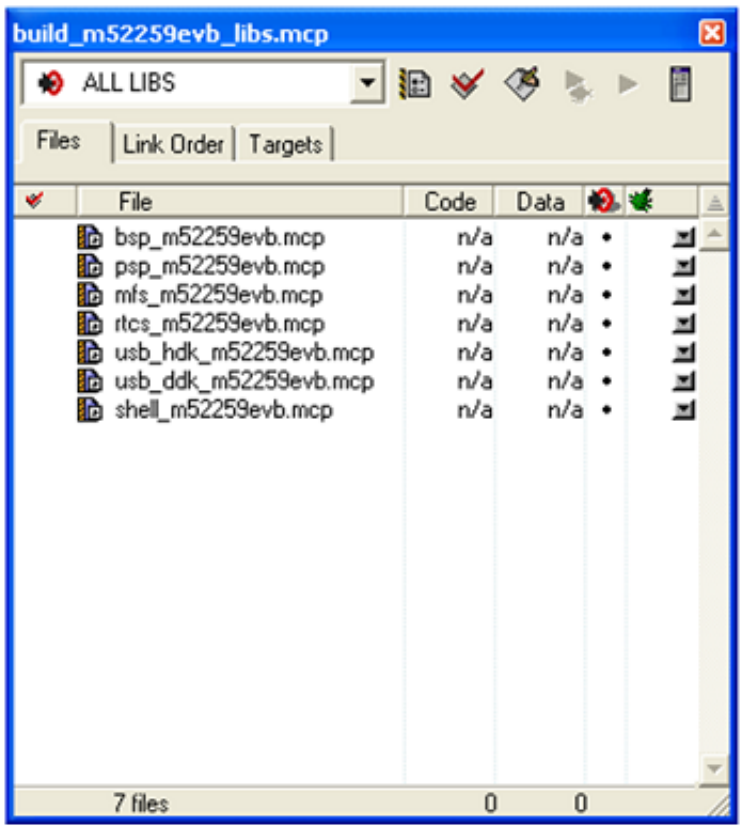

图 4. 编译 MQX 库

3. 创建 MQX 应用程序。本节测试中使用的是项目“Freescale MQX 3.7.0\mfs\examples\mfs_usb”。
4. 选择“Flash Debug”或“Flash Release”目标。

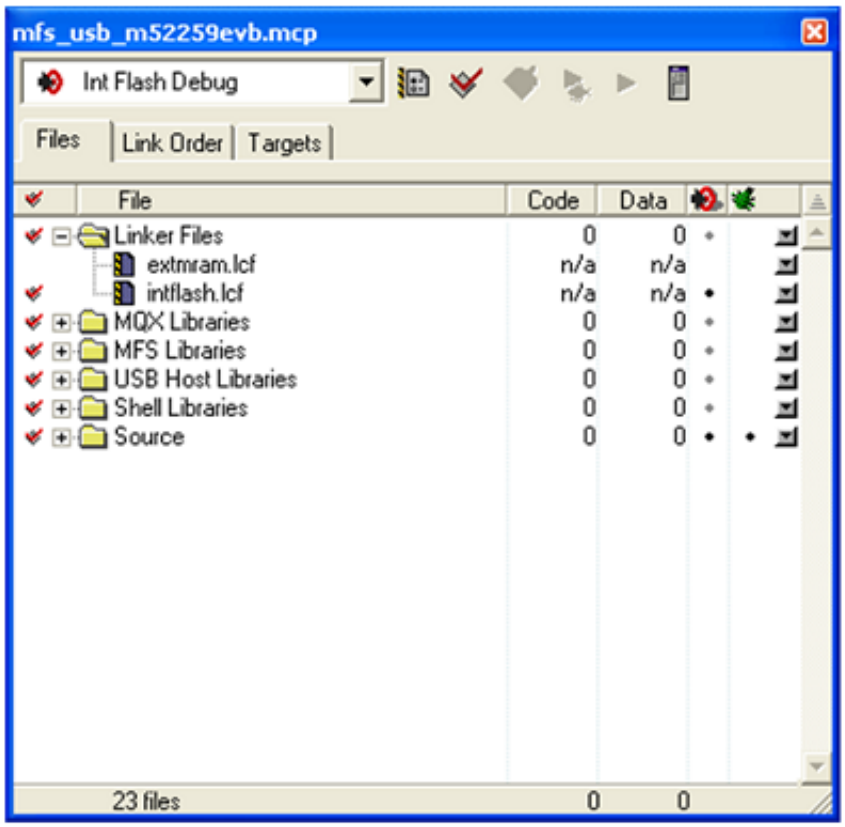

图 5. MQX 示例

5. 修改 intflash.lcf 链接器文件，将代码部分（vectorrom、cfmprotrom 和 ROM 存储器段）移至 USB DFU 引导加载程序的用户应用程序区。用户应用程序区始于 0x0000_9000。

```
vectorrom    (RX): ORIGIN = 0x00009000, LENGTH = 0x00000400
cfmprotrom   (RX): ORIGIN = 0x00009400, LENGTH = 0x00000020
rom          (RX): ORIGIN = 0x00009420, LENGTH = 0x00075BE0 # Code+Const
data
```

6. 配置项目以生成 s19 和二进制映像文件。这些是适用于 USB DFU PC 应用程序的有效文件格式。

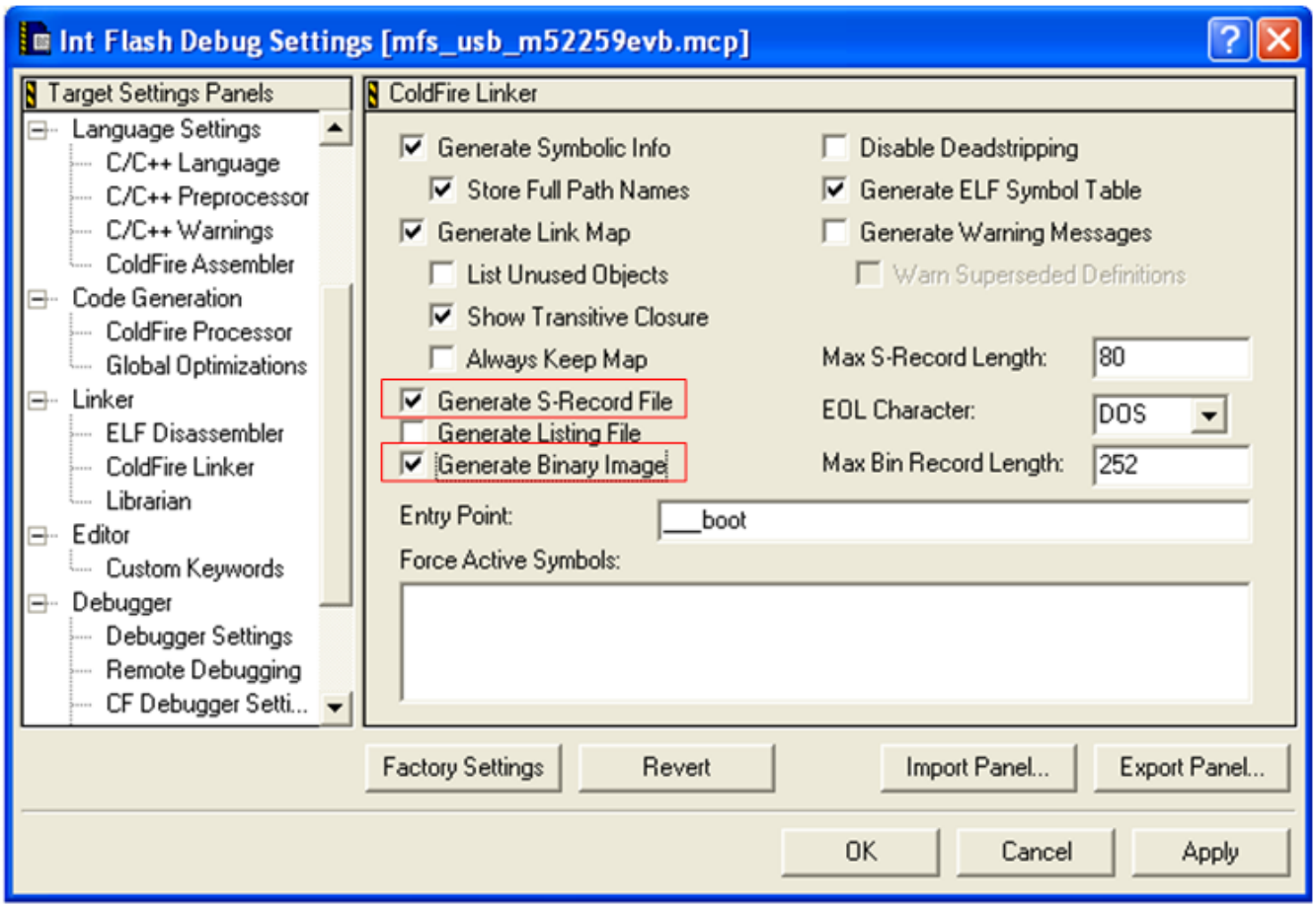

图 6. 生成 s19 和二进制固件映像的选项

7. 编译用户应用程序。完成编译过程后，m52259evb 文件夹包含两个有效文件格式：
   - intflash_d.elf.S19
   - intflash_d.elf.bin

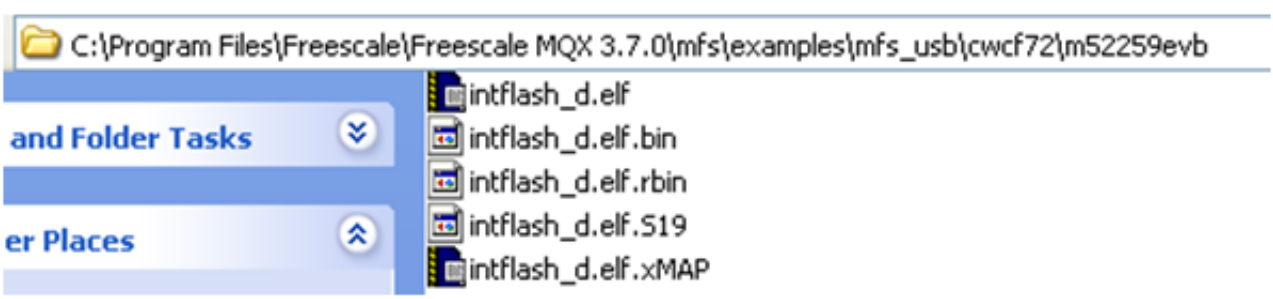

图 7. 固件镜像文件

生成的 s19 文件起始地址为 0x0000_9000。

8. 上一步中生成的 s19 和二进制文件将用于下载固件 。

5.3 编译应用程序

1. 在 CodeWarrior 版本 7.2 IDE 上，打开针对 M52259EVB 平台的 USB DFU 引导加载程序项目并进行构建。该 mcp 文件的路径如下：
   - \Source\Device\app\dfu_bootloader\codewarrior\cfv2usbm52259
   - 或使用 CW10.2：Source\Device\app\dfu_bootloader\cw10\cfv2usbm52259
2. 利用 CodeWarrior Flash Programmer 实用程序，将项目载入 MCF52259 Flash 存储器。

5.4 运行应用程序

下节介绍如何在 Windows PC 上安装 USB DFU 引导加载程序设备。

5.2 节中使用的测试固件采用串口终端与用户通信。请在 115.2Kbps 8-N-1 无流量控制条件下打开串口控制台。

5.4.1 驱动程序安装

USB DFU PC 应用程序使用 WinUSB 2.0。WinUSB 是 Microsoft 提供的通用 USB 驱动程序。若要安装 USB DFU 引导加载程序设备驱动程序，请按照以下步骤操作：

1. 复位 M52259EVB，然后利用 USB 2.0 A 至 miniB 线缆将其连接到 PC。强烈建议将 USB 线缆直接连接至 PC 的 USB 端口。Windows 要求提供 USB 驱动程序以便使用新的设备。将出现“找到新硬件”窗口，如下图所示。

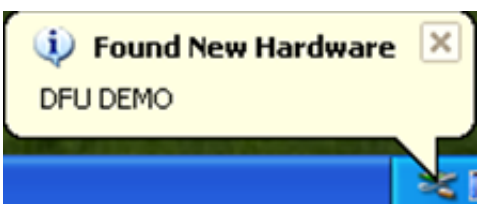

图 8. 找到新硬件

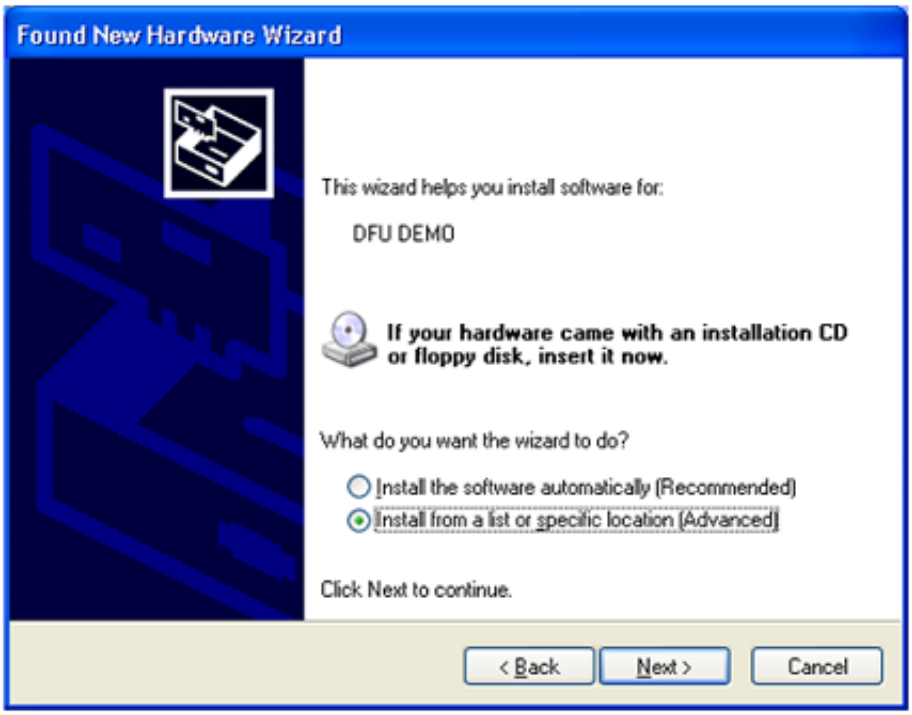

图 9. “找到新硬件”向导

2. 选择“从列表或指定位置安装(高级)”选项，然后单击“下一步”按钮。下图显示的是 Windows 弹出的当前消息。选择“不要搜索。我要自己选择要安装的驱动程序”选项，然后单击“下一步”。

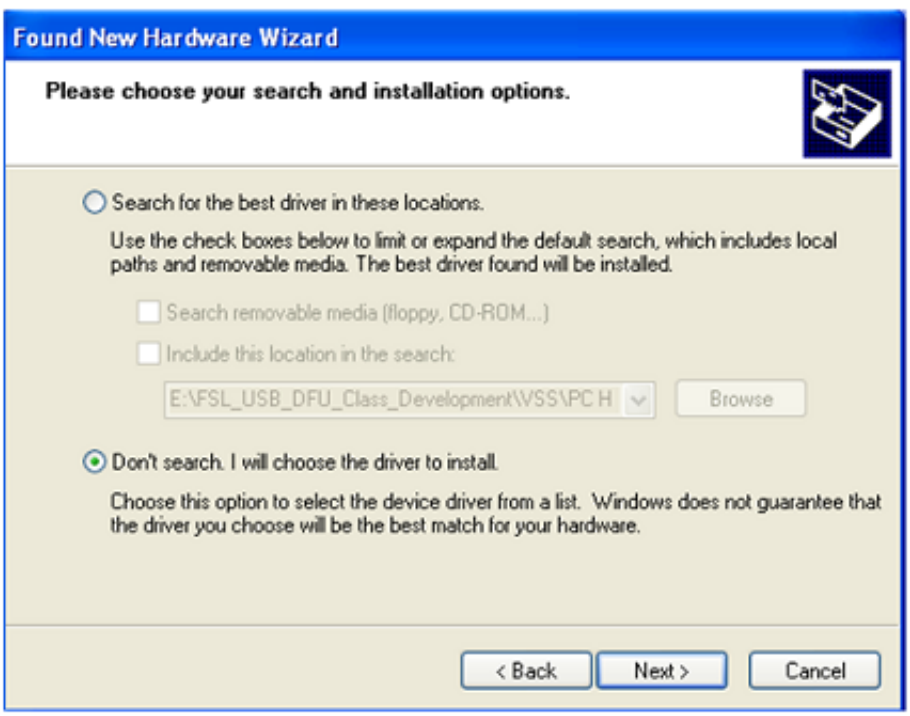

图 10. 搜索和安装选项

3. 将出现“硬件类型”窗口。选择“显示所有设备”选项，然后单击“下一步”按钮。出现“选择设备驱动程序”窗口时，选择“从磁盘安装...”按钮。

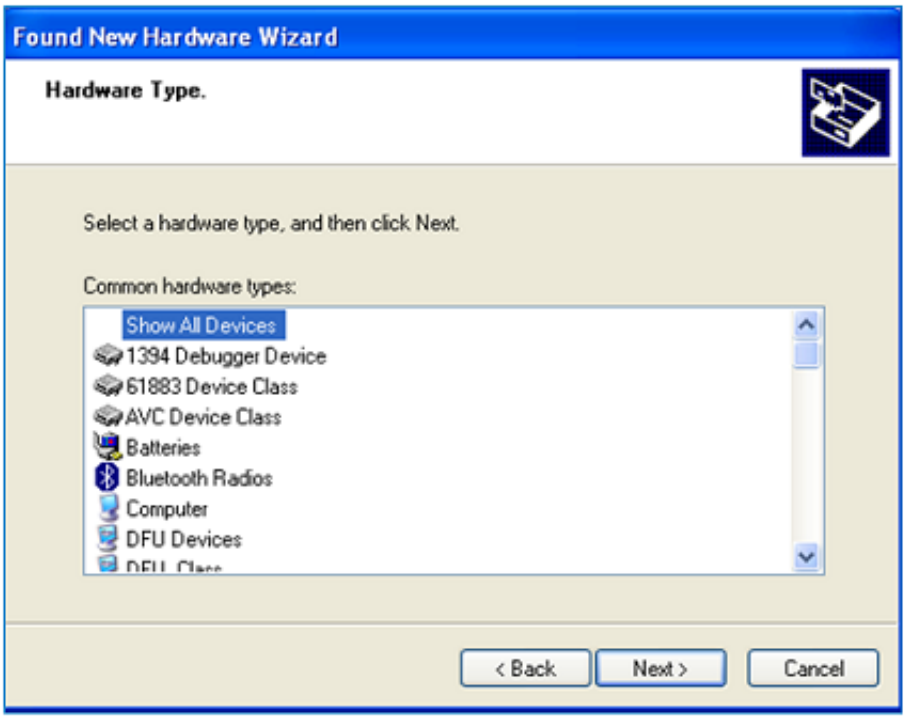

图 11. 硬件类型

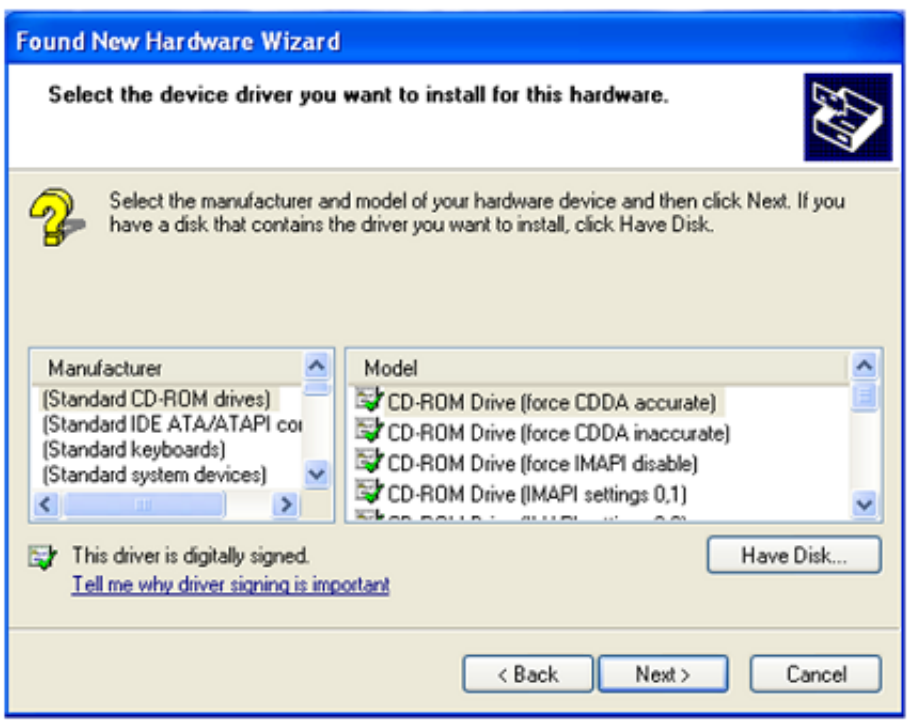

图 12. 选择设备驱动程序

4. 浏览至位于\DFU_winusb_driver 中的 INF 文件，然后选择 DFU_Device_Runtime.inf 文件。单击“打开”，然后单击“下一步”以便安装 USB 驱动程序。

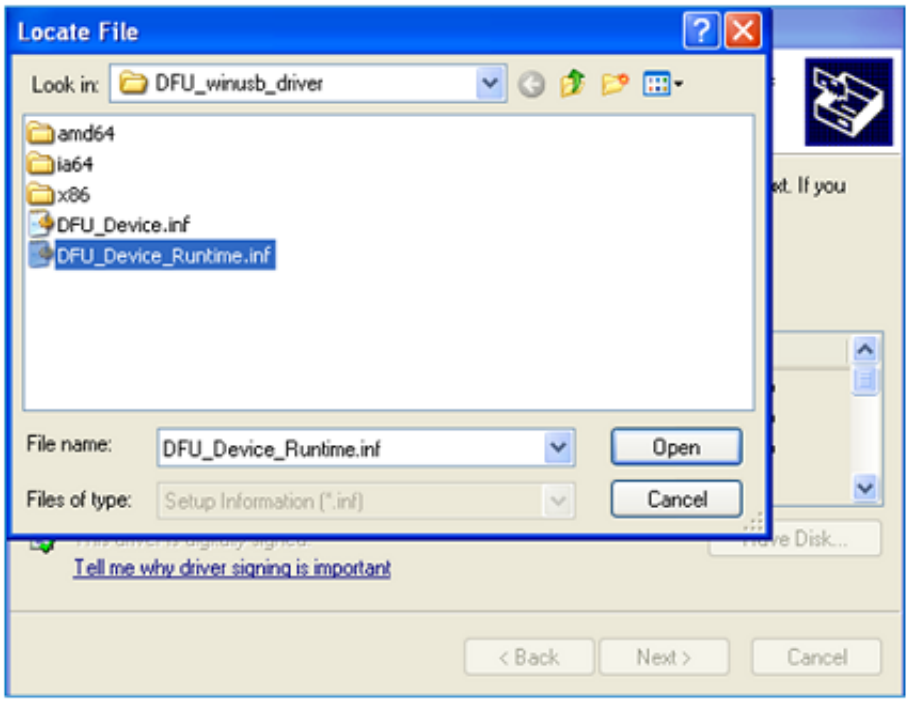

图 13. 选择驱动程序

5. 驱动程序安装完成后，Windows 将其识别为 DFU 类和 HID 鼠标组成的复合设备，如引导加载程序概述 中所述。

   若要验证 USB 安装，请打开“Windows 设备管理器”。设备管理器将显示“设备固件更新” (DFU)和“USB 人机接口设备”条目，如下图所示。

图 14. 设备管理器中的 DFU 设备和人机接口设备

6. 打开 USB DFU PC 应用程序。PC 应用程序会自动识别运行时模式（USB 复合设备）正在运行，如下图所示。单击“进入 DFU 模式”，将设备切换到 DFU 模式。

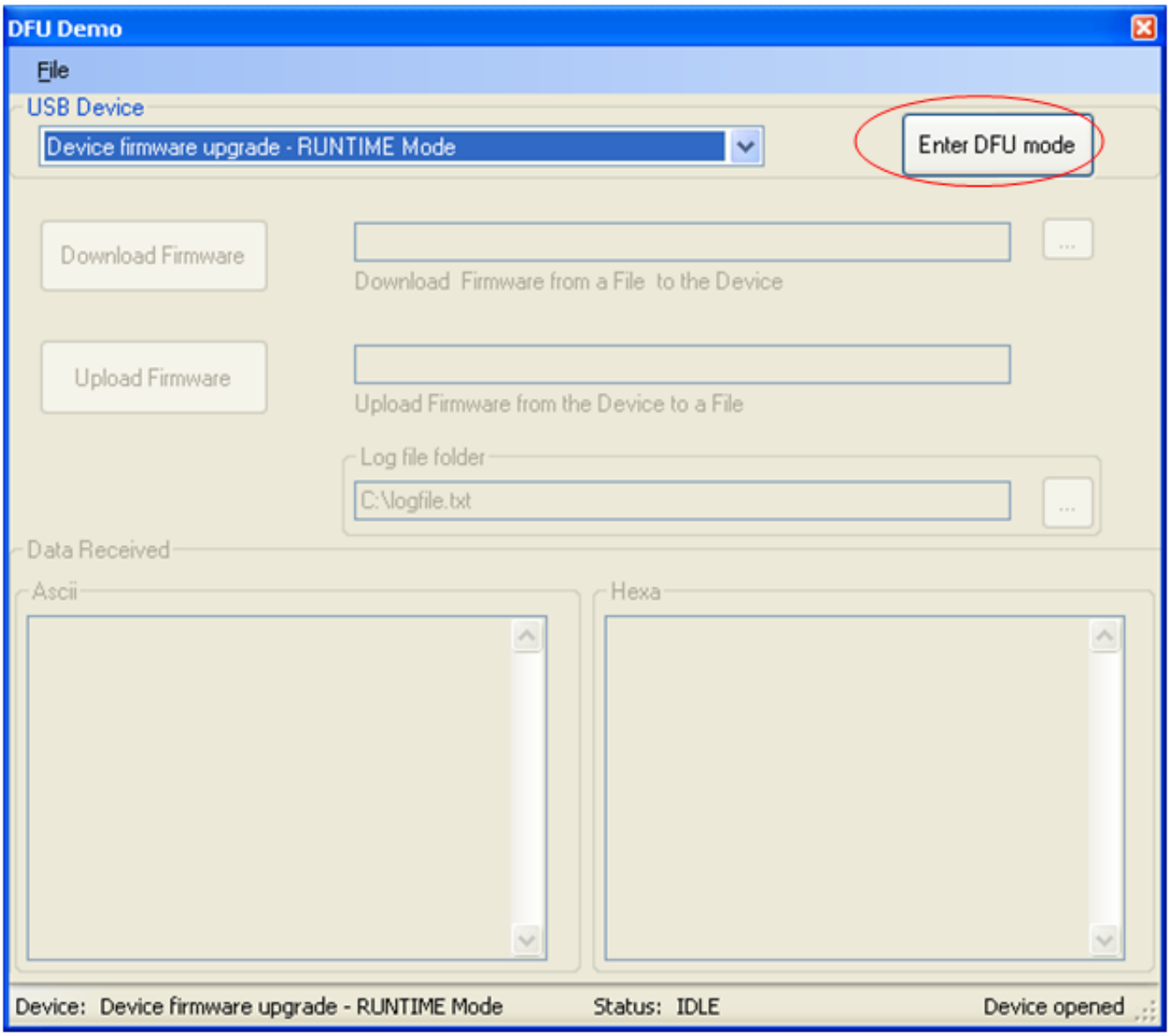

图 15. 设备固件更新 - 运行时模式

7. 拔掉再插上 USB 线缆，以便复位 USB 总线。M52259EVB USB 设备将进入 DFU 模式。
8. 进入 DFU 模式后，Windows 操作系统将再次要求提供驱动程序。遵循步骤 2 至步骤 4，安装 USB DFU 驱动程序，这次选择 DFU_Device.inf，如下图所示。

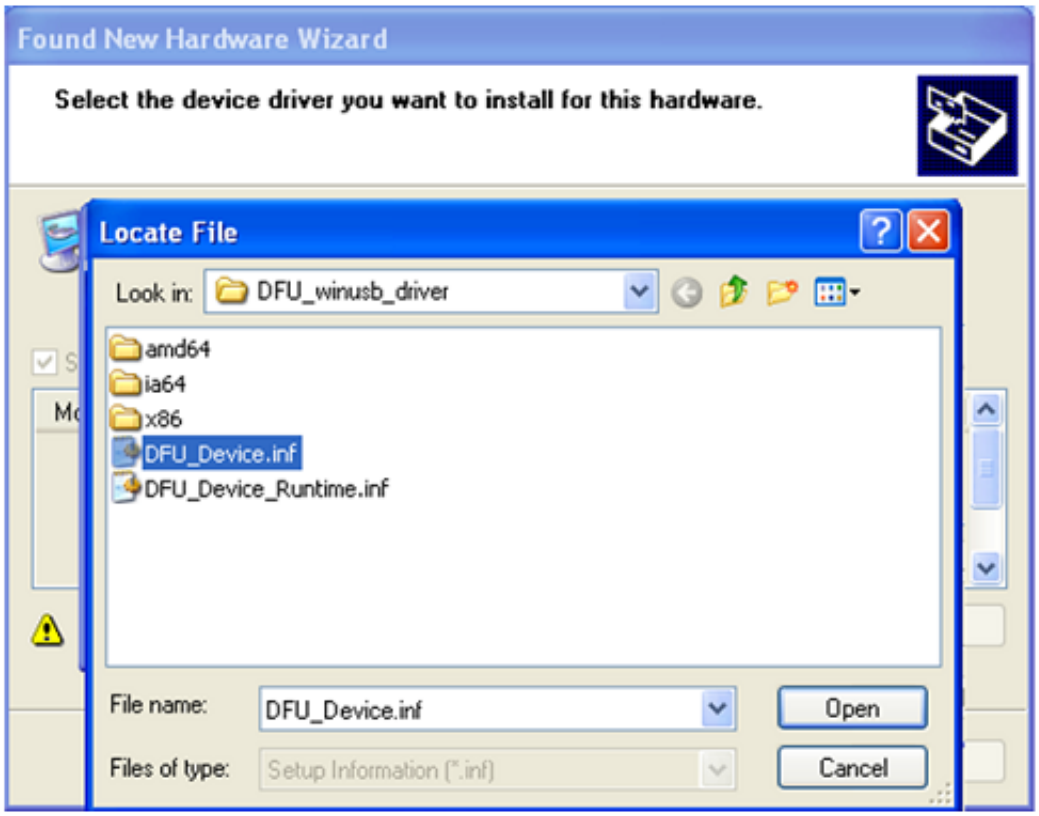

图 16. 安装 DFU 模式驱动程序

9. 成功安装 DFU 模式驱动程序后, USB DFU 设备的引导加载程序便处于 DFU 模式且可供使用。USB DFU PC 应用程序如下所示:

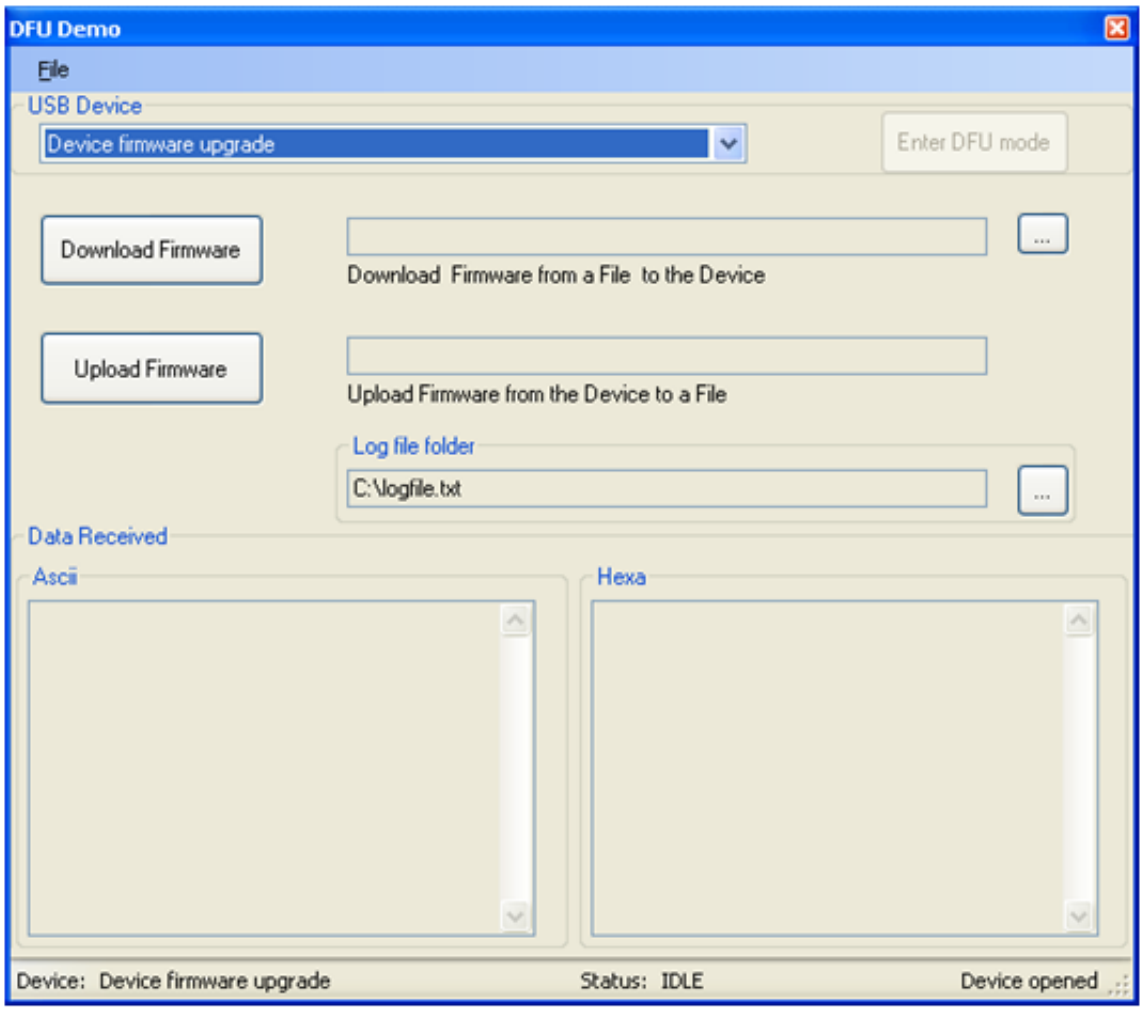

图 17. DFU 设备演示（DFU 模式）

注

针对 USB DFU 设备引导加载程序，不建议使用 USB 集线器或坞站。

5.5 下载固件

必须遵循下列步骤，通过 USB DFU 引导加载程序下载固件。

1. 此时，必须已完成驱动程序安装 。使用 USB DFU PC 应用程序，选择要下载至设备的固件映像文件，如图 21 所示。编译应用程序 中生成的文件可用于此步骤。

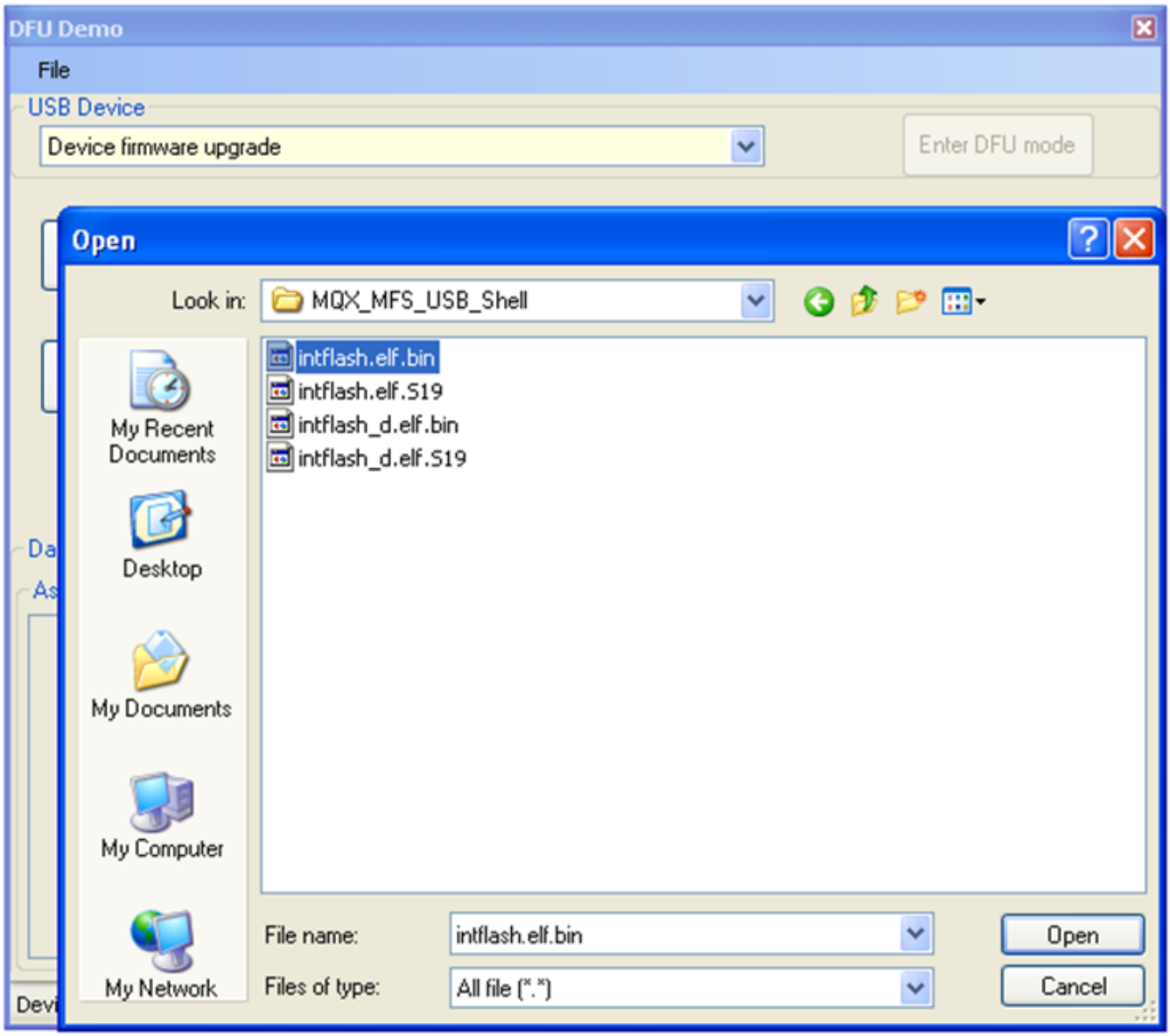

图 18. 选择固件文件

2. 选择 S19 文件时，固件文件的内容以 ASCII 和十六进制(HEX)格式显示。若选择 CodeWarrior 二进制格式，则固件内容仅以十六进制(HEX)格式显示，如下图所示。

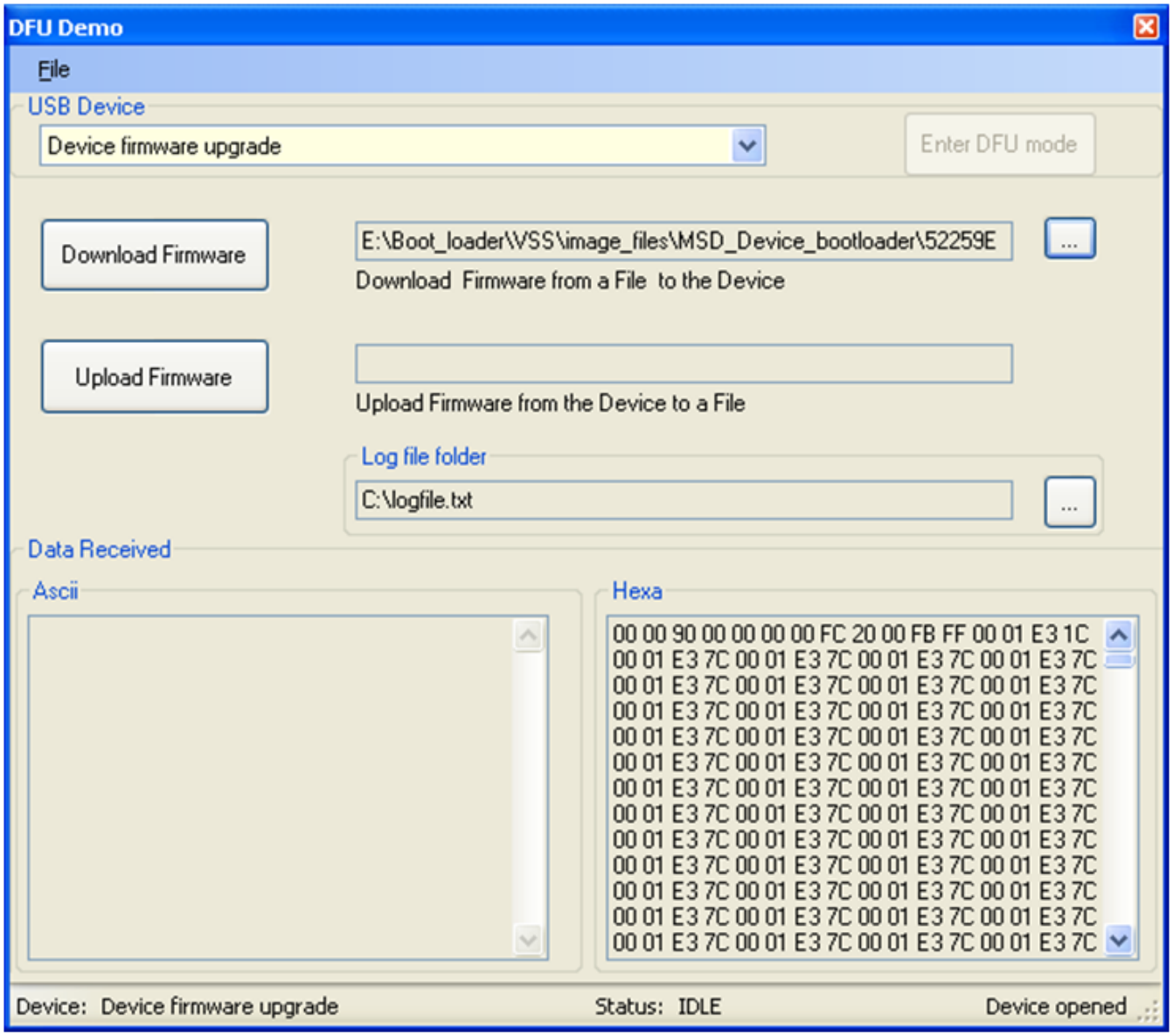

图 19. 显示固件内容

3. 单击“下载固件”按钮。固件将下载至设备。

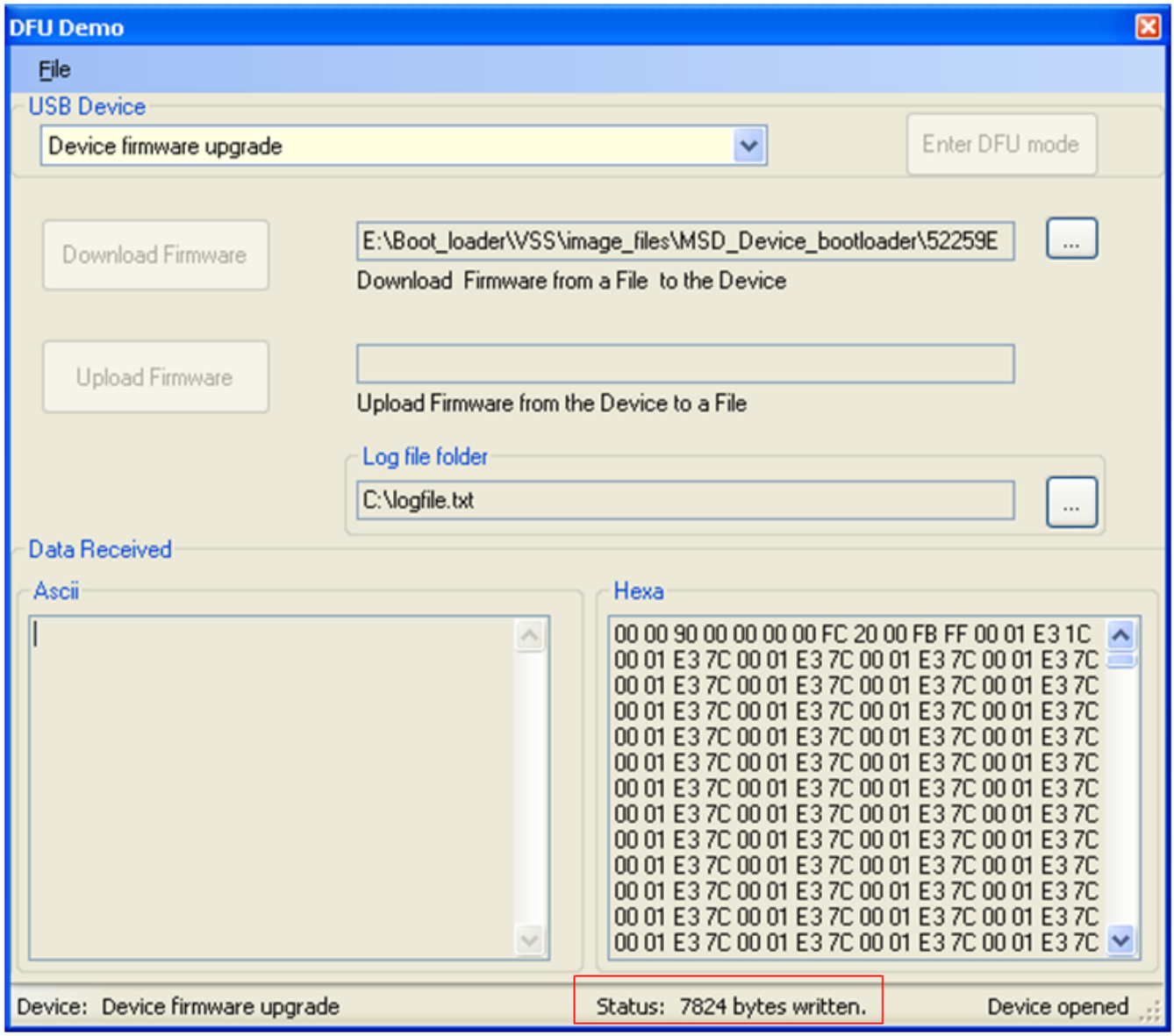

图 20. 固件已下载

4. 下载固件过程完成后，USB DFU PC 应用程序会显示下载过程的最终状态。

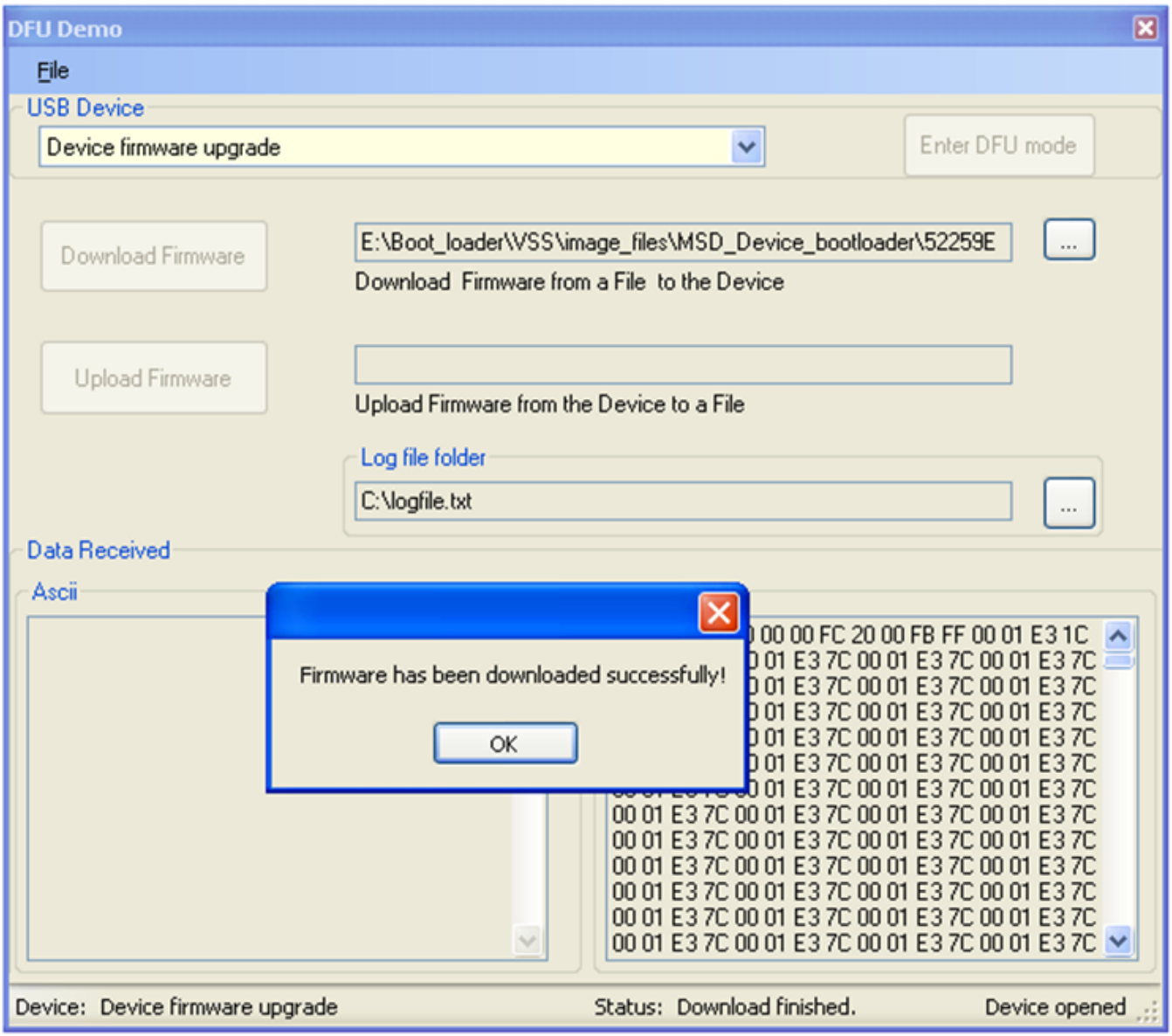

图 21. 下载已完成

5. 下载过程中会产生一个包含事件的日志文件，作为额外的验证步骤。

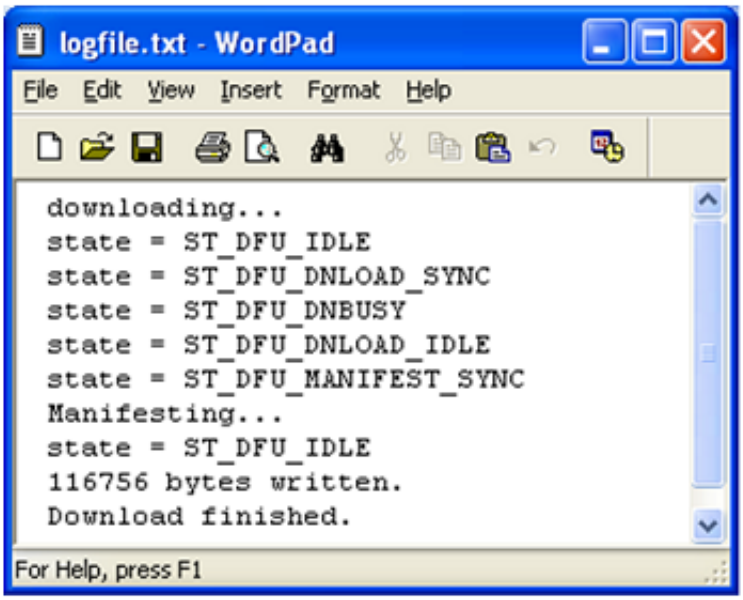

图 22. 日志文件内容

6. 按电路板上的复位键，运行用户应用程序。串行终端显示 MQX 用户应用程序发送的菜单。

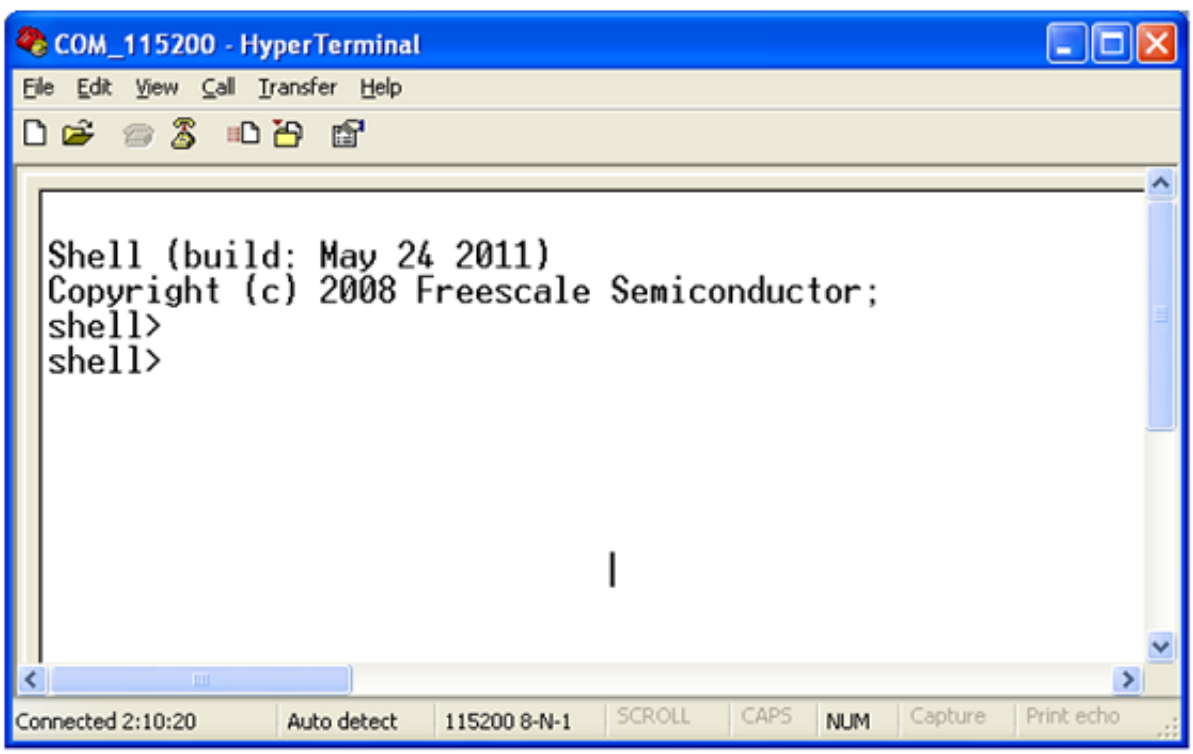

图 23. 用户应用程序正在运行

注

若在下载过程中拔出 USB 线缆，则 USB DFU PC 应用程序将在 USB 线缆重新插回后询问是否继续下载，如图 24 所示。

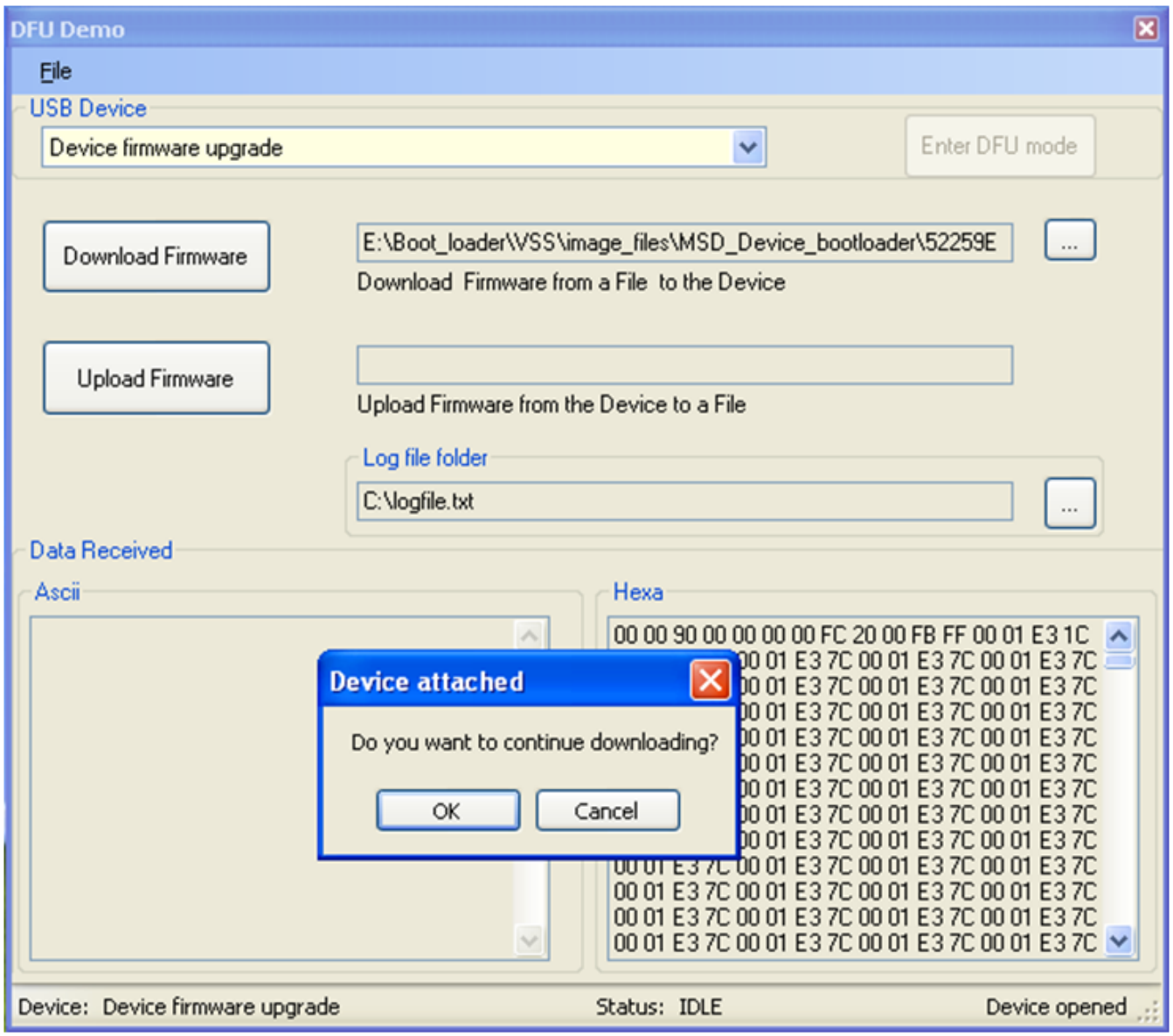

图 24. 恢复下载

6 将引导加载程序移植到其他平台

下节解释如何在其他平台上开发新的 USB DFU 引导加载程序。采用“Freescale USB Stack with PHDC v3.0”软件，可开发 USB DFU 引导加载程序。

6.1 USB DFU 引导加载程序文件结构

下图显示了 DFU 源代码的文件夹结构:

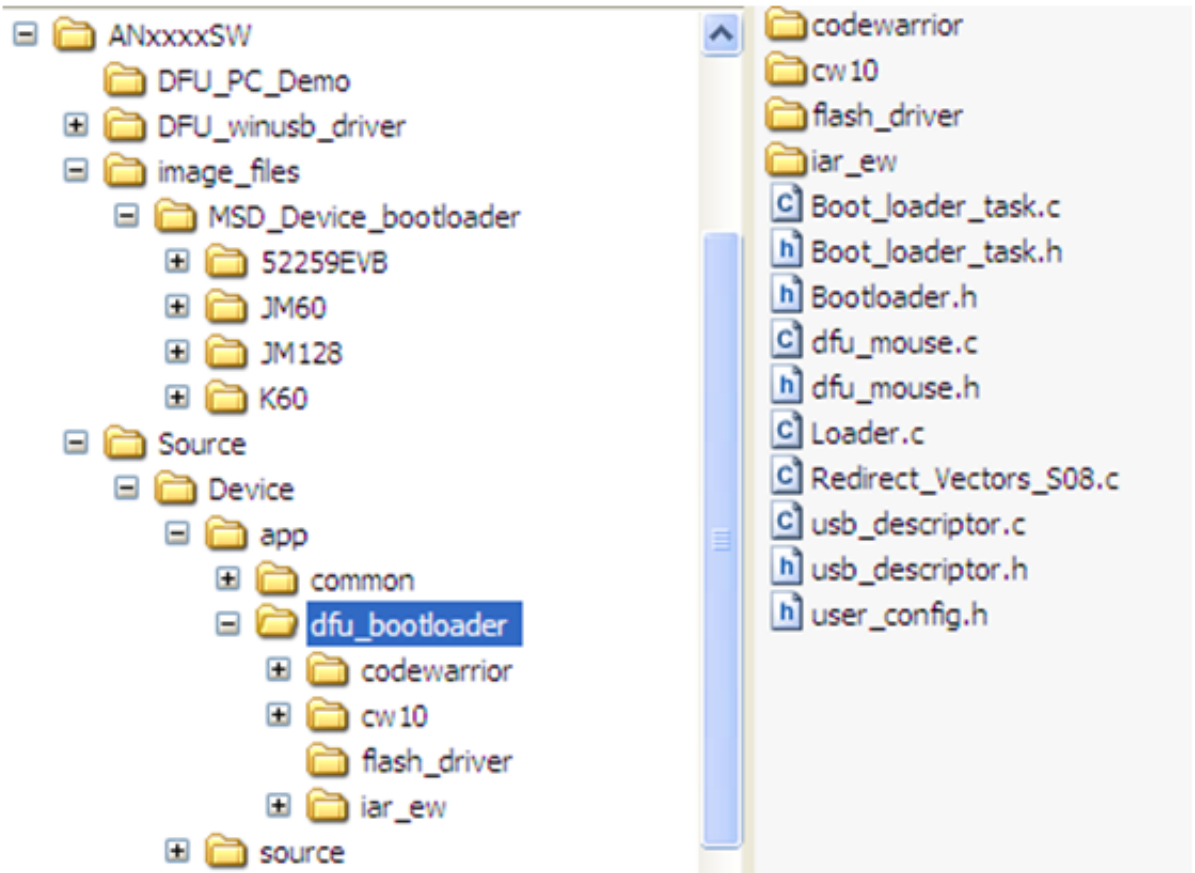

图 25. USB DFU 引导加载程序文件结构

顶层文件夹包含:

- DFU_PC_Demo: 包含 USB DFU PC 应用程序
- DFU_winusb_driver: 包含 Windows 操作系统需要的 USB 驱动程序
- image_files: 包含 MC9S08JM60、MCF51JM128、MCF52259 和 K60 MCU 的示例固件映像文件
- Source: 包含 USB DFU 引导加载程序源代码

dfu_bootloader 文件夹包含下列文件夹:

- codewarrior: 包含 CodeWarrior v6.3 和 v7.2 项目
- cw10: 包含 CodeWarrior 10.2 项目
- iar_ew: 包含 IAR 项目
- flash_driver: 包含可支持 MCU 的 flash 驱动程序

下面是 dfu_bootloader 目录中的部分文件

- Boot_loader_task.c: 包含引导加载程序一般任务
- Boot_loader_task.h: 包含函数原型
- Bootloader.h: 包含移植开发板到 DFU 引导加载程序的存储器映射定义
- dfu_mouse.c: 包含 DFU 应用程序和鼠标功能
- dfu_mouse.h: 包含 DFU 参数定义
- Loader.c: 包含用以解析固件映像并将其载入 MCU Flash 存储器的函数
- Redirect_Vectors_S08.c: 包含 MC9S08JM60 (S08 MCU)的引导加载程序中断
- usb_descriptor.c: 包含 USB 描述符结构体和函数
- usb_descriptor.h: 包含 USB 描述符参数
- user_config.h: 包含用户配置

6.2 建立新项目

如需建立新的 USB DFU 引导加载程序项目：

1. 在下列位置建立新项目：
   - Source\Device\app\dfu_bootloader\codewarrior，或
   - Source\Device\app\dfu_bootloader\cw10

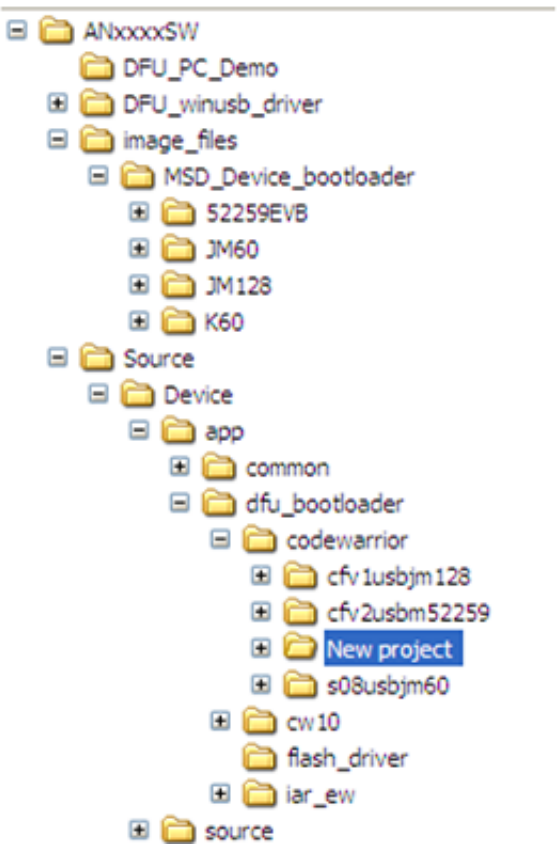

图 26. 创建新的项目文件夹

2. 创建项目，使其文件结构与 M52259EVB 的引导加载程序项目相似。使用 cfv2usbm52259 项目作为 CodeWarrior 模板。

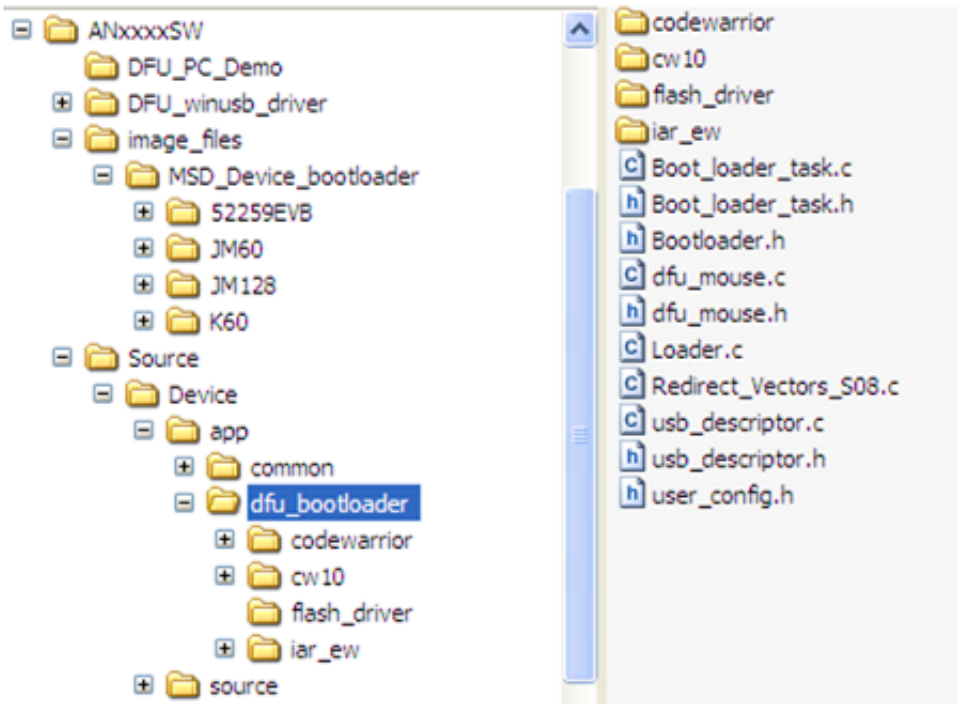

图 27. M52259 引导加载程序项目

3. 向项目添加文件：
   - Flash 驱动程序源代码：
     - flash.c：CFV1 和 ColdFire+ Flash 驱动程序
     - flash_cfv2.c：CFV2 Flash 驱动程序
     - flash_FTFL：Kinetis（L 和 K 系列）Flash 驱动程序
     - flash_hcs：S08 Flash 驱动程序
     - flash_NAND.c：NAND Flash 驱动程序
   - USB 类（DFU 和 HID 类）源代码
   - USB 设备驱动程序源代码
   - dfu_mouse.c、dfu_mouse.h、Boot_loader_task.c、Boot_loader_task.h、Loader.c、Bootloader.h、usb_descriptor.c、usb_descriptor.h 和特定于各板的必要文件。
4. 针对应用 DFU 引导加载程序的特定板修改 Boot_loader_task.c 文件。
5. 如下所示，在 Bootloader.h 中修改表示平台应用程序区域的存储器映射：

```
#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)0x9000)
#define ERASE_SECTOR_SIZE        (0x1000)  /* 4K bytes*/
#define FIRMWARE_SIZE_ADD       (0x0007FFF0 )
#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 ERASE_SECTOR_SIZE        (0x0400)  /* 1K bytes*/
#define FIRMWARE_SIZE_ADD       (0x0001FFF0 )
#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)0xA000)
```

```
#define ERASE_SECTOR_SIZE       (0x800)  /* 2K bytes*/
#define FIRMWARE_SIZE_ADD      (0x0007FFF0 )
#endif
```

7 结语

USB DFU 类可用来作为现场升级 MCU 固件的一个选择。通过 DFU 引导加载程序运行的应用程序仅需修改链接器文件和异常表格即可。本文档概述的解决方案适用于任何 Freescale 8/16/32 位 MCU。

7.1 问题报告说明

如果对本文档和驱动程序有任何问题和建议，请通过技术支持页面提交：freescale.com/support 。届时，请引用本应用笔记。

7.2 考虑因素与参考文献

- 如需查找 MCU USB DFU 引导加载程序的最新软件更新并了解相关信息，请访问飞思卡尔半导体主页：freescale.com 。
- 有关在 Freescale MCU 中使用 USB DFU 类的更多配置信息，请参考最新的“带 PHDC 的 Freescale USB 堆栈”软件：freescale.com/usb 。
- 有关 USB DFU 类的更多详情，请参考“USB 设备固件更新规格”：usb.org 。
- AN4370SW 软件包含在嵌入式设备和 Windows PC 上运行 USB DFU 类所需的全部软件。
- 下载 AN4370SW 软件(AN4370SW.zip)源文件：freescale.com/ 。

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 AN4370
Revision 1, 2012