

# USB BootLoader for MCF51JM128

by: Derek Liu  
Systems and Applications Engineering  
Microcontroller Solutions Group

## 1 Introduction

MCF51JM128 is a member of the Freescale Flex JM family. It uses the ColdFire® V1 core and integrates abundant peripherals, such as a full-speed USB OTG module, SPI, IIC, CAN, SCI, and ADC. This application note describes a bootloader that allows you to upgrade the MCF51JM128 firmware via USB interface.

[Figure 1](#) is a hardware connection example for the USB bootloader. The PC communicates with the MCF51JM128 target system via a USB interface. With a Freescale-specific USB protocol, the PC can update the MCF51JM128 firmware. The update speed is faster than background debugging mode (BDM) because USB has a higher transfer rate than BDM communication.

The system architecture is shown in [Figure 1](#). The USB bootloader needs software support from the PC and the MCF51JM128.

In this application note, a firmware framework is introduced to support the bootloader. You can create

## Contents

1	Introduction	1
2	Bootloader Framework Overview	2
2.1	Project Files Introduction	3
2.2	Memory Map	3
2.3	Bootloader Flowchart	6
3	Create a Project with the Bootloader	7
3.1	Create the Project	7
3.2	File Modifications	13
4	Using the PC Tool	16
4.1	Installing the Bootloader GUI	16
4.2	Identify the USB Bootloader Device	16
4.3	Update the Firmware via the Bootloader	17
5	Porting to MCF51JM64/32 MCU	18
6	Summary	19
	Appendix A Bootloader Framework Example Code	19
	Appendix B Bootloader PC Software	19

## Bootloader Framework Overview

various applications based on this framework. The following sections describe in detail the framework and the steps for creating an application based on it.

The PC software can be used for all applications. Its installation and usage are also discussed in this document.

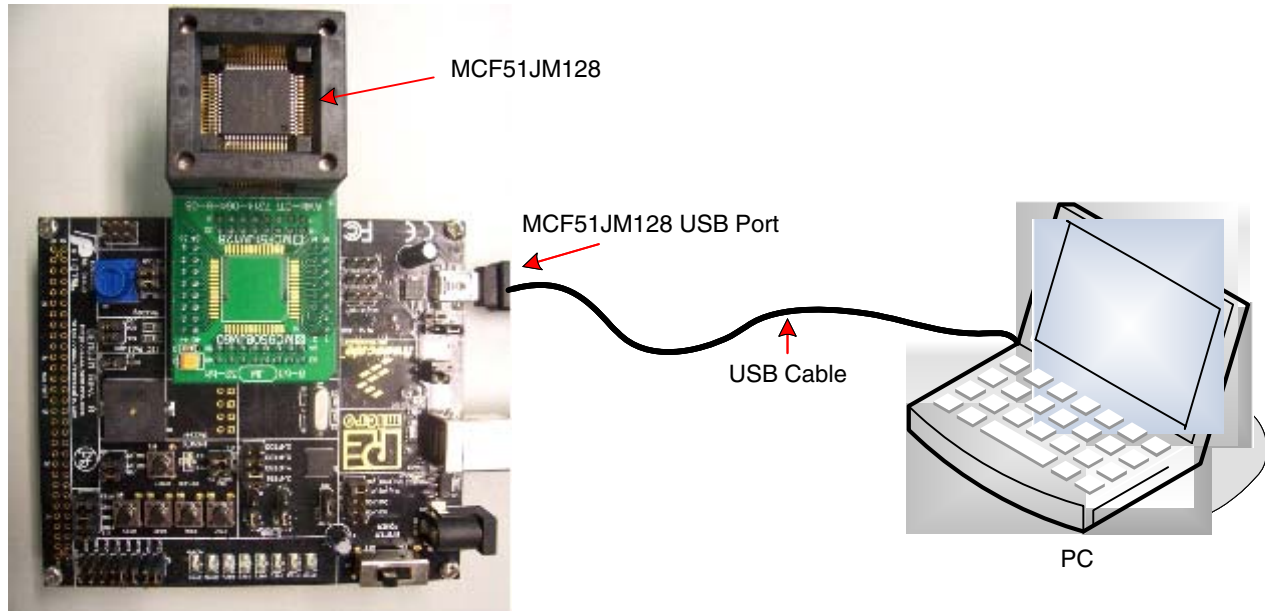


Figure 1. USB Bootloader

You can port the MCF51JM128 USB bootloader to other members of the JM family with a ColdFire V1 core. Refer to [Section 5, “Porting to MCF51JM64/32 MCU,”](#) for more details.

The USB bootloader for the MCU with a ColdFire V1 core has some differences from the one for the MCU with S08 core. You can refer to AN3561, “USB Bootloader for the MC9S08JM60,” for more information about the USB bootloader for the S08 MCU.

## 2 Bootloader Framework Overview

The bootloader framework is provided as a CodeWarrior™ project attached to this application note. You can port this framework to your application or develop based directly on it. The porting process is easy and fast. Refer to [Section 3, “Create a Project with the Bootloader,”](#) for more information.

After integrating the bootloader framework, you can debug your application and execute it step by step. You can also insert a breakpoint at any time during debug. The bootloader does not impact the system debug.

[Figure 2](#) illustrates the project for the MCF51JM128 USB bootloader framework.

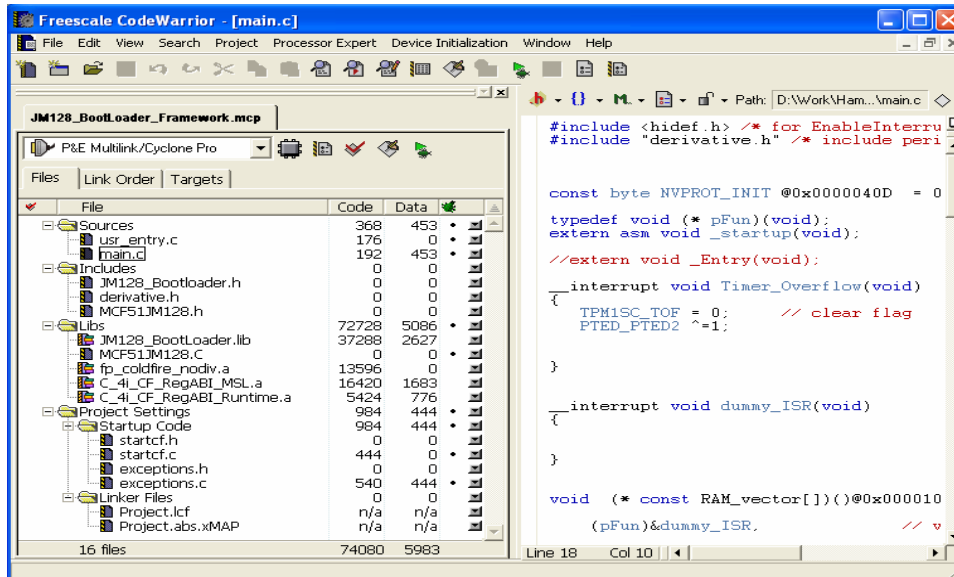


Figure 2. JM128 USB Bootloader Framework

## 2.1 Project Files Introduction

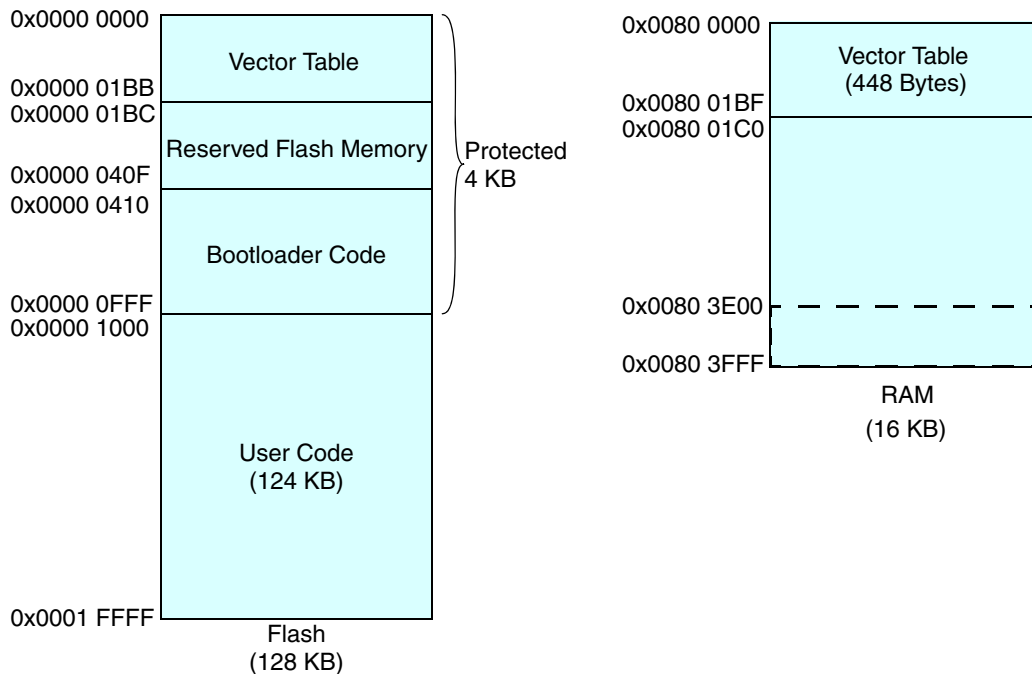
There are six files associated with the bootloader:

- **Main.c** — It is the main file for a project. The main() function is located in this file. Some codes associated with flash protection and vector redirection are also in this file.
- **Usr\_entry.c** — This file defines the entry function of the bootloader. You can modify it according to the system hardware configuration.
- **JM128\_Bootloader.lib** — This library file encapsulates most functions for the USB boot loader that involve the operation of USB and flash.
- **JM128\_Bootloader.h** — The USB configuration for the pullup resistor and regulator are defined in this file.
- **Exceptions.c** — This file is generated by CodeWarrior automatically; it is modified to make sure the PC (program counter) points to the correct entry point after the MCU resets.
- **Project.lcf** — This file is generated by CodeWarrior automatically for the linker. It is modified to place the USB bootloader code in memory space.

## 2.2 Memory Map

Figure 3 is the memory footprint for the MCF51JM128 USB bootloader.

## Bootloader Framework Overview



**Figure 3. Bootloader Memory Map**

The bootloader uses as few resources as possible to minimize impact on the application. Its code is optimized to occupy a small amount of flash and RAM space.

### 2.2.1 Flash Space

The MCF51JM128 USB bootloader code size is less than 3 KB. It is placed in the low 4 KB flash space (from 0x0000 0000 to 0x0000 0FFF) that is protected in case of accidental damage. The flash protection is achieved by this code:

```
const byte NVPROT_INIT @0x0000040D = 0xFB;
```

Refer to Section 4.4.2.4, “Flash Protection Register,” of the *MCF51JM128 Reference Manual* for more information about flash protection.

When upgrading the firmware, the vector table based at 0x0000 0000 to 0x0000 01BB cannot be modified because of flash protection. This means you cannot modify the interrupt service routine entry. The method to solve this problem is interrupt redirection. On the MCF51JM128, the vector table can only be redirected to the RAM (0x0080 0000), which will be discussed later.

In the vector table section of flash (located at 0x0000 0000), the first eight bytes are used for keeping the address of the initial stack pointer (SP) and program counter (PC) after reset. They are initialized by bootloader; the other bytes are reserved in this framework.

The user program section is in the range 0x0000 1000 to 0x0001 FFFF, a total of 124 KB.

## 2.2.2 RAM Space

When the USB bootloader is integrated in the application, the first 448 bytes in RAM (from 0x0080 0000 to 0x0080 01BF) are assigned to the relocated vector table.

There are about 130 bytes allocated for flash operation and 100 bytes for USB RAM. Your application can override these RAM bytes because they are valid only if the bootloader is working.

### 2.2.2.1 Redirected Vector Table

The first section of RAM is the vector table. The default vector table for MCF51JM128 (located at 0x0000 0000) is redirected to RAM for the purpose of protecting the bootloader code. It is your responsibility to redirect the vector table to RAM if the interrupt is used in an application. The bootloader framework provides one solution for automatically implementing the interrupt redirection.

The interrupt redirection is implemented by the assembly code given here. The value in the VBR register is modified to 0x0080 0000.

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

A new interrupt vector table is created and maintained. This table will be copied to the RAM-based vector table during the initialization process. The new vector table in the bootloader framework is declared by this code:

```
void (* const RAM_Vector[])()@0x00001000=
{
(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 Vtpmlch0
(pFun)&dummy_ISR,           // vector_72 Vtpmlch1
(pFun)&dummy_ISR,           // vector_73 Vtpmlch2
.....
}
```

If you want to add a new interrupt, you can replace the address of the dummy\_ISR with that of the new interrupt service routine (ISR) according to the vector number. In the above code, the RAM\_Vector is fixed at 0x0000 1000 (from 0x0000 1000 to 0x0000 11BB).

This code copies the above table (RAM\_Vector) to the vector table located in RAM:

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

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

### 2.2.2.2 User RAM

The user RAM that can be assigned to your application is in the range 0x0080 01C0 to 0x0080 3FFF.

The bootloader occupies 130 bytes in user RAM that are used for USB communication and flash operation. At the end of the user RAM section, there are more than 100 bytes assigned to USB operation (the buffer descriptor table and endpoint buffer). This section is designated as USB RAM. The USB RAM must be 512 bytes aligned according to the requirements of the MCF51JM128 USB module.

The RAM occupied by the bootloader is only valid if it is working, so the user code can reuse it in its application code. For example, if you set the USB BDT in your application to USB RAM (starting from 0x0080 3E00), the USB RAM can now be used for the bootloader or the user application.

## 2.3 Bootloader Flowchart

Figure 4 is the flowchart for the MCF51JM128 USB bootloader.

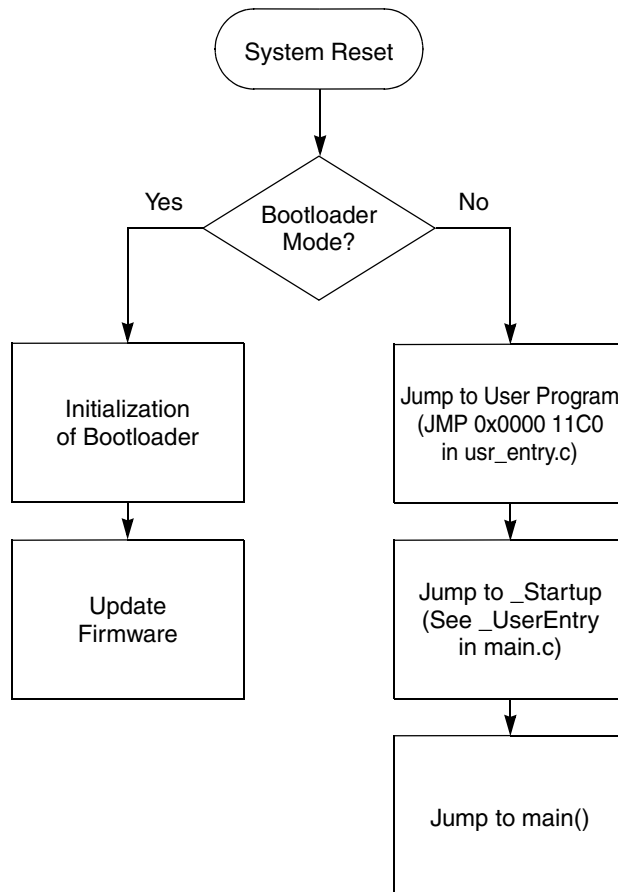


Figure 4. Bootloader Flowchart

The program jumps to the user entry (usr\_entry.c) after MCU reset. You can select the method to enter the bootloader. For example, the MCU enters boot mode if the button is held down (PTG0) when the MCU is reset — otherwise it jumps to the user program.

If you select to enter bootloader mode, the system does some initialization for the bootloader at first, such as initializing the USB module, configuring the MCG module to generate a 24 MHz bus clock, disabling the COP, etc. After that the system program jumps to the bootloader code (JM128\_bootloader.lib) and works with the PC GUI to upgrade the firmware.

The MCU jumps to the user application code if bootloader mode is not selected. To implement this, the MCU goes to address 0x0000 11C0 and executes the code; the code located at 0x000 011C0 is used to jump to the `_Startup` function. Therefore the program ends up at the `main()` function in `_Startup`.

## 3 Create a Project with the Bootloader

This section explains how to create a new MCF51JM128 project which supports the USB bootloader.

After the project is created, the project files are added to support the USB bootloader.

### 3.1 Create the Project

To create the project, you must execute all of the steps given here.

#### 3.1.1 Create a New Project

1. Launch the CodeWarrior IDE:  
Start → Programs → Freescale CodeWarrior → CW for Microcontrollers 6.1 → CodeWarrior IDE.
2. Create a new MCF51JM128 CodeWarrior project by clicking New Project (Figure 5).



Figure 5. CodeWarrior IDE

3. Select MCF51JM128 MCU in the New Project wizard (Figure 6).

## Create a Project with the Bootloader

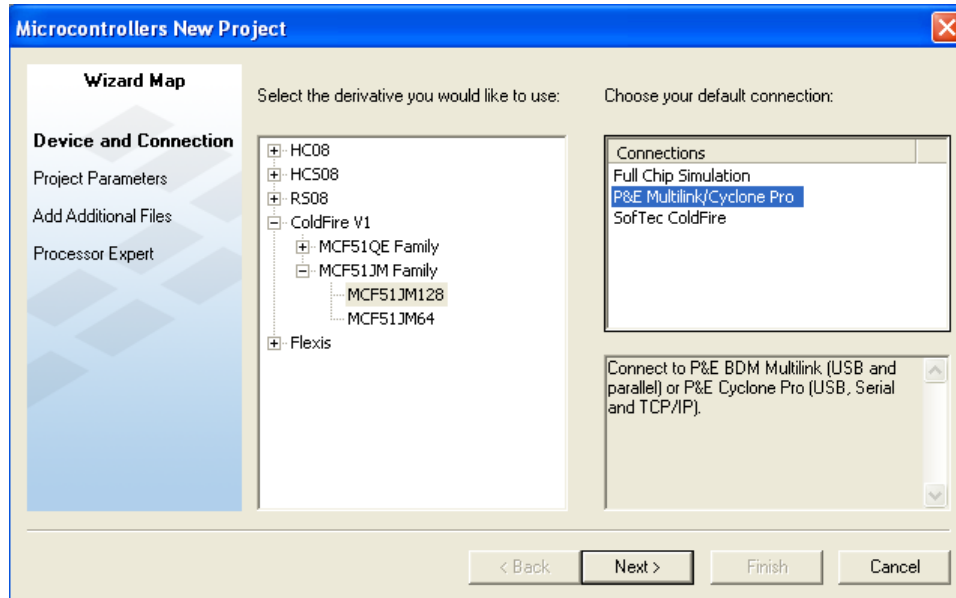


Figure 6. Create A New MCF51JM128 Project

4. With the default settings, the new created project looks like the project JM128\_USB\_Bootloader.mcp in Figure 7.

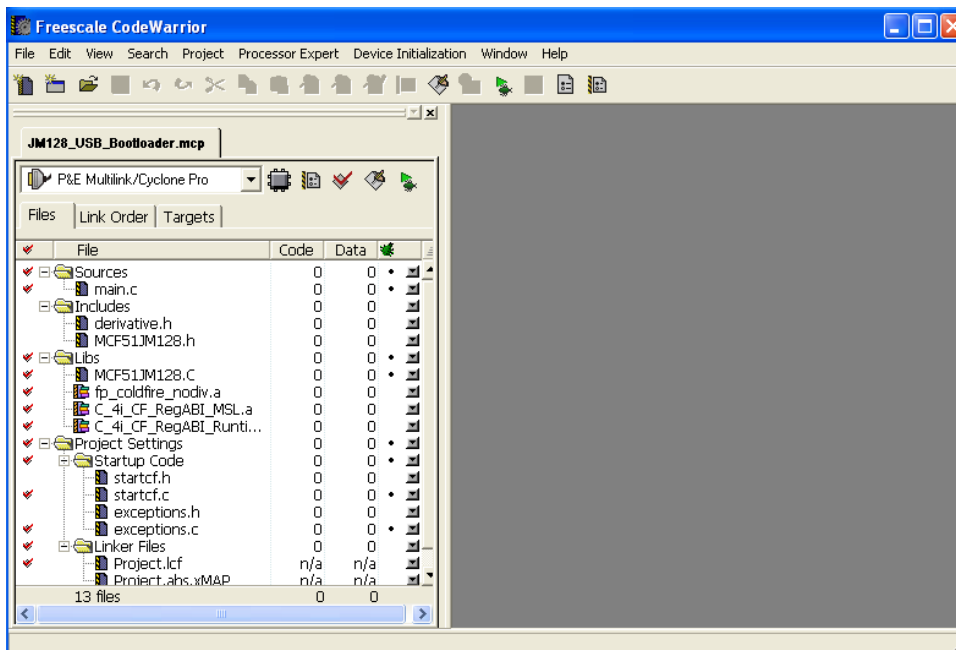


Figure 7. New CodeWarrior Project

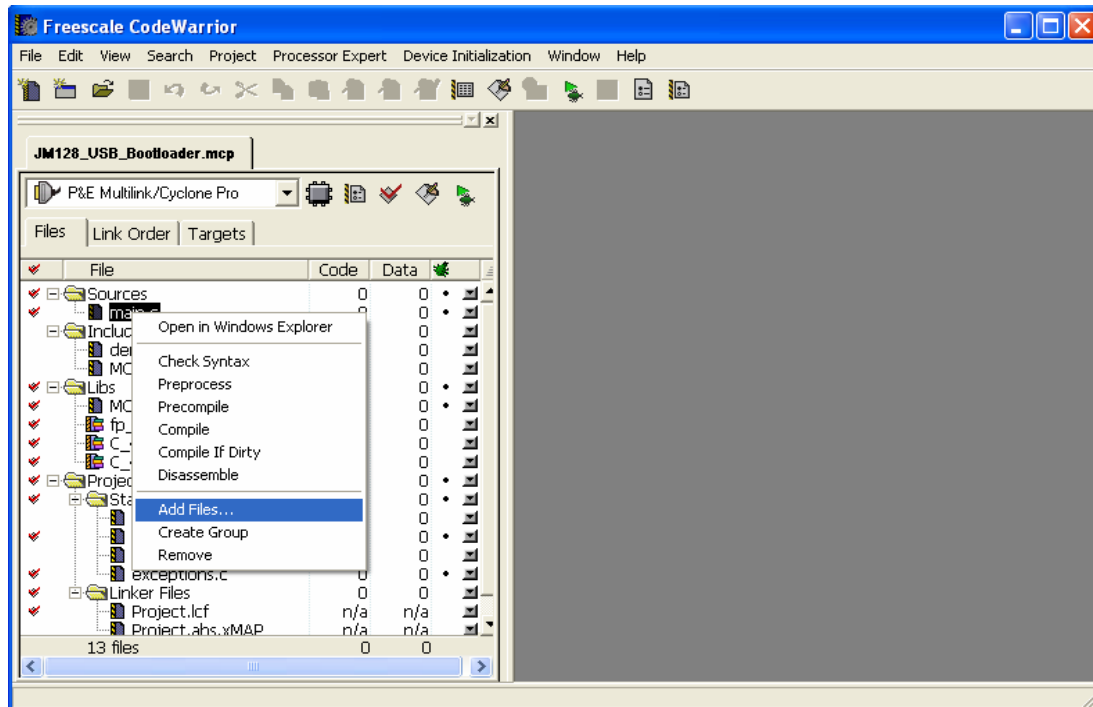
### 3.1.2 Add Files for Supporting the USB Bootloader

1. Copy the JM128\_Bootloader.lib, JM128\_Bootloader.h, exceptions.c, main.c, and usr\_entry.c files located in the directory ..\JM128\_Bootloader\_Framework\source to the directory of the new project (..\JM128\_USB\_Bootloader\source).

Select to override the loading of the main.c and exceptions.c files.

2. Copy the Project.lcf file located in the ..\JM128\_Bootloader\_Framework\prm directory to the ..\JM128\_USB\_Bootloader\prm directory and override the original file.
3. Add the usr\_entry.c, JM128\_Bootloader.h, and JM128\_Bootloader.lib files to the project.

Select the directory in CodeWarrior IDE and click the right mouse button. A pop-up menu is displayed. To add the file, select Add Files (see [Figure 8](#)).



**Figure 8. Add the File to the Project**

Add the usr\_entry.c file to the Source folder and JM128\_bootloader.lib to the Libs folder. The project file tree now looks like the tree structure in [Figure 9](#).

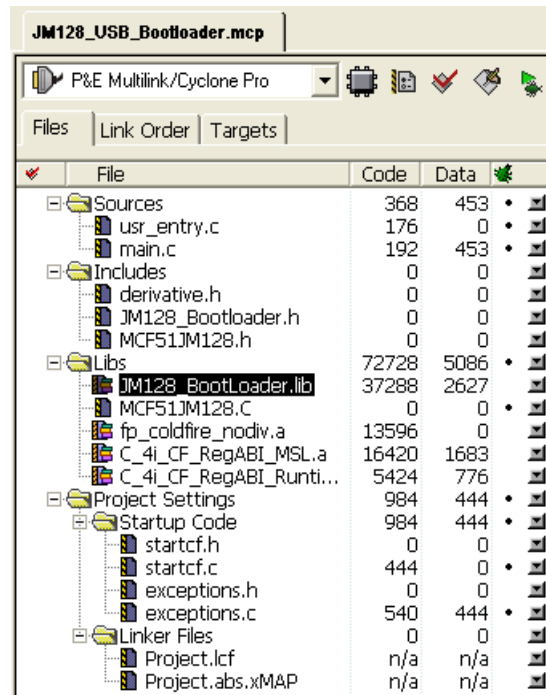


Figure 9. Project Files

### 3.1.3 Set the Program Entry Point

To support the bootloader, the system program must have the capability to enter bootloader mode or user mode after the MCU is reset. In our solution, the `_Entry` function included in the `user_entry.c` file is executed first after a reset.

The default setting of the CodeWarrior project executes the `_Startup` function after reset. You need to modify the settings discussed here to change the entry point.

1. Set the initial value of the program counter (located at `0x0000 0004`) with the address of the `_Entry` function by modifying the `exceptions.c` file. The bootloader framework includes this modification, and overrides the `exceptions.c` file.
2. Modify the ColdWarrior standard settings. Click Standard Setting (Figure 10) to open the dialog box in Figure 11.
3. Find the ColdFire Linker in Target Setting Panel and change `__Startup` to `__Entry` in the entry point text box.

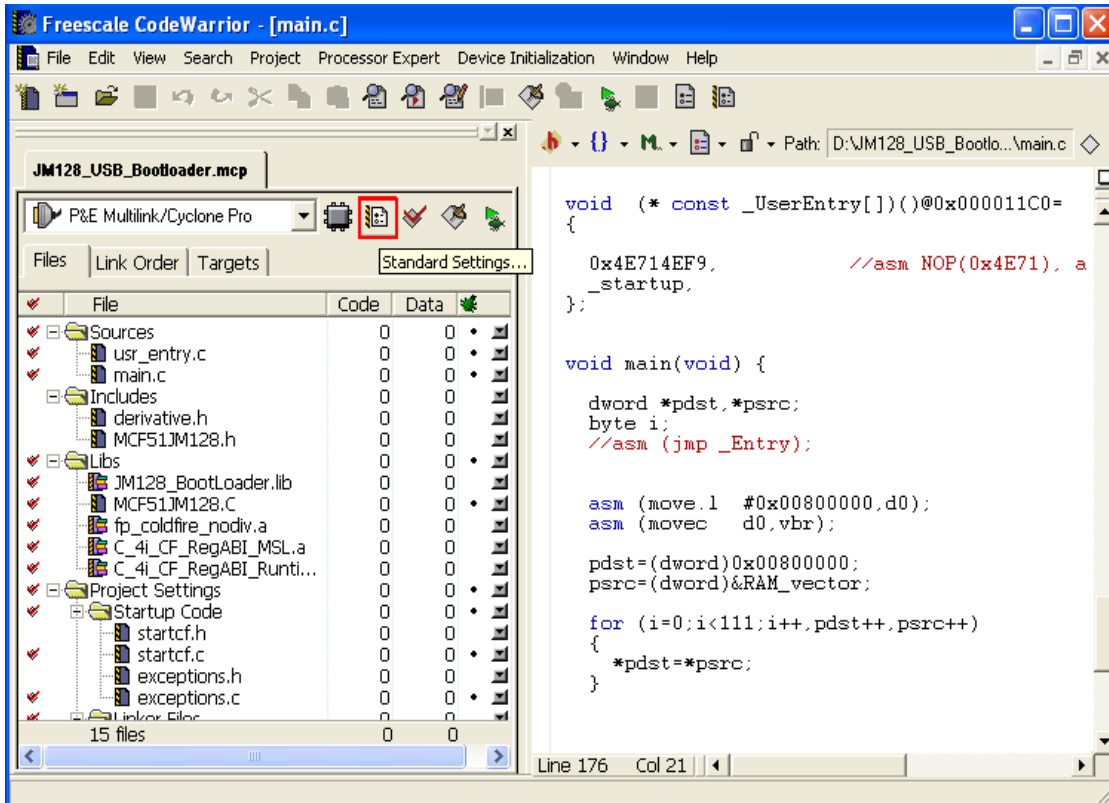


Figure 10. Open Standard Settings Window

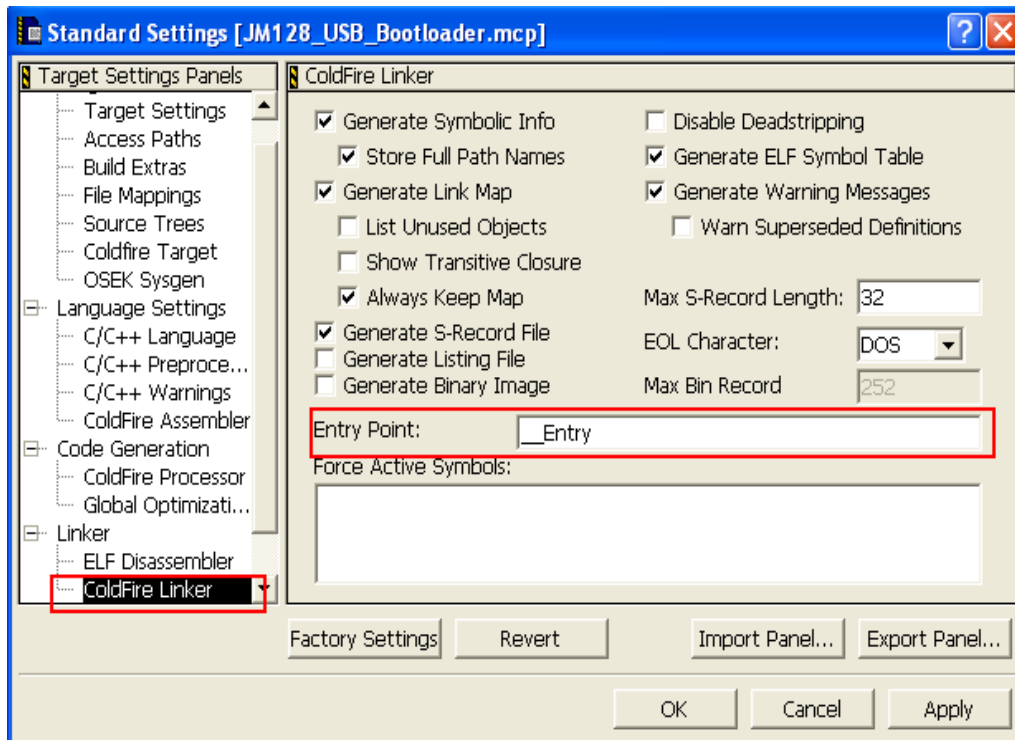


Figure 11. Set Entry Point

### 3.1.4 Change USB Configuration

The USB configuration is essentially the USB pullup resistor and regulator. MCF51JM128 can be configured to use either an internal or external pullup resistor and regulator. The definition of USB\_CONFIG in JM128\_Bootloader.h can be changed according to the hardware settings. The definition of the USB\_CONFIG macro is the same as the USBTRC0 register. Refer to Section 16.5.1.26, “USB Transceiver and Regulator Control Register 0,” in the *MCF51JM128 Reference Manual* for more information.

The project example in the framework is adapted to use the internal pullup resistor and regulator.

### 3.1.5 Add User Code

Add your application code based on the framework.

#### Example:

The code in this example uses a TPM timer to control the LED.

1. Add the initialization code.

```
PTEDD_PTEDD2 = 1;

TPM1MOD = 0x3E8;
TPM1SC_PS = 2;
TPM1SC_CLKSx = 0x01;
```

```
TPM1SC_TOIE = 1;
```

The above code initializes the GPIO port (PTED2 output) and TPM module (to work as a timer).

2. Add the TPM interrupt service routine.

```
__interrupt void Timer_Overflow(void)
{
    TPM1SC_TOF = 0;    // clear flag
    PTED_PTED2 ^=1;
}
```

3. Add the TPM interrupt service routine to the vector table (RAM\_vector). The default, dummy\_ISR, is replaced by Timer\_Overflow according to the vector number. The interrupt number for the TPM1 timer overflow of the MCF51JM128 is 77. You can determine the correct position by the annotation in each line.

```
(pFun)&Timer_Overflow,    // vector_77 Vtpmlovf
```

### 3.1.6 Build the Project and Download to MCU

The project has now been completed. You can compile, download, or debug this project. The bootloader is also downloaded into flash with the user application. The user code can be updated later, but the bootloader code is fixed and protected.

One LED on the demo board blinks if the firmware works properly. You can force the system to enter bootloader mode and update the firmware by using the PC tool. Refer to [Section 4, “Using the PC Tool,”](#) for more details on how to do this.

## 3.2 File Modifications

### 3.2.1 Exceptions.c

The exceptions.c file includes the default process for all exceptions. The SP (stack pointer) and PC (program counter) are initialized in this file. To support the bootloader, the PC value must be modified.

In the new exceptions.c file (bootloader framework), the PC value is changed to the address of the `_Entry` function.

```

/*
 * MCF51JM128 vector table
 * CF V1 has 110 vector + SP_INIT in the vector table (111 entries)
 */

__declspec(weak) vectorTableEntryType vector_0    @INITSP =
(vectorTableEntryType)&_SP_INIT;
__declspec(weak) vectorTableEntryType vector_1    @INITPC = (vectorTableEntryType)
&_Entry;
__declspec(weak) vectorTableEntryType vector_2    @Vaccerr = asm_exception_handler;
__declspec(weak) vectorTableEntryType vector_3    @Vadderr = asm_exception_handler;
.....

```

The `_Entry` function is defined in `usr_entry.c`.

### 3.2.2 usr\_entry.c

The `_Entry` function defined in the `usr_entry.c` file is modifiable. This enables you to change the method for entering bootloader mode, and to initialize the system for the bootloader according to the hardware configuration.

Here is the code of the `_Entry` function.

```

void _Entry(void)
{
    byte i;

    PTGDD_PTGDD0 = 0;           // PTG0 is input
    PTGPE_PTGPE0 = 1;         // internal pullup for PTG0

    if(PTGD_PTGD0)
    {
        asm (JMP 0x000011C0);    // jump to user entry
    }
    else
    {
        SOPT1 = 0x00;           // disable COP
        SOPT2 = 0x00;

        MCGC2 = 0x36;
        while(!(MCGSC & 0x02)){}; //wait for the OSC stable
        MCGC1 = 0x98;
        while((MCGSC & 0x1C) != 0x08){}; // external clock is selected
    }
}

```

## Create a Project with the Bootloader

```
        MCGC3 = 0x48;
        while ((MCGSC & 0x48) != 0x48){};          //wait for the PLL is locked
        MCGC1 = 0x18;
        while((MCGSC & 0x6C) != 0x6C){};

        Bootloader_Main();
    }
}
```

The MCU enters bootloader mode by checking the status of the PTG0 pin. If PTG0 is high, the MCU executes the user code — otherwise, it enters bootloader mode.

The MCU jumps to the user program by executing the instruction `JMP 0x0000 11C0`. The entry point of the user program is placed at that location. This is explained in more detail in [Section 3.2.6, “Main.c.”](#)

After the MCU enters bootloader mode, the `_Entry` function does some initialization and then jumps to `Bootloader_Main`. The COP and interrupt are disabled because they may impact the flash operation (erase or program). The MCG must be configured to work in PEE mode for generating a 24 MHz bus clock.

### 3.2.3 Project.lcf

`Project.lcf` is the linker command file. It defines certain rules for the linker to allocate flash and RAM space.

The memory is divided into four sections:

- Bootcode from 0x0000 0410 to 0x0000 0FFF
- Code from 0x0000 1200 to 0x0001 FFFF
- VectorRAM from 0x0080 0000 to 0x0080 01BF
- UserRAM from 0x0080 01C0 to 0x0080 3FFF

```
MEMORY{
    bootcode(RX):ORIGIN = 0x00000410,LENGTH = 0x00000BF0
    code(RX):ORIGIN = 0x00001200,LENGTH = 0x0001EE00
    vectorram(RWX):ORIGIN = 0x00800000,LENGTH = 0x000001C0
    userram(RWX):ORIGIN = 0x008001C0,LENGTH = 0x00003E40
}
```

The code shown next enables the linker to place the code that is in `JM128_Bootloader.lib` and in `usr_entry.c` into the bootcode section.

```
.bootcode:
{
    __Boot_START = .;
    JM128_Bootloader.lib (.text)
    JM128_Bootloader.lib (.rodata)
    usr_entry.c (.text)
    usr_entry.c (.rodata)
    . = ALIGN (0x4);
    __Boot_END = .;
} > bootcode
```

### 3.2.4 JM128\_Bootloader.h

The USB configuration is defined in this file.

```
#define USB_CONFIG 0x44
```

In this definition (USB\_CONFIG), the USB is configured to use an internal pullup resistor and regulator. [Table 1](#) lists the different configurations for USB.

**Table 1. USB Configurations**

Configuration Value	Pullup Resistor	Regulator
0x00	External	External
0x04	External	Internal
0x40	Internal	External
0x44	Internal	Internal

### 3.2.5 JM128\_Bootloader.lib

This library includes all USB and flash operation functions for the USB bootloader. You can add it to your project without any modifications.

### 3.2.6 Main.c

This line of code is used to protect the flash, from address 0x0000 0000 to 0x0000 0FFF.

```
const byte NVPROT_INIT @0x0000040D = 0xFB
```

You can modify and increase the protected flash area, but cannot decrease or delete it.

The vector table is redirected to RAM. Because RAM cannot retain the vector table when power is lost, the RAM\_vector array is defined to retain the entry address for all exceptions. This array is copied to RAM (0x0080 0000) in the initialization routine of the user code. Refer to Section 6.2.6, “Vector Base Register,” of the *MCF51JM128 Reference Manual* for more information about interrupt redirection.

The entry is set by this function:

```
void (* const _UserEntry[])(@0x000011C0=
{
    0x4E714EF9,          //asm NOP(0x4E71), asm JMP(0x4EF9)
    _startup,
};
```

This function is fixed to 0x0000 11C0 in flash. Executing this function makes the program jump to the \_Startup function. At the end of the \_Startup function, the program jumps to the main function.

At the beginning of the main function, the program sets the VBR register to redirect and copy the vector table to RAM. After that, you can add your application code.

## 4 Using the PC Tool

### 4.1 Installing the Bootloader GUI

The .NET framework 2.0 must be installed on your computer before you can install the bootloader GUI.

To install the bootloader GUI:

1. Double-click the setup.exe program in this application note's associated software.
2. The Freescale MCF51JM128 WinUSB Driver 0.1 setup license agreement dialog box appears. Select "I agree" to accept the agreement.
3. Select the setup installation folder and click install. The USB driver for the bootloader is now installed.

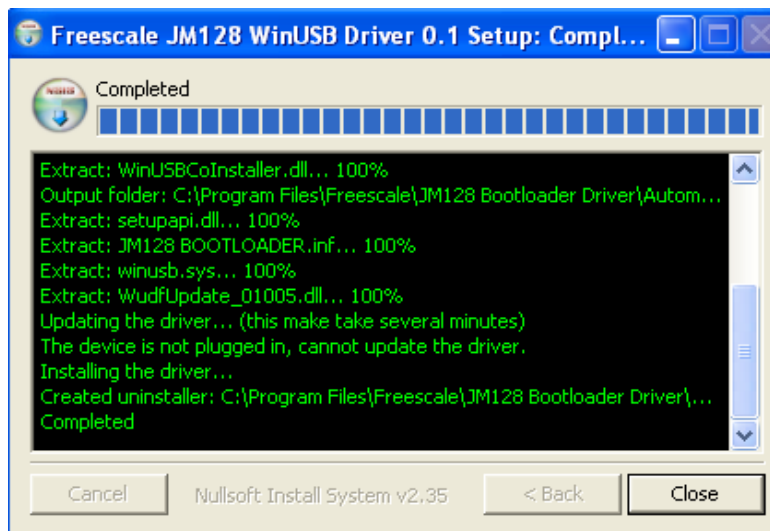
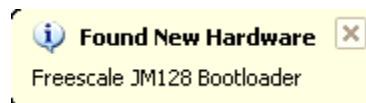


Figure 12. Installing USB Drivers for Bootloader GUI

### 4.2 Identify the USB Bootloader Device

If you power on or reset the MCU with the button (PTG0) held down and the mini-USB port connected to the PC, the MCU enters bootloader mode. The PC will detect the new hardware and prompt you to find the Freescale JM128 bootloader.



It will also prompt you to install the USB driver for the device. Select to install the software automatically.



Figure 13. Install the Software Automatically

The system will then find the bootloader driver and install by itself.

## 4.3 Update the Firmware via the Bootloader

### 4.3.1 Open the Bootloader GUI

After the driver has been successfully installed, launch the JM128 bootloader GUI.



## Porting to MCF51JM64/32 MCU

From the GUI, select the S19 file to upload it to the MCU. Execute the bootloader commands and check the command execution status.

Commands:

Erase: Erase the MCF51JM128 flash (0x000 01000 – 0x0000 1FFF).

Program: Download the selected S19 file to the MCF51JM128 flash.

Verify: Compare the S19 file with the code that has been downloaded to flash.

Auto: Execute the erase, program, and verify commands step by step.

Status Window:

Display the operating status and results.

### NOTE

The USB icon located in the bottom right corner of the GUI is green when the USB bootloader has been identified by the PC.

### 4.3.2 Select the S19 File

1. Click  to select the S19 file.



Figure 14. Selecting the S19 File

### NOTE

The S19 file must be generated from the CodeWarrior project which supports bootloader (include the bootloader frame).

### 4.3.3 Download the S19 File to Flash

Click execute to download the S19 file. The GUI will then execute the erase, program, and verify commands automatically, one by one.

You can also execute these commands individually by clicking an appropriate button from the GUI.

After the S19 file has successfully downloaded to the flash, reset the MCU to run the user program. If the update fails, select to erase the flash and reprogram the flash. You can also select the other S19 file for update. The bootloader can work properly even if the MCU is reset because the system can enter bootloader again.

## 5 Porting to MCF51JM64/32 MCU

Although the USB bootloader in this application note is designed for the MCF51JM128, it is very easy to port to other members of the JM family that have the V1 core (MCF51JM64/32).

All members of the JM family MCU with the ColdFire V1 core have the same RAM size and the same register definitions, so the code of the bootloader for the MCF51JM128 can work without modification on the related devices. The only difference in the bootloader is related to flash size. The length in the code section can be modified if it places the code out of the range of the flash space.

For example, if the bootloader of MCF51JM128 is ported to MCF51JM64, you can replace the MCF51JM128.c and MCF51JM128.h files with MCF51JM64.c and MCF51JM64.h, and change the code length to 0x0000 EE00 in the project.lcf file (in the second line of the code shown here).

```
MEMORY {
    bootcode(RX):ORIGIN = 0x00000410, LENGTH = 0x00000BE0
    code(RX):ORIGIN = 0x00001200, LENGTH = 0x0000EE00
    vectorram(RWX):ORIGIN = 0x00800000, LENGTH = 0x000001C0
    userram(RWX):ORIGIN = 0x008001C0, LENGTH = 0x00003E40
}
```

## 6 Summary

In this application note, the USB bootloader for MCF51JM128 is introduced. These topics have been discussed:

- The bootloader framework
- Creating a new project with the bootloader supported
- Installing the bootloader GUI and driver
- Updating the MCF51JM128 firmware with the bootloader

With the bootloader framework introduced above, you can easily create a project with the USB bootloader. The developer must copy and add some files to the project, and do a few modifications that make the bootloader easy to integrate. The friendly GUI for the bootloader can be reused without modification. The bootloader is flexible because it can be customized based on different hardware configurations in different applications.

## Appendix A Bootloader Framework Example Code

The CodeWarrior project with the bootloader framework is attached to this application note. All code referred in this document comes from it. You can create your own application based on this framework.

The code is developed under CodeWarrior for HC(S)08 V6.1 with the MCF51JM128 service pack installed. It can be compiled and downloaded to the MCF51JM128 demo board directly.

## Appendix B Bootloader PC Software

The software of the USB bootloader is provided to you for reference. It includes the PC driver and GUI.

Installing the GUI and PC driver is discussed in [Section 4, “Using the PC Tool,”](#) of this application note. With the code in [Appendix A, “Bootloader Framework Example Code,”](#) the bootloader can work like the description in [Section 4.3, “Update the Firmware via the Bootloader.”](#)

## How to Reach Us:

### Home Page:

[www.freescale.com](http://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](http://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](http://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](mailto: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](mailto:support.asia@freescale.com)

### For Literature Requests Only:

Freescale Semiconductor Literature Distribution Center  
P.O. Box 5405  
Denver, Colorado 80217  
1-800-441-2447 or 303-675-2140  
Fax: 303-675-2150  
[LDCForFreescaleSemiconductor@hibbertgroup.com](mailto:LDCForFreescaleSemiconductor@hibbertgroup.com)

Document Number: AN3748  
Rev. 0  
08/2008

Information in this document is provided solely to enable system and software implementers to use Freescale Semiconductor products. There are no express or implied copyright licenses granted hereunder to design or fabricate any integrated circuits or integrated circuits based on the information in this document.

Freescale Semiconductor reserves the right to make changes without further notice to any products herein. Freescale Semiconductor makes no warranty, representation or guarantee regarding the suitability of its products for any particular purpose, nor does Freescale Semiconductor assume any liability arising out of the application or use of any product or circuit, and specifically disclaims any and all liability, including without limitation consequential or incidental damages. "Typical" parameters that may be provided in Freescale Semiconductor data sheets and/or specifications can and do vary in different applications and actual performance may vary over time. All operating parameters, including "Typicals", must be validated for each customer application by customer's technical experts. Freescale Semiconductor does not convey any license under its patent rights nor the rights of others. Freescale Semiconductor products are not designed, intended, or authorized for use as components in systems intended for surgical implant into the body, or other applications intended to support or sustain life, or for any other application in which the failure of the Freescale Semiconductor product could create a situation where personal injury or death may occur. Should Buyer purchase or use Freescale Semiconductor products for any such unintended or unauthorized application, Buyer shall indemnify and hold Freescale Semiconductor and its officers, employees, subsidiaries, affiliates, and distributors harmless against all claims, costs, damages, and expenses, and reasonable attorney fees arising out of, directly or indirectly, any claim of personal injury or death associated with such unintended or unauthorized use, even if such claim alleges that Freescale Semiconductor was negligent regarding the design or manufacture of the part.

RoHS-compliant and/or Pb-free versions of Freescale products have the functionality and electrical characteristics as their non-RoHS-compliant and/or non-Pb-free counterparts. For further information, see <http://www.freescale.com> or contact your Freescale sales representative.

For information on Freescale's Environmental Products program, go to <http://www.freescale.com/epp>.

Freescale™ and the Freescale logo are trademarks of Freescale Semiconductor, Inc. All other product or service names are the property of their respective owners.  
© Freescale Semiconductor, Inc. 2008. All rights reserved.

