**Freescale Semiconductor**
应用说明

Document Number: AN4367
Rev 0, 09/2011

# 面向 MCU 的以太网引导加载程序

**作者:**

**Alejandro Lozano, Alí Piña**
**Mexico**
**Guadalajara**

## 1 简介

本文档介绍如何使用飞思卡尔提供的以太网引导加载程序。Kinetis、MCF52259 和 MCF51CN128 等多种平台都支持此以太网引导加载程序。

本应用笔记的介绍重点为采用 ARM Cortex-M4 内核的 Kinetis 系列。软件和主要测试都是在 K60N512 上执行的。

本应用说明将介绍如何修改 CW 和 IAR IDE 分别对应的.lcf 和.icf 链接文件，以便为您的应用添加以太网引导加载程序支持。

## 2 概述

引导加载程序是一个较小的软件，可让用户在闪存中下载/更新代码或数据。

MCU 与主机之间的通信层（包含新的应用二进制文件）根据用例的不同而有所不同。

此通信接口可能是 SPI I2C、以太网和 USB 等。

本文使用以太网通信接口。

提供的引导加载程序将作为 TFTP 客户端运行，另外，还将使用 DHCP 功能。

内容

1 简介 ..... 1
2 概述 ..... 1
2.1 引导加载程序架构 ..... 1
2.2 引导加载程序工作流程 ..... 2
3 开发支持以太网引导加载程序的应用 ..... 3
3.1 在 IAR 中开发支持以太网引导加载程序的裸板应用 ..... 3
3.2 在 CodeWarrior for MCU 10.1 中开发支持以太网引导加载程序的裸板应用 ..... 6
3.3 在 CodeWarrior for MCU 10.1 中开发支持以太网引导加载程序的 MQX 应用 ..... 10
4 运行以太网引导加载程序 ..... 12
5 结语 ..... 17
6 附录 A：IAR 链接器配置文件(.icf) ..... 18
7 附录 B：CodeWarrior 裸板工程中的链接器命令文件(.lcf) ..... 19
8 附录 C：CodeWarrior MQX 工程中的链接器命令文件(.lcf) ..... 20

© 2011 Freescale Semiconductor, Inc.

## 2.1 引导加载程序架构

引导加载程序放置在片上闪存的前 48 KB 区域中，但是，引导加载程序的实际占用的空间可能更小。可以将用户应用放置在引导加载程序闪存区域上方的闪存区域中。闪存的最后 4 KB 区域必须保留给引导加载程序的参数存储使用，该存储由片上闪存的擦除页面大小定义。

下图显示了存储器映射和引导加载程序存储器使用明细。引导加载程序代码位于闪存的第一个地址区域，即 0x0000_0000 至 0x0000_BFFF。引导加载程序参数位于闪存最后一个逻辑块的第一个地址区域，即 0x0007_F000 至 0x0008_0000（Kinetis MK60N512）。

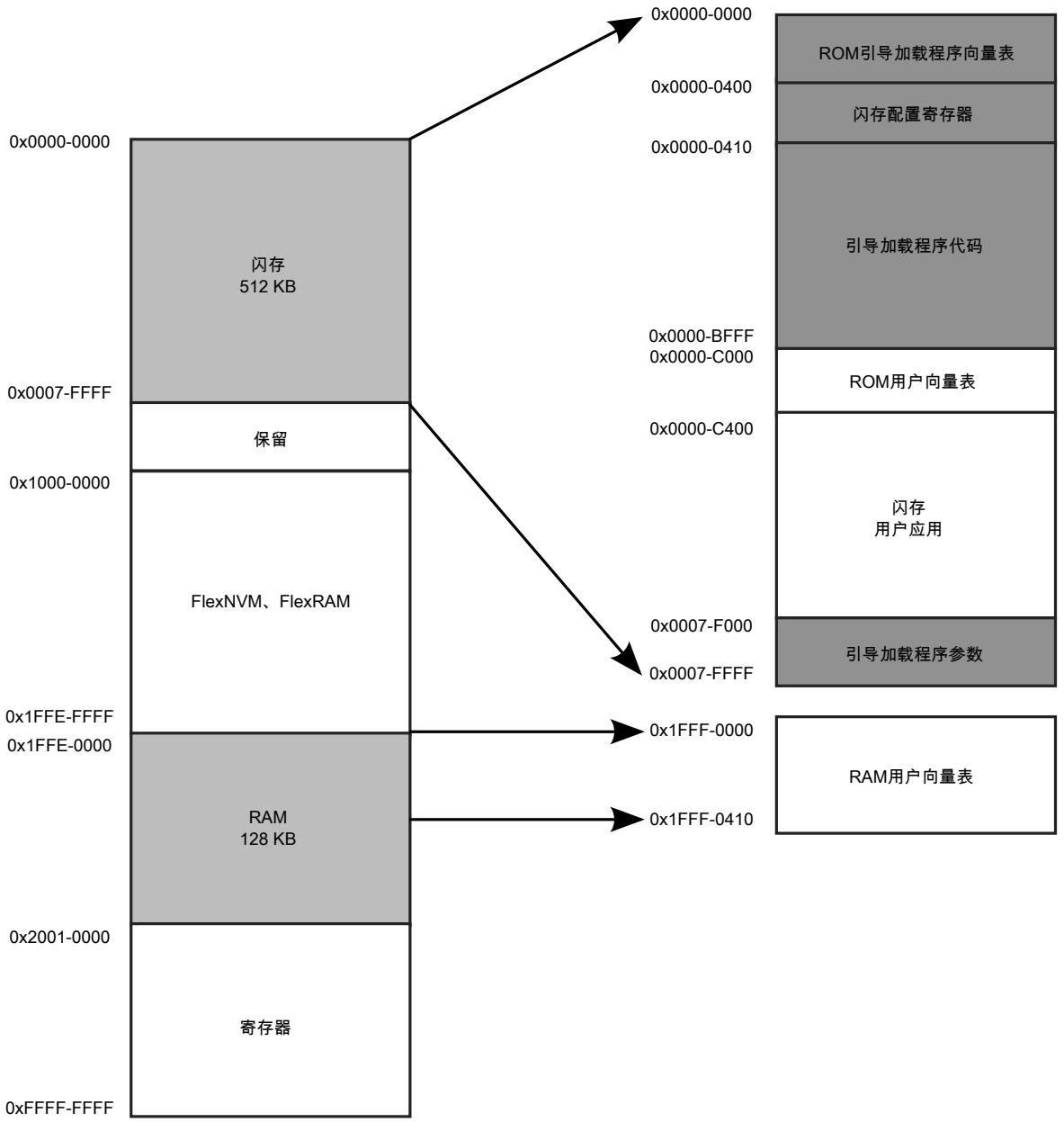



**图 1. 存储器映射**

## 2.2 引导加载程序工作流程

下图显示了引导加载程序工作流程。在执行上电复位后，将读取引导值。此引导值用于确定应用是要在复位后执行“运行”、执行设定脚本“script”还是引导加载程序进入“stop”模式。

默认的引导值为 stop。在此模式下，上电复位后，引导加载程序将立即运行。您可以修改引导加载程序的某些参数(要下载的映像等)；例如，您可以设定一个脚本，在复位后，只要设置了“script”模式，便可以运行该脚本。

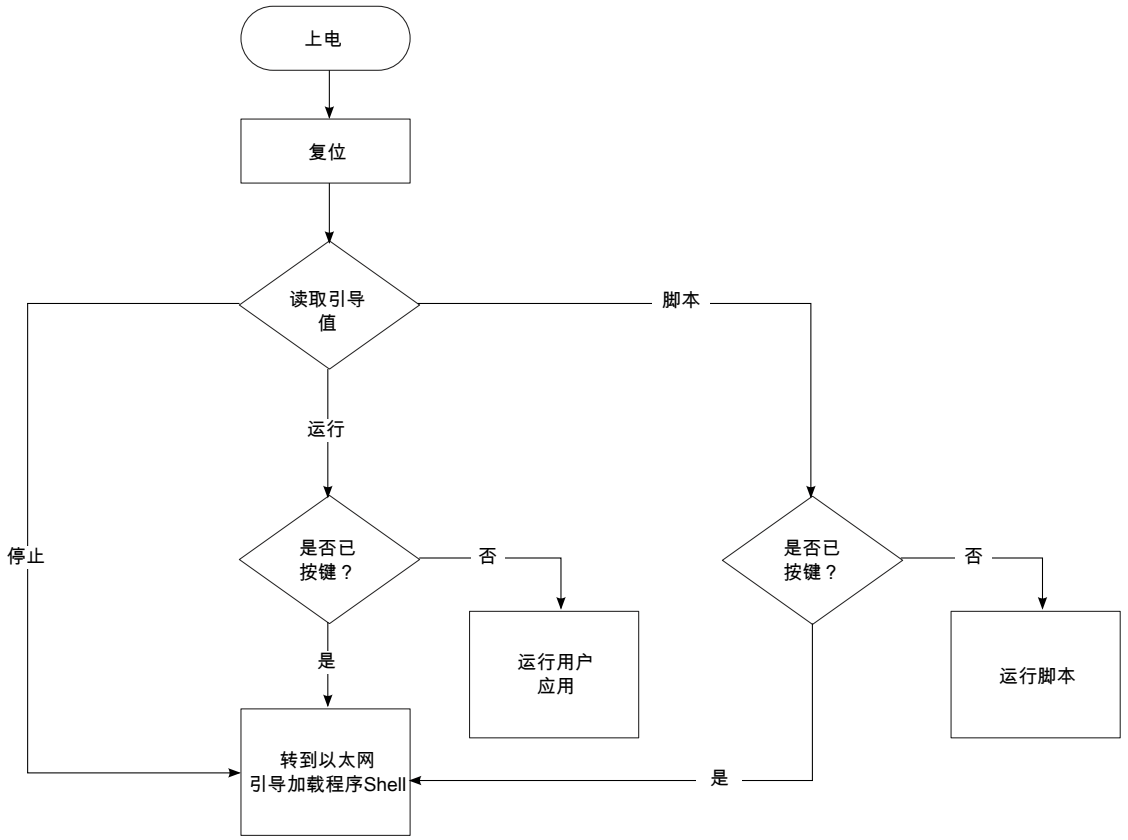



**图 2. 以太网引导加载程序工作流程**

在“script”模式下，您可以设定类似于下面的命令：‘**dhcp; erase all; tftp; set boot go; save; go**’

“运行以太网引导加载程序”中简要说明了此脚本。可以指定在复位后，如果 5 秒内按某个按键是否要执行该脚本。

如果在 5 秒内按任意键，“go”模式会强制 MCU 进入用户应用。

# 3 开发支持以太网引导加载程序的应用

本节详细介绍如何修改应用链接器命令文件（对于 CW，为.lcf；对于 IAR，为.icf），以便在引导加载程序所在的闪存区域中保留空间。应用代码必须放置在闪存区域的另一个段中。正因如此，需要修改 CW 和 IAR 分别对应的 LCF 和 ICF 文件。

## 3.1 在 IAR 中开发支持以太网引导加载程序的裸板应用

此时，您可以在 IAR 中使用您自己的工程，或者使用在 http://www.freescale.com 中搜索 KINETIS512_SC 得到的的示例工程。

本例将使用一个“Hello World”工程，可以在以下路径中找到该工程：…\kinetis-sc\ build\iar\hello_world。以下步骤说明如何修改应用链接器命令文件。

1. 打开 hello_world.eww 工作区。
2. 将目标更改为 FLASH_512KB_PFLASH，然后打开 512KB_Pflash.icf。

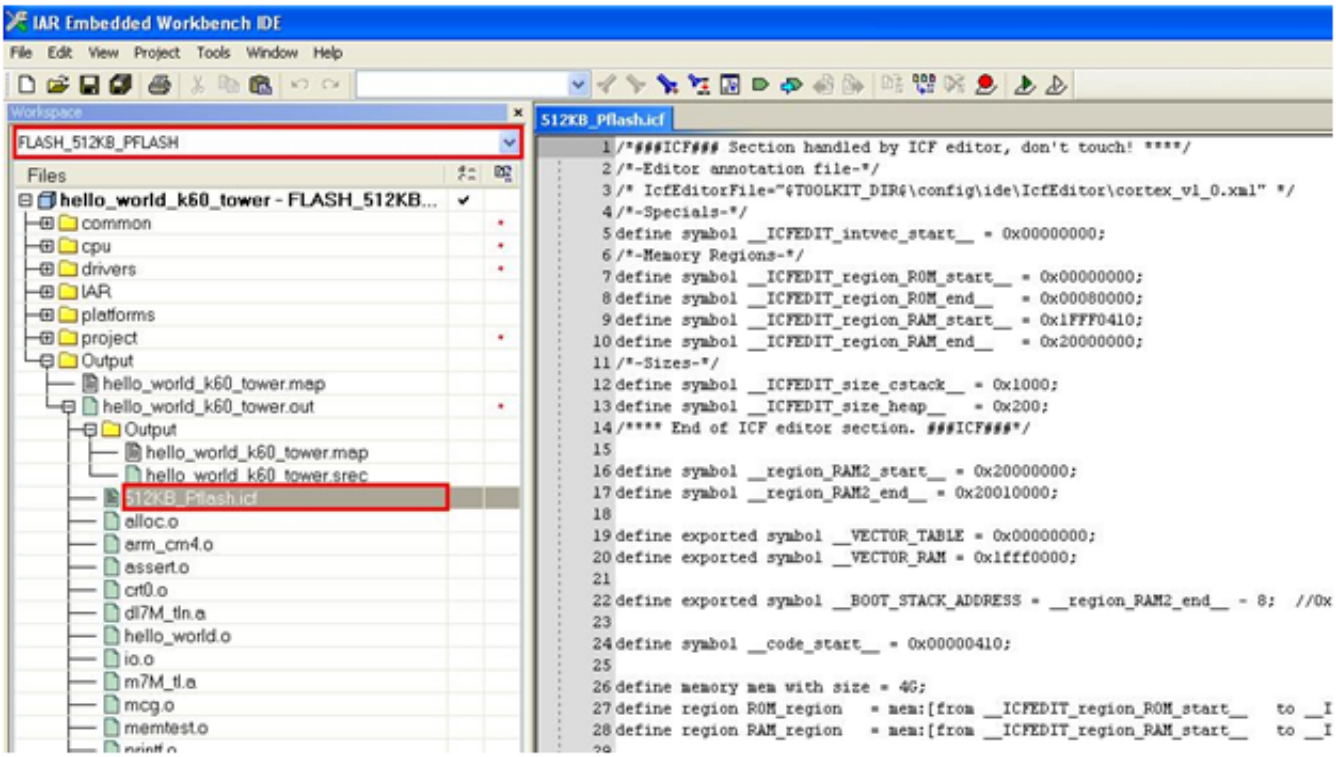



**图 3. 不包含通知的链接器命令(.icf)文件**

3. 修改.icf 链接文件。
   a. 将应用闪存中的起始地址从 0x0000_0000 修改为 0x0000_C000，包括 ROM 向量表和代码开头部分。

```
define symbol __ICFEDIT_intvec_start__ = 0x0000C000;

define symbol __ICFEDIT_region_ROM_start__ = 0x0000C000;
define symbol __ICFEDIT_region_ROM_end__   = 0x00080000;

define exported symbol __VECTOR_TABLE = 0x0000C000;
define exported symbol __VECTOR_RAM = 0x1fff0000;

define symbol __code_start__ = 0x0000C410;
```

   b. 通过删除 RAM2 段并将 RAM 段增大到 0x2001_0000，将拆分的 RAM 段转换为仅有一个 RAM 段。

```
define symbol __ICFEDIT_region_RAM_start__ = 0x1FFF0410;
define symbol __ICFEDIT_region_RAM_end__   = 0x20010000;

//define symbol __region_RAM2_start__ = 0x20000000;
//define symbol __region_RAM2_end__ = 0x20010000;
```

   c. 为引导加载程序参数创建闪存段。 仅当应用稍后要更改引导加载程序参数时，才需要此段。首先，您需要定义该段的起始地址和结束地址。因此，需要定义两个符号用于表示这些值:

```
define symbol FappParams_start__= __ICFEDIT_region_ROM_end__ - (2*0x400); //FAPP
define symbol FappParams_end__   = __ICFEDIT_region_ROM_end__-1; //FAPP
```

然后，创建要在其中放置应用的 ROM 区。

```
define region ROM_region    = mem:[from __ICFEDIT_region_ROM_start__    to
__ICFEDIT_region_ROM_end__-1] - mem:[from FappParams_start__  to FappParams_end__];

define region RAM_region    = mem:[from __ICFEDIT_region_RAM_start__    to
__ICFEDIT_region_RAM_end__-1]; //| mem:[from __region_RAM2_start__ to
__region_RAM2_end__-1];

place at address mem:FappParams_start__ { section fapp_params }; //APP
```

附录 A 中显示了修改后的整个.icf 文件。

4. 编译并链接工程。阅读.map 文件，以检查应用代码是否位于所需的闪存地址中。

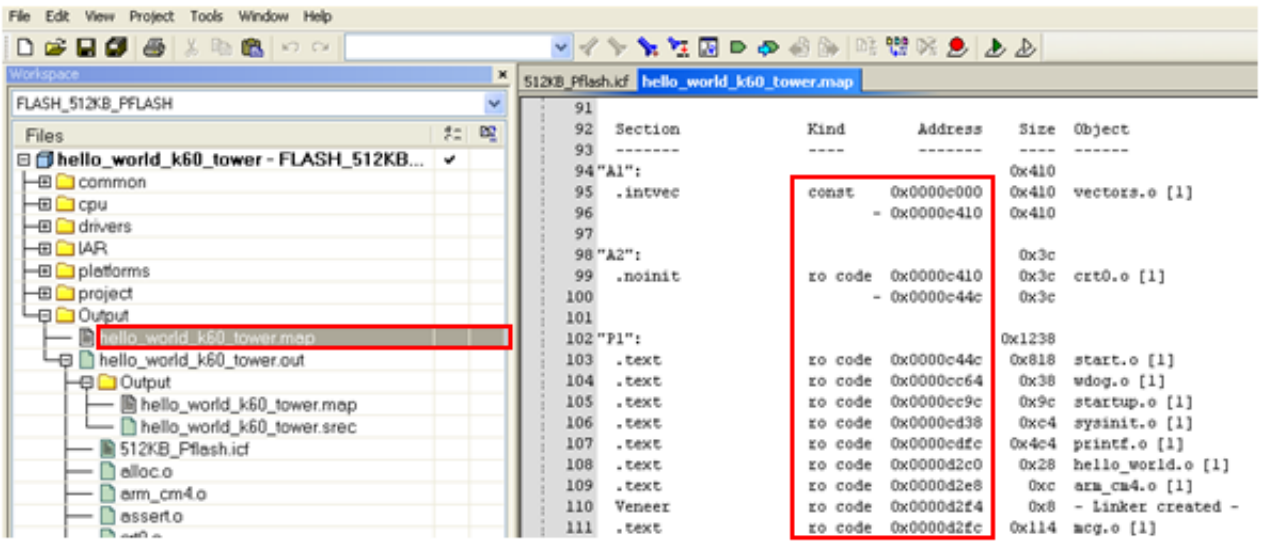



**图 4. 已修改链接器命令文件后面的存储器映射输出文件。**

现在，可以通过以太网引导加载程序将生成的.srec 文件下载到您的 MCU，并且该文件已出现在工程的输出文件夹中。您可以按下图所示查找该文件。第 4 节的操作需要使用此文件。

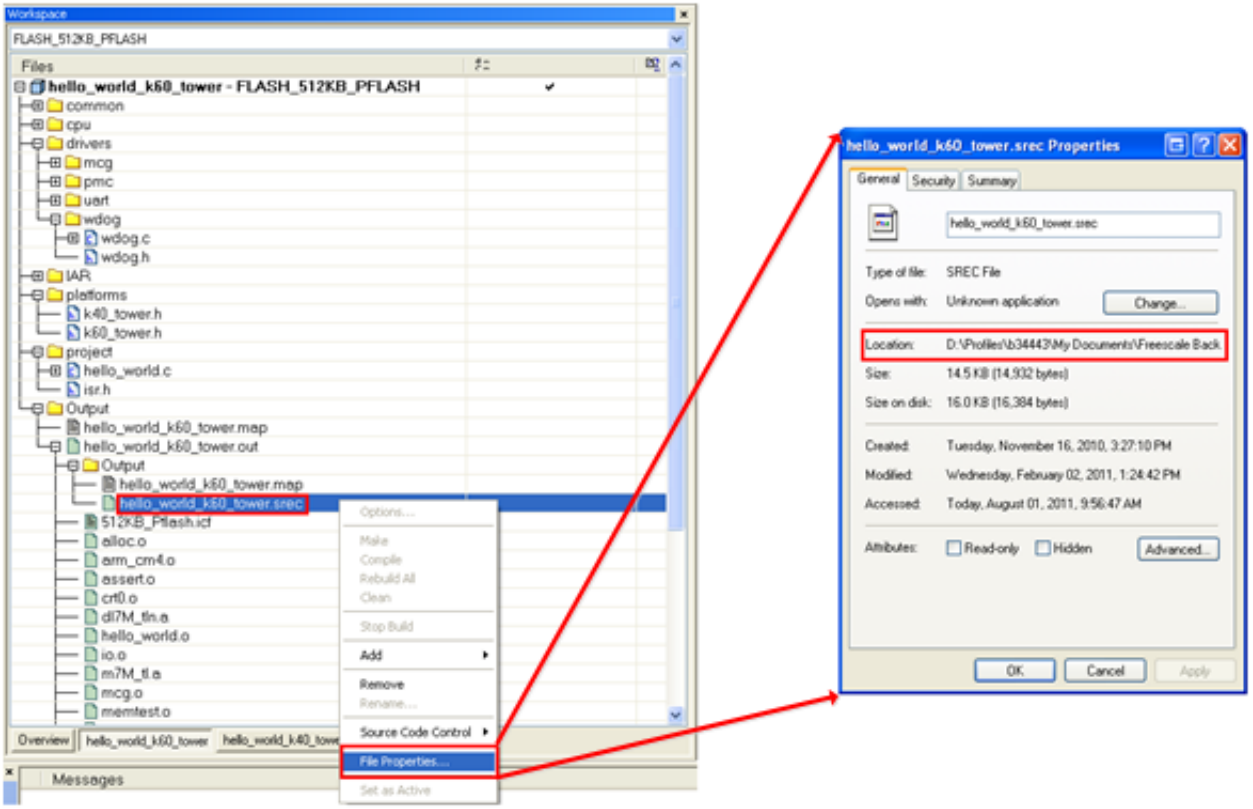


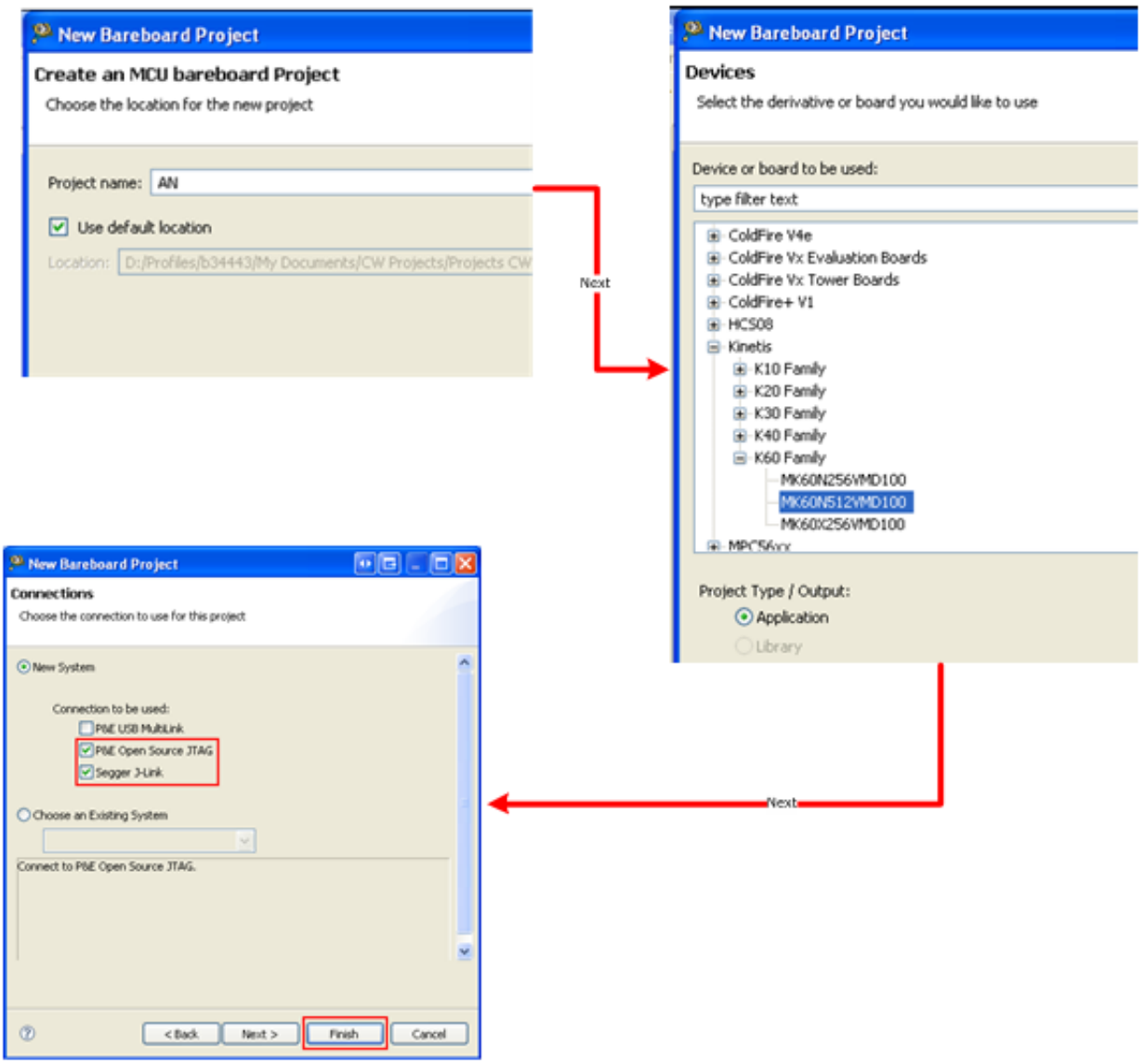

**图 5. s-rec 文件的位置。**

## 3.2 在 CodeWarrior for MCU 10.1 中开发支持以太网引导加载程序的裸板应用

要创建裸板应用，您可以在 CodeWarrior (CW) 10.1 中创建新工程，也可以使用现有工程。为了演示本应用说明，我们创建了一个新工程。

1. 通过“File”>“New”>“Bareboard Project”创建一个固定裸板工程，然后按照下面所述的过程操作。

**图 6. 在 CodeWarrior 中创建裸板工程。**




2. 将目标更改为 MK60N512VMD100_INTERNAL_FLASH，然后打开 MK60N512VMD100_flash.lcf

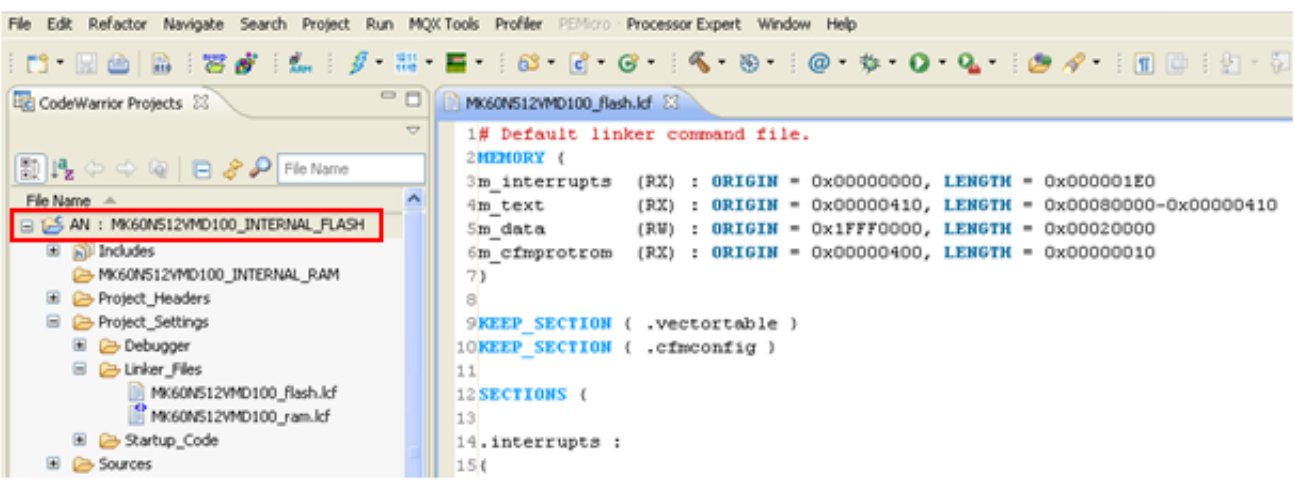



**图 7. CW 中的内部闪存目标。**

3. 修改.lcf 文件

a. 将应用闪存中的起始地址从 0x0000 修改为 0xC000，包括 ROM 向量表。

```
# 48KB (Reserved for Ethernet Bootloader)
fnet_bootloader (RX)  : ORIGIN = 0x00000000, LENGTH = 0x0000C000
m_interrupts  (RX) : ORIGIN = 0x0000C000, LENGTH = 0x000001E0
m_text        (RX) : ORIGIN = 0x0000C400, LENGTH = 0x0007F000-0x0000C400
```

b. 删除 m_cfmprtrom 区，因为您不需要使用此区。由于此段是为闪存保护寄存器保留的，因此，引导加载程序将处理此段。

```
# m_cfmprotrom  (RX) : ORIGIN = 0x00000400, LENGTH = 0x00000010

# .cfmprotect :
# {
# *(.cfmconfig)
# . = ALIGN (0x4);
# } > m_cfmprotrom
```

c. 为引导加载程序参数创建闪存段。

```
# 4Kbytes (Last logical-block reserved for parameters)
fnet_params   (RW)  : ORIGIN = 0x0007F000,LENGTH = 0x00001000
```

附录 B 中显示了修改后的整个.lcf 文件。

4. 编译工程。默认情况下，CodeWarrior 不生成 Link Map 文件；您需要按下图所示启用此功能。

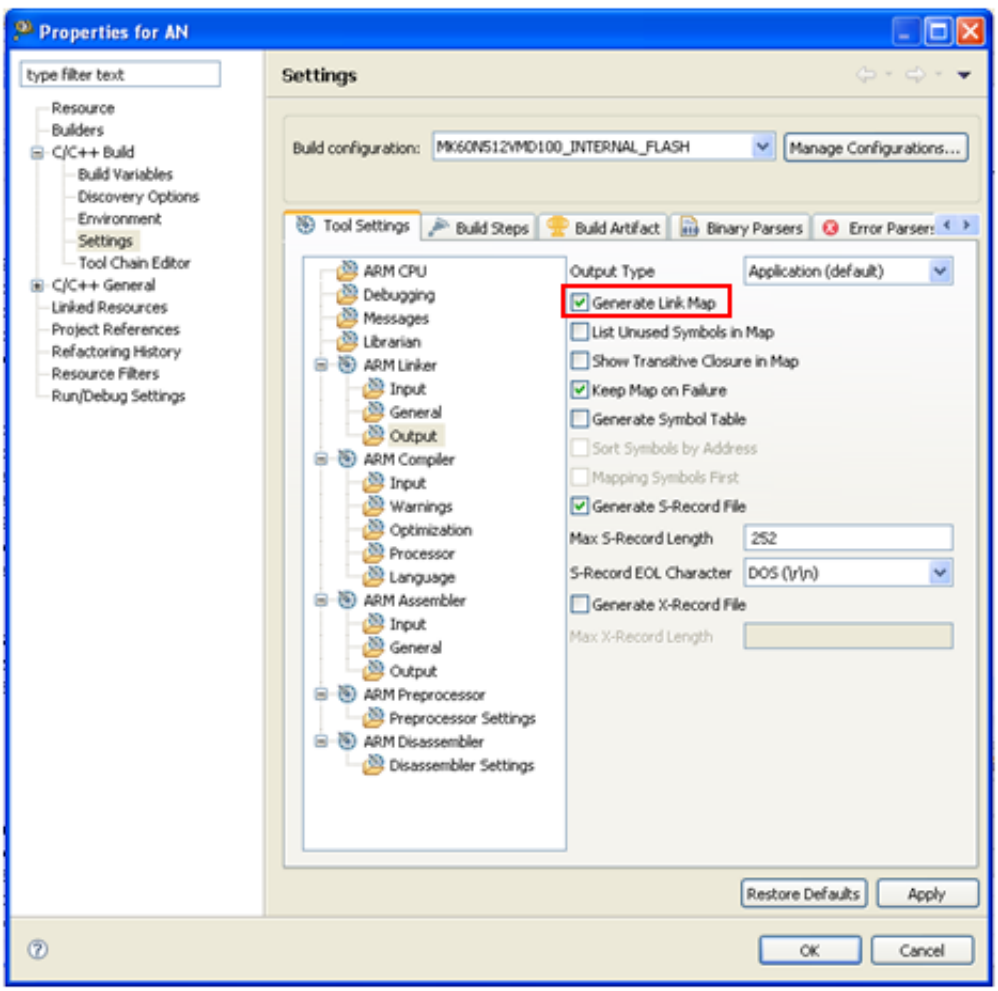



**图 8. 启用 Link Map 输出。**

5. 阅读.xmap 文件，以检查应用代码是否位于所需的闪存地址中。

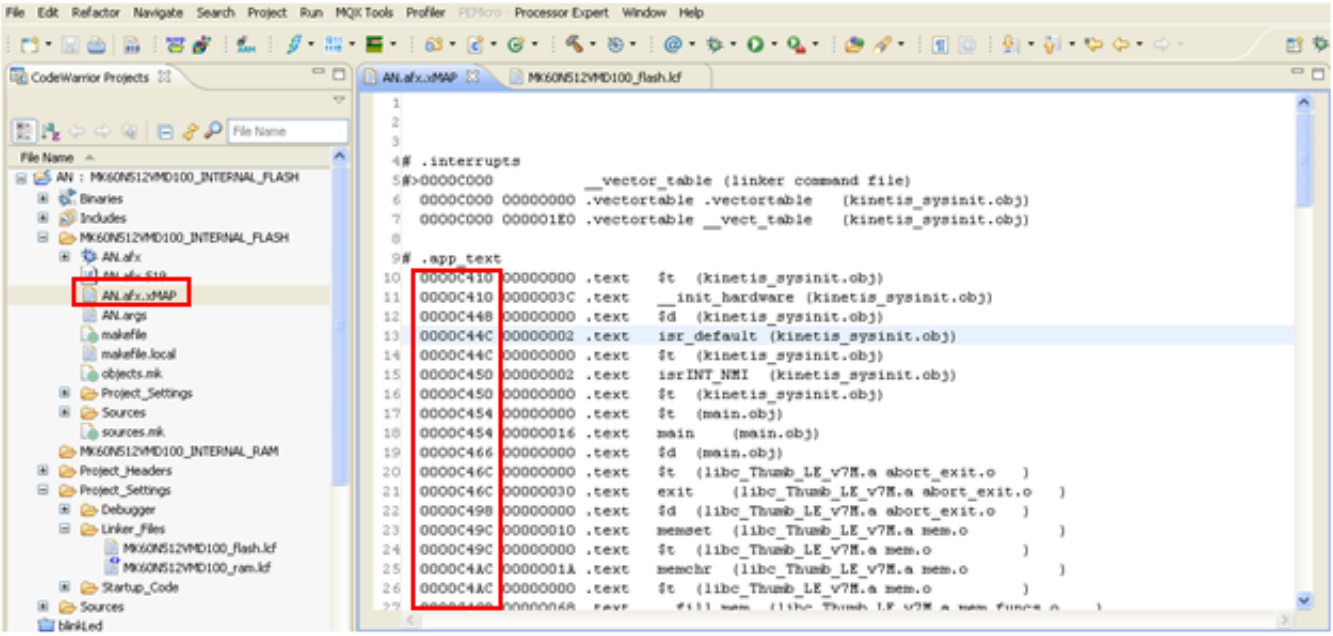



**图 9. 已在 CW 中修改链接器命令文件后面的存储器映射输出文件。**

现在，可以通过以太网引导加载程序将生成的.S19 文件下载到您的 MCU。您可以按下图所示查找.S19 文件。第 4 节的操作需要使用此文件。

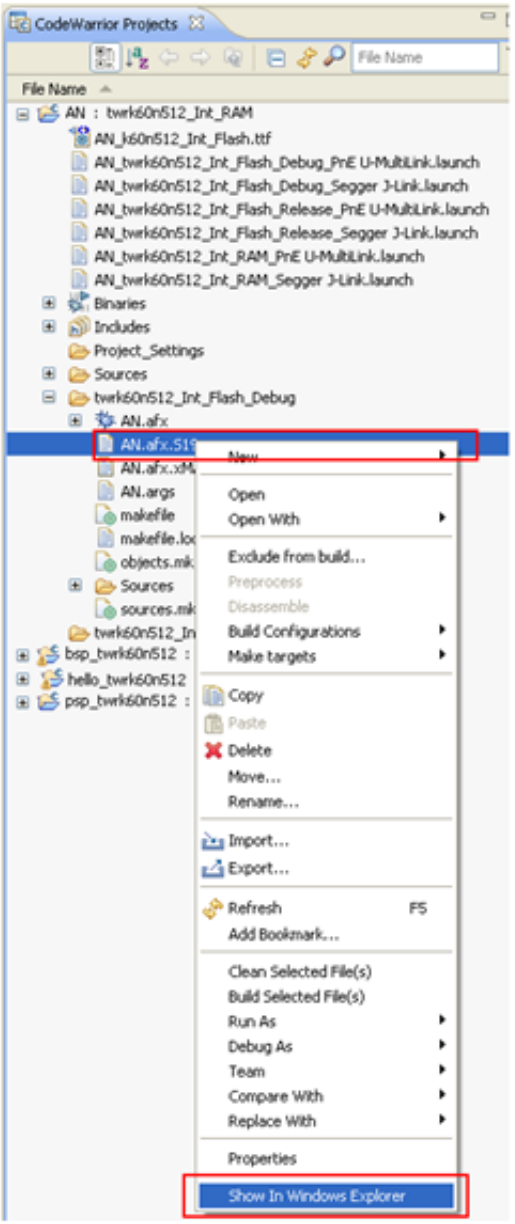


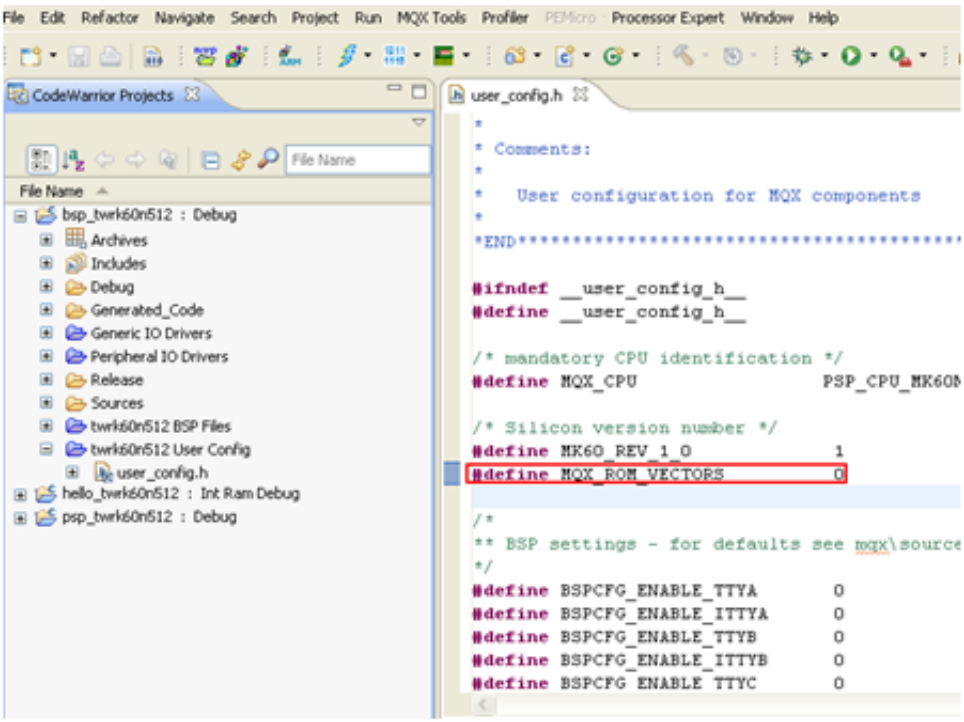

**图 10. .S19 文件的位置。**

## 3.3 在 CodeWarrior for MCU 10.1 中开发支持以太网引导加载程序的 MQX 应用

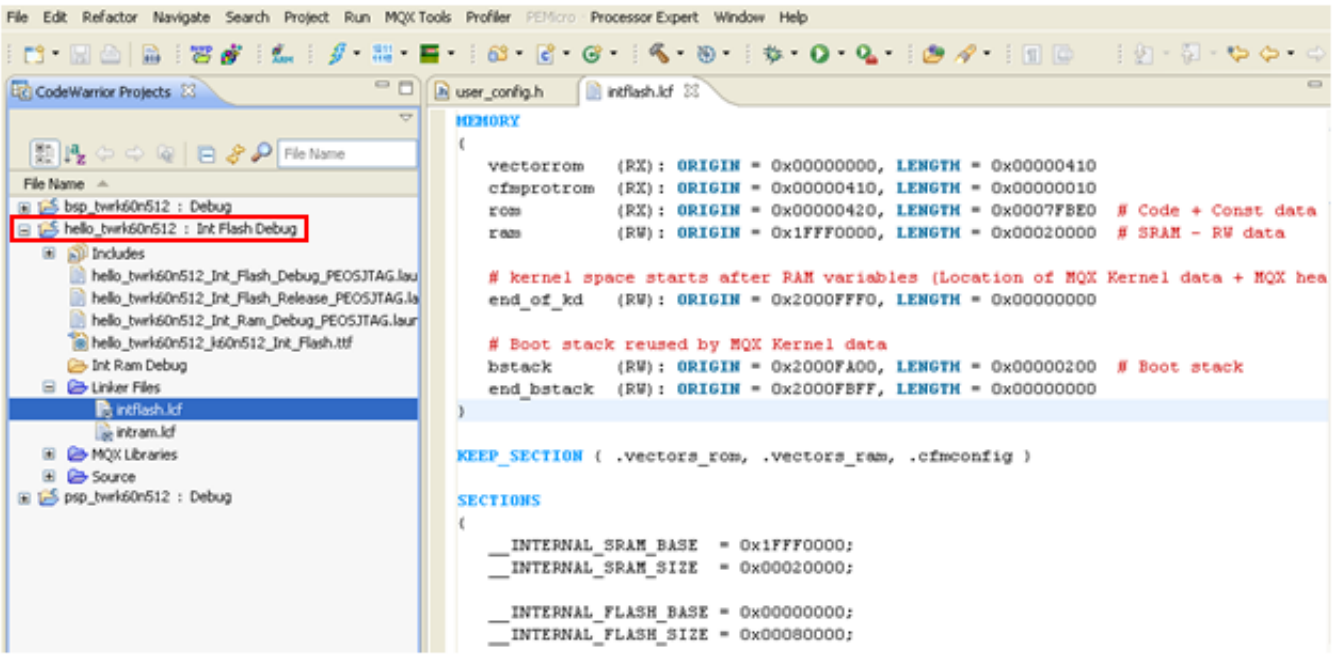
本方案使用一个“Hello World”工程，该工程可以在 C:\Program Files\Freescale\Freescale MQX 3.7\mqx\examples\hello\cw10\hello_twrk60n512 中找到。

1. 将 hello_twrk60n512 工程的.project 文件拖放到 CodeWarrior 的“Projects”窗口中。
2. 默认情况下，MQX 会将向量表分配到闪存的第一个地址，即 0x0000-0000。由于此闪存区由以太网引导加载程序使用，因此，需要在 RAM 段中重新分配向量表。要重新分配，请打开库（bsp_twrk60n512 和 psp_twrk60n512），将以下宏添加到 user_config.h 中，然后重新编译这两个库。您可以在“入门”文档 C:\Program Files\Freescale\Freescale MQX 3.7\doc\FSL_MQX_Getting_Started.pdf 的第 2.4 部分“Freescale CodeWarrior Development Studio 版本 10.1”中找到有关此过程的详细信息。

```
#define MQX_ROM_VECTORS          0
```




3. 将目标更改为 MK60N512VMD100_INTERNAL_FLASH，然后打开 MK60N512VMD100_flash.lcf




4. 修改.lcf 文件
   a. 将应用闪存中的起始地址从 0x0000_0000 修改为 0x0000_C000，包括 ROM 向量表。

```
# 48KB (Reserved for Ethernet Bootloader)
fnet_bootloader (RX)   : ORIGIN = 0x00000000, LENGTH = 0x0000C000
vectorrom    (RX): ORIGIN = 0x0000C000, LENGTH = 0x00000410
rom          (RX) : ORIGIN = 0x0000C420, LENGTH = 0x0007F000-0x0000C420
```

   b. 删除 m_cfmprtrom 区，因为您不需要使用此区。由于此段是为闪存保护寄存器保留的，因此，引导加载程序将处理此段。按如下所示为 m_cfmprotrom 段提供注释。

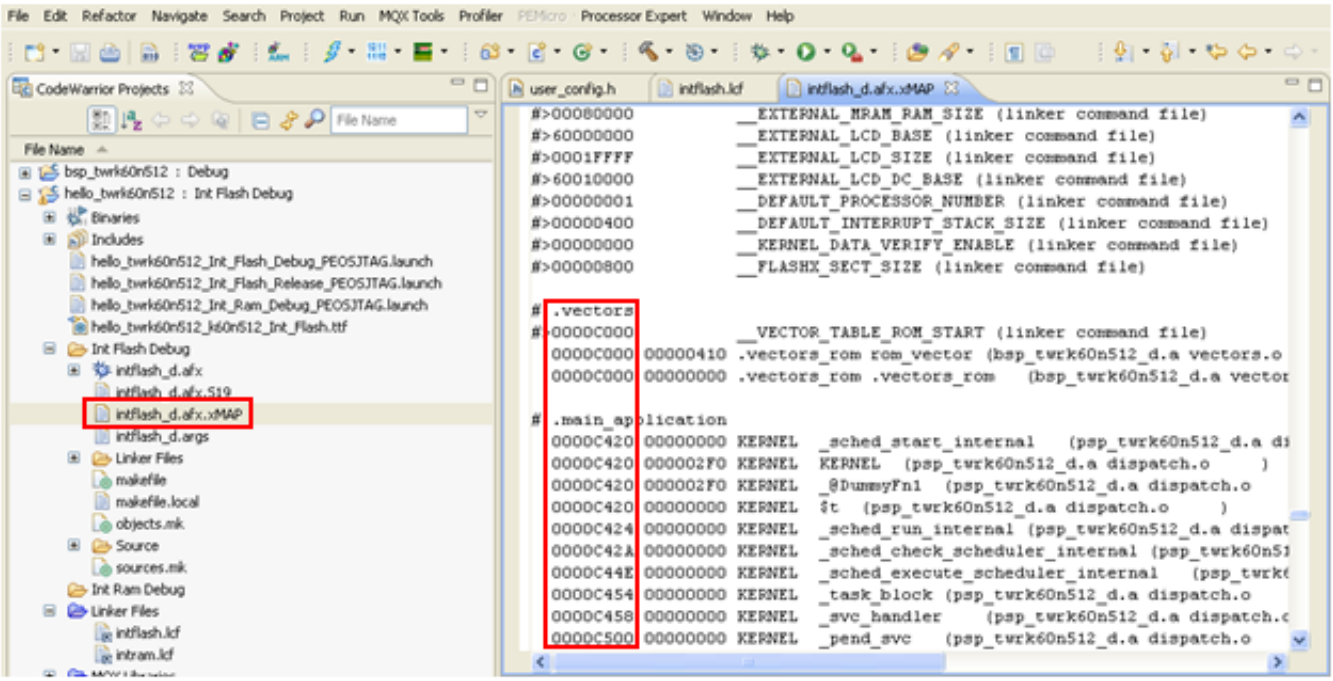
```
# m_cfmprotrom  (RX) : ORIGIN = 0x00000400, LENGTH = 0x00000010

# .cfmprotect :
# {
# *(.cfmconfig)
# . = ALIGN (0x4);
# } > m_cfmprotrom
```

c. 为引导加载程序参数创建闪存段。

```
# 4Kbytes (Last logical-block reserved for parameters)
fnet_params   (RW)  : ORIGIN = 0x0007F000,LENGTH = 0x00001000
```

附录 C 中显示了修改后的整个.lcf 文件。

5. 阅读.xmap 文件，以检查应用代码是否位于所需的闪存地址中。




现在，可以通过以太网引导加载程序将生成的.afx.S19 文件下载到您的 MCU。您可以在以下路径中找到该.S19 文件：C:\Program Files\Freescale\Freescale MQX 3.7\mqx\examples\hello\cw10\hello_twrk60n512\Int Flash Debug。下一节的操作需要使用此文件。

# 4 运行以太网引导加载程序

为演示本文档的内容，我们使用了以下工具和硬件：

- IAR Systems Embedded Workbench 6.0.3
- 终端程序（Windows 超级终端）
- TFTP 服务器和客户端（Tftpd 版本 3.33 http://tftpd32.jounin.net ）
- 本文档随附的以太网引导加载程序源代码
- TWR-K60N512-KIT

以下步骤说明如何在 MK60N512 中运行以太网引导加载程序：

1. 通过将 C:\Program Files\Freescale\FNET 1.0.0\fnet_demos\mk60n512\boot\iararm6.1 中的 twrk60n512_boot.ewp 文件拖放到 IAR IDE 中，打开以太网引导加载程序工程。
2. 使用 FLASH_512KB_PFLASH 目标进行编译，并下载到芯片。

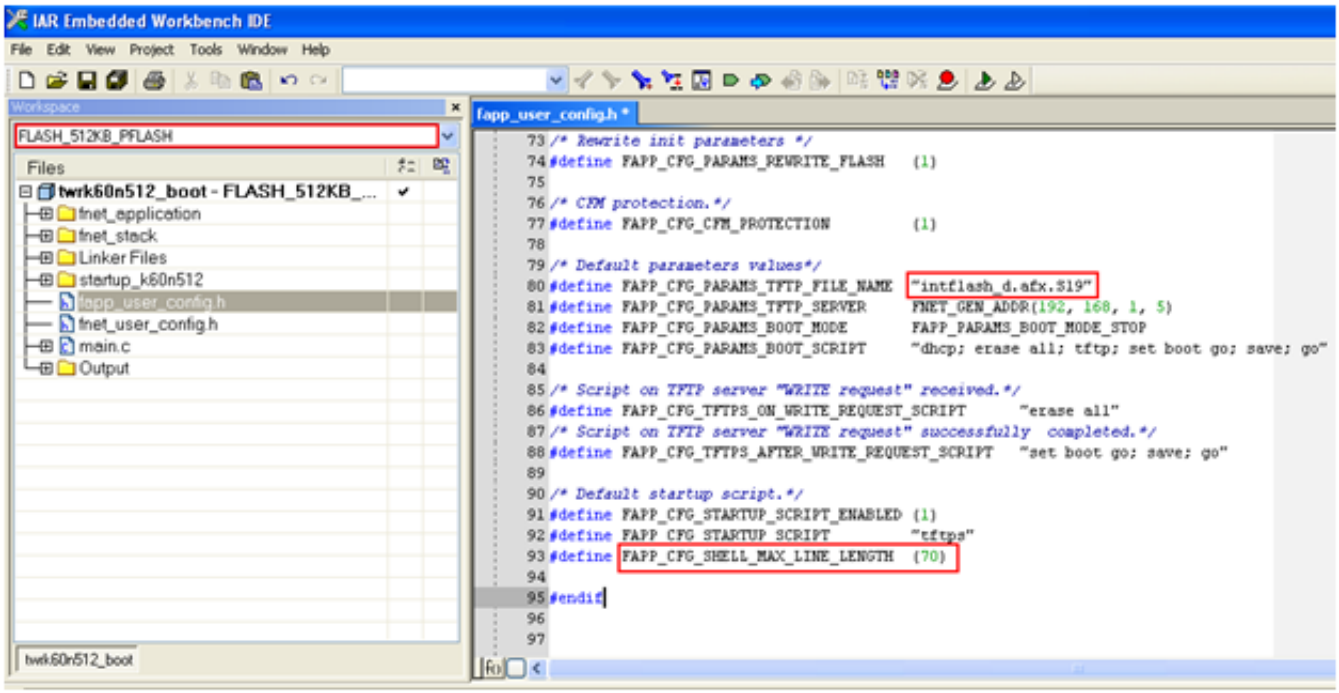
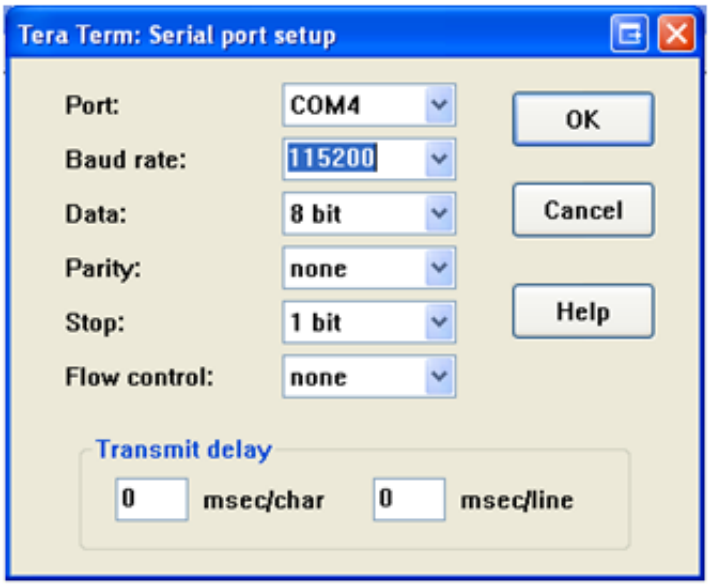



**图 11. 默认的引导加载程序配置。**

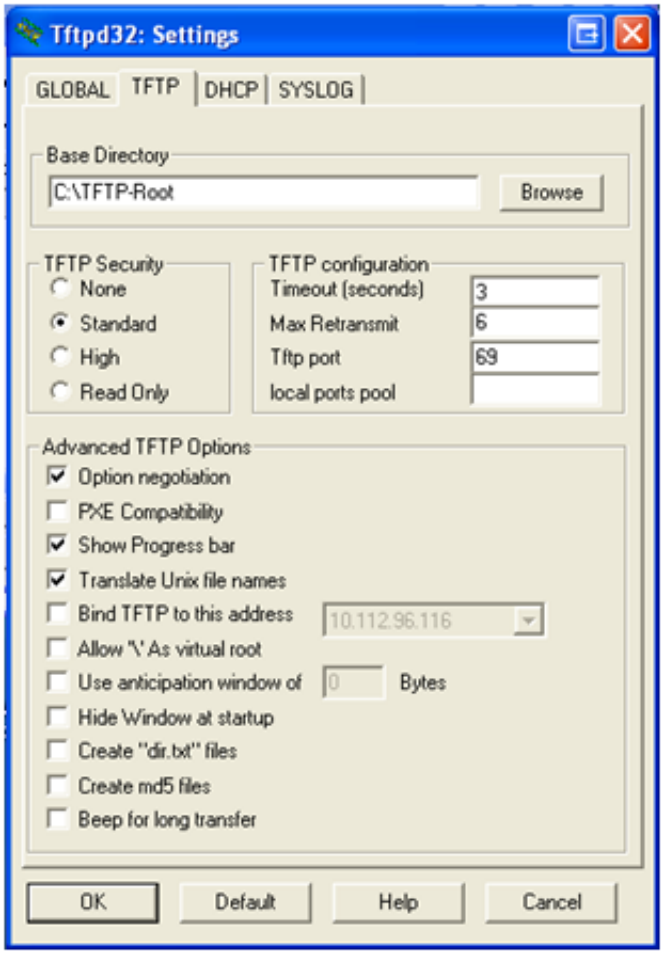
注意：您可以更改以太网引导加载程序的默认参数，例如，s-rec 文件（.s19 或.srec）的名称、字符串 Shell 的最大长度以及脚本。通过修改 fapp_user_config.h 来执行这些更改。下图显示了本文档中使用的值。

3. 根据 TWR-K60N512 用户手册准备您的 TWR-K60N512-KIT。
4. 借助 TWR-SER，使用 RS-232 公/母 DB-9 串行线缆（通过 COM 端口）将 PC 连接到塔式系统板。
5. 将塔式系统板连接到 LAN 网络。
6. 在塔式系统板连接到的 PC 上调用终端程序，并将该程序配置为：
   a. 波特率：115200。
   b. 数据位数：8。
   c. 奇偶校验：无。
   d. 停止位数：1。
   e. 流量控制：无。




7. 启动 TFTP 服务器：

a. 启动 TFTP 服务器应用程序。转到[开始]->[程序]->[Tftpd32]->[Tftpd32]
b. 按[设置]按钮，然后设置 TFTP 服务器基目录（例如 C:\TFTP-Root）:




8. 使用板背面的按钮关闭板的电源，然后再打开。也可以按 SW5 将板复位。
9. 在终端程序窗口中，应该可以看到如下所示的以太网引导加载程序欢迎消息:

```
Parameters loaded from Flash.

************************************************
 FNET Bootloader
************************************************
 FNET TCP/IP Stack for MK60N512
 Version 1.0.0
 Built Jul 28 2011 at 15:46:37 by IAR
 Copyright 2005-2011 by Freescale Semiconductor
 GNU LGPLv3
************************************************
 Interface        : eth0
 IP address       : 192.168.1.22 (set manually)
 Subnet mask      : 255.255.255.0
 Gateway          : 10.171.88.254
 MAC address      : 00:04:9F:15:51:27
 Link status      : connected
 TX Packets       : 3
 RX Packets       : 0
 Free Heap        : 14144
 DHCP client      : disabled
 TFTP server      : disabled

 Enter 'help' for command list.
************************************************

Startup script: tftps
************************************************
  TFTP server (192.168.1.22) started.
************************************************
BOOT>
```

10. 现在，以太网引导加载程序已准备好接受用户命令。输入“help”命令以显示引导加载程序 Shell 中所有可用命令的列表：

```
BOOT> help
>   help                                    - Display this help message
>    set [<parameter> <value>]              - Set parameter
>    get [<parameter>]                      - Get parameters
>   info                                    - Show detailed status
>   dhcp [release]                          - Start DHCP client
>   tftp [<image name>[<server ip>[<type>]]]  - TFTP firmware loader
> tftpup [<image name>[<server ip>[<type>]]]  - TFTP firmware uploader
>  tftps [release]                          - TFTP firmware server
>    mem                                    - Show memory map
>  erase all|[0x<erase address> <bytes>]    - Erase flash memory
>   save                                    - Save parameters to the FLASH
>     go [0x<address>]                      - Start application at address
>  reset                                    - Reset the board
BOOT>
```

键入“dhcp”以获取 IP 地址。如果您未使用 DHCP 客户端，MCU 将使用默认 IP 地址，该地址可以在 fnet_user_config.h 文件中的以下行上修改：

```
#define FNET_CFG_ETH_IP_ADDR        (FNET_GEN_ADDR(192, 168, 1, 22))
```

```
BOOT> dhcp
Press [Ctr+C] to cancel.
```

```
Sending DHCP discover...

************************************************
 DHCP has updated/renewed parameters:
************************************************
 Interface     : eth0
 IP address    : 10.112.102.45
 Subnet mask   : 255.255.255.0
 Gateway       : 10.112.102.254
```

11. 键入“**set tfpt <IP** 地址**>**”以设置 tfpt 固件服务器。在本例中，请使用 10.112.96.116。

```
BOOT> set tftp 10.112.96.116
 tftp     : 10.112.96.116
```

12. 键入“**set script 'dhcp\; erase all\; tftp\; set boot go\; save\; reset**”以更改脚本参数。

```
BOOT> set script 'dhcp\; erase all\; tftp\; set boot go\; save\; reset'
script   : dhcp; erase all; tftp; set boot go; save; reset
```

13. 为系统正确地设置了所有参数后，将“**boot**”参数设置为脚本值。为此，请键入“**set boot script**”以设置脚本值。

```
BOOT> set boot script
 boot     : script
```

14. 输入“**save**”命令将所有运行时参数保存到非易失性存储器（位于片上闪存的最后一个擦除页面中）。
15. 键入“**reset”**。
16. 在终端程序中，应该可以看到包含脚本的以太网引导加载程序已加载，并已准备好运行用户应用：

```
Parameters loaded from Flash.

************************************************
 FNET Bootloader
************************************************
 FNET TCP/IP Stack for MK60N512
 Version 1.0.0
 Built Jul 28 2011 at 15:46:37 by IAR
 Copyright 2005-2011 by Freescale Semiconductor
 GNU LGPLv3
************************************************
 Interface         : eth0
 IP address        : 192.168.1.22 (set manually)
 Subnet mask       : 255.255.255.0
 Gateway           : 10.112.102.254
 MAC address       : 00:04:9F:15:51:27
 Link status       : connected
 TX Packets        : 4
 RX Packets        : 0
 Free Heap         : 14144
 DHCP client       : disabled
```

```
 TFTP server       : disabled

 Enter 'help' for command list.
************************************************

Press any key to stop (script):   0
```

17. 如果在复位后的 5 秒内按任意键，以太网引导加载程序将执行前面设定的脚本。

```
dhcp; erase all; tftp; set boot go; save; reset
Press [Ctr+C] to cancel.
Sending DHCP discover...

***********************************************
 DHCP has updated/renewed parameters:
***********************************************
 Interface     : eth0
 IP address    : 10.112.102.45
 Subnet mask   : 255.255.255.0
 Gateway       : 10.112.102.254

Erasing...
 0x00000000 to 0x0000BFFF skipped
 0x0000C000 to 0x0007F7FF erased
 0x0007F800 to 0x0007FFFF skipped
Press [Ctr+C] to cancel.
TFTP downloading 'intflash.afx.S19' (srec) from 10.112.96.116: /
Entry point set to 0x00012CA0

TFTP completed (140540 bytes)
 boot     : go
Parameters saved

Parameters loaded from Flash.

***********************************************
 FNET Bootloader
***********************************************
 FNET TCP/IP Stack for MK60N512
 Version 1.0.0
 Built Jul 28 2011 at 15:46:37 by IAR
 Copyright 2005-2011 by Freescale Semiconductor
 GNU LGPLv3
***********************************************
 Interface         : eth0
 IP address        : 192.168.1.22 (set manually)
 Subnet mask       : 255.255.255.0
 Gateway           : 10.112.102.254
 MAC address       : 00:04:9F:15:51:27
 Link status       : connected
 TX Packets        : 4
 RX Packets        : 0
 Free Heap         : 14144
 DHCP client       : disabled
 TFTP server       : disabled

 Enter 'help' for command list.
***********************************************

Press any key to stop (go):   0
```

18. 此时，应用已驻留在闪存区域中，如果在 5 秒后按任意键，应用将运行，直到执行复位时为止。

```
Press any key to stop (go):   0=Hello World
Hello World
Hello World
Hello World
```

# 5 结语

以太网引导加载程序包括 3 种不同的应用方案。介绍的选项为:

- 在 IAR 中开发的支持以太网引导加载程序的裸板应用。
- 在 CodeWarrior for MCU 10.1 中开发的支持以太网引导加载程序的裸板应用。
- 在 CodeWarrior for MCU 10.1 中开发的支持以太网引导加载程序的 MQX 应用。

我们将会详细介绍每个选项，并提供所需的步骤让开发员在任何产品中嵌入以太网引导加载程序。多样化的实施方案使开发员能够为其应用选择最佳的选项。

我们还提供了引导加载程序的使用指南，用户可以参考其中的具体信息来使用以太网引导加载程序。

本应用笔记可以帮助开发员快速方便地在其产品中包含以太网引导加载程序。

# 6 附录 A：IAR 链接器配置文件(.icf)

```
/*###ICF### Section handled by ICF editor, don't touch! ****/
/*-Editor annotation file-*/
/* IcfEditorFile="$TOOLKIT_DIR$\config\ide\IcfEditor\cortex_v1_0.xml" */
/*-Specials-*/
define symbol __ICFEDIT_intvec_start__ = 0x0000C000;
/*-Memory Regions-*/
define symbol __ICFEDIT_region_ROM_start__ = 0x0000C000;
define symbol __ICFEDIT_region_ROM_end__   = 0x00080000;
define symbol __ICFEDIT_region_RAM_start__ = 0x1FFF0410;
define symbol __ICFEDIT_region_RAM_end__   = 0x20010000;
/*-Sizes-*/
define symbol __ICFEDIT_size_cstack__ = 0x1000;
define symbol __ICFEDIT_size_heap__   = 0x200;
/**** End of ICF editor section. ###ICF###*/

define symbol FappParams_start__= __ICFEDIT_region_ROM_end__ - (2*0x400); //FAPP
define symbol FappParams_end__  = __ICFEDIT_region_ROM_end__-1; //FAPP

//define symbol __region_RAM2_start__ = 0x20000000;
//define symbol __region_RAM2_end__ = 0x20010000;

define exported symbol __VECTOR_TABLE = 0x0000C000;
define exported symbol __VECTOR_RAM = 0x1fff0000;

define exported symbol __BOOT_STACK_ADDRESS = __ICFEDIT_region_RAM_end__ - 8;           //
0x2000FFF8;

define symbol __code_start__ = 0x0000C410;

define memory mem with size = 4G;
//define region ROM_region   = mem:[from __ICFEDIT_region_ROM_start__   to
__ICFEDIT_region_ROM_end__];
//define region RAM_region   = mem:[from __ICFEDIT_region_RAM_start__   to
__ICFEDIT_region_RAM_end__] | mem:[from __region_RAM2_start__ to __region_RAM2_end__];
define region ROM_region   = mem:[from __ICFEDIT_region_ROM_start__   to
__ICFEDIT_region_ROM_end__-1] - mem:[from FappParams_start__  to FappParams_end__];
define region RAM_region   = mem:[from __ICFEDIT_region_RAM_start__   to
__ICFEDIT_region_RAM_end__-1]; //| mem:[from __region_RAM2_start__ to __region_RAM2_end__-1];

define block CSTACK    with alignment = 8, size = __ICFEDIT_size_cstack__   { };
define block HEAP      with alignment = 8, size = __ICFEDIT_size_heap__     { };

initialize manually { readwrite };
initialize manually { section .data};
initialize manually { section .textrw };
do not initialize  { section .noinit };

define block CodeRelocate { section .textrw_init };
define block CodeRelocateRam { section .textrw };

place at address mem:__ICFEDIT_intvec_start__ { readonly section .intvec };
place at address mem:__code_start__ { readonly section .noinit };
place at address mem:FappParams_start__ { section fapp_params }; //APP

place in ROM_region   { readonly, block CodeRelocate};
```

```
place in RAM_region   { readwrite, block CodeRelocateRam,
                        block CSTACK, block HEAP };
```

# 7 附录 B：CodeWarrior 裸板工程中的链接器命令文件(.lcf)

```
# Default linker command file.
MEMORY {
fnet_bootloader (RX)  : ORIGIN = 0x00000000, LENGTH = 0x0000C000 # 48KB (Reserved for FNET
Bootloader)
m_interrupts  (RX) : ORIGIN = 0x0000C000, LENGTH = 0x000001E0
m_text        (RX) : ORIGIN = 0x0000C410, LENGTH = 0x00080000-0x0000C410
fnet_params   (RW)  : ORIGIN = 0x20010000,LENGTH = 0x00001000 # 4Kbytes (Last logical-block
reserved for parameters)
m_data        (RW) : ORIGIN = 0x1FFF0000, LENGTH = 0x00020000
#m_cfmprotrom  (RX) : ORIGIN = 0x0000C400, LENGTH = 0x00000010
}

KEEP_SECTION { .vectortable }
KEEP_SECTION { .cfmconfig }

SECTIONS {

.interrupts :
{
    __vector_table = .;
    * (.vectortable)
    . = ALIGN (0x4);
} > m_interrupts

#.cfmprotect :
#{
#    *(.cfmconfig)
#    . = ALIGN (0x4);
#} > m_cfmprotrom

.app_text:
{
      ALIGNALL(4);
      * (.init)
      * (.text)
            .= ALIGN(0x8) ;
      * (.rodata)
            .= ALIGN(0x4) ;
    ___ROM_AT = .;
} > m_text
.app_data: AT(___ROM_AT)
{
      * (.sdata)
      * (.data)
            .= ALIGN(0x4) ;
      *(.ARM.extab)
            .= ALIGN(0x4) ;
       __exception_table_start__ = .;
      EXCEPTION
       __exception_table_end__ = .;
            .= ALIGN(0x4) ;
       __sinit__ = .;
      STATICINIT
      .= ALIGN(0x8) ;
} > m_data
.bss :
{
      .= ALIGN(0x4) ;
      __START_BSS = .;
```

```
      * (.bss)
      __END_BSS = .;
      .= ALIGN(0x8) ;
} >> m_data

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

__SP_INIT = . + 0x00008000;
__heap_addr = __SP_INIT;
__heap_size = 0x00008000;

}
```

# 8 附录 C: CodeWarrior MQX 工程中的链接器命令文件(.lcf)

```
MEMORY
{
   fnet_bootloader (RX)  : ORIGIN = 0x00000000, LENGTH = 0x0000C000 # 48KB (Reserved for
FNET Bootloader)
   vectorrom   (RX): ORIGIN = 0x0000C000, LENGTH = 0x00000410
   #cfmprotrom  (RX): ORIGIN = 0x00000410, LENGTH = 0x00000010
   rom         (RX): ORIGIN = 0x0000C420, LENGTH = 0x0007FBE0  # Code + Const data
   ram         (RW): ORIGIN = 0x1FFF0000, LENGTH = 0x00020000  # SRAM - RW data

   # kernel space starts after RAM variables (Location of MQX Kernel data + MQX heap)
   end_of_kd   (RW): ORIGIN = 0x2000FFF0, LENGTH = 0x00000000

   # Boot stack reused by MQX Kernel data
   bstack      (RW): ORIGIN = 0x2000FA00, LENGTH = 0x00000200  # Boot stack
   end_bstack  (RW): ORIGIN = 0x2000FBFF, LENGTH = 0x00000000

   fnet_params   (RW)  : ORIGIN = 0x0007F000,LENGTH = 0x00001000
}

KEEP_SECTION { .vectors_rom, .vectors_ram, .cfmconfig }

SECTIONS
{
   __INTERNAL_SRAM_BASE  = 0x1FFF0000;
   __INTERNAL_SRAM_SIZE  = 0x00020000;

   __INTERNAL_FLASH_BASE = 0x00000000;
   __INTERNAL_FLASH_SIZE = 0x00080000;

   __INTERNAL_FLEXNVM_BASE = 0;
   __INTERNAL_FLEXNVM_SIZE = 0;

   __EXTERNAL_MRAM_BASE  = 0x70000000;
   __EXTERNAL_MRAM_SIZE  = 0x00080000;
   __EXTERNAL_MRAM_ROM_BASE = 0x70000000;
   __EXTERNAL_MRAM_ROM_SIZE = 0x00000000;
   __EXTERNAL_MRAM_RAM_BASE = 0x70000000;
   __EXTERNAL_MRAM_RAM_SIZE = 0x00080000;
```

```
__EXTERNAL_LCD_BASE = 0x60000000;
__EXTERNAL_LCD_SIZE = 0x1FFFF;
__EXTERNAL_LCD_DC_BASE = 0x60010000;

# MQX link time configurations
__DEFAULT_PROCESSOR_NUMBER = 1;
__DEFAULT_INTERRUPT_STACK_SIZE = 1024;
__KERNEL_DATA_VERIFY_ENABLE = 0;    # Test SDRAM read/write

# Flashx configurations
__FLASHX_SECT_SIZE = 0x800;

  .vectors :
  {
      __VECTOR_TABLE_ROM_START = .;         # Runtime vector table in sram
      *(.vectors_rom)
     . = ALIGN (0x4);
  } > vectorrom

#  .cfmprotect :
#  {
#      *(.cfmconfig)
#      . = ALIGN (0x4);
#  } > cfmprotrom

  .main_application :
  {
      *(KERNEL)
      *(S_BOOT)
      *(IPSUM)
      *(.text)
      *(.init)
      *(.fini)
      *(.eini)
      *(.ctors)
      *(.dtors)
      . = ALIGN(0x4);
      *(.rodata)
      . = ALIGN(0x4);
      *(.rdata)
      . = ALIGN(0x4);
      *(.exception)
      . = ALIGN(0x4);
      __exception_table_start__ = .;
      EXCEPTION
      __exception_table_end__ = .;
      __sinit__ = .;
      STATICINIT

      . = ALIGN(0x4);
      __COPY_OF_DATA = .;
  } > rom

  .main_application_data : AT(__COPY_OF_DATA)
  {
      . = ALIGN(0x10000);
      __VECTOR_TABLE_RAM_START = .;        # Runtime vector table in sram
      *(.vectors_ram)

      . = ALIGN(512);
      __BDT_BASE = .;
      *(.usb_bdt)
      __BDT_END = .;

      __START_DATA = .;
      *(.data)
      __END_DATA = .;

      . = ALIGN(0x4);
      __START_SDATA = .;
```

```
        *(.sdata)
        __END_SDATA = .;

        . = ALIGN(0x4);
        __SDA_BASE  = .;
        __SDA_BASE_ = __SDA_BASE;
        . = ALIGN(16);
    } > ram

    .main_application_bss :
    {
        . = ALIGN(0x10);
        __START_SBSS = .;
        *(.sbss)
        *(SCOMMON)
        __END_SBSS = .;

        __START_BSS = .;
        *(.bss)
        *(COMMON)
        __END_BSS = .;
        . = ALIGN(16);
    } >> ram

    .kernel_data : #AT(ADDR(.main_application_bss) + SIZEOF(.main_application_bss))
    {
        __KERNEL_DATA_START = ALIGN(0x10);
    }
    .end_of_kernel_data :
    {
        __KERNEL_DATA_END = .;
    } > end_of_kd

    .boot_stack :
    {
        _stack_end = .;
    } > bstack

    .end_of_boot_stack :
    {
        _stack_addr  = .;
        __SP_INIT    = .;
        __BOOT_STACK_ADDRESS = .;
    } > end_bstack

    # Locate the ROM copy table into ROM after the initialized data
    _romp_at = __COPY_OF_DATA + SIZEOF(.main_application_data);

    .romp : AT (_romp_at)
    {
        __S_romp = _romp_at;
        WRITEW(__COPY_OF_DATA);                    #ROM start address
        WRITEW(ADDR(.main_application_data));      #RAM start address
        WRITEW(SIZEOF(.main_application_data));    #size
        WRITEW(0);
        WRITEW(0);
        WRITEW(0);
    }

    _flashx_start = __COPY_OF_DATA + SIZEOF(.main_application_data) + SIZEOF(.romp);

    # flashx working area spans across the whole rest of Flash memory
    __FLASHX_START_ADDR = ((_flashx_start + __FLASHX_SECT_SIZE - 1) / __FLASHX_SECT_SIZE) *
__FLASHX_SECT_SIZE;
    __FLASHX_END_ADDR = __INTERNAL_FLASH_BASE + __INTERNAL_FLASH_SIZE;

}

/* EOF */
```

***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.

Freescale, the Freescale logo, and Kinetis, are trademarks of Freescale Semiconductor, Inc., Reg. U.S. Pat. & Tm. Off. All other product or service names are the property of their respective owners.

© 2011 Freescale Semiconductor, Inc.

© 2011 飞思卡尔半导体有限公司

Document Number AN4367
Revision 0, 09/2011