Freescale Semiconductor
Application Note

Document Number: AN4367
Rev. 0, 09/2011

Ethernet Bootloader for MCU

by:

Alejandro Lozano, Alí Piña
Mexico
Guadalajara

Contents

1 Introduction

This document describes how to use an Ethernet bootloader provided by Freescale. This Ethernet bootloader is supported in different platforms, such as Kinetis, MCF52259, and MCF51CN128. This application note is focused on the Kinetis family, which has an ARM Cortex-M4 core. The software and main tests were performed on a K60N512. It explains how to modify the .lcf and .icf link files of CW and IAR IDEs respectively, in order to add the Ethernet bootloader support for your application.

2 Overview

A bootloader is a small piece of software that allows the user to download/update either code or data into flash memory. The communication layer between the MCU and the host, which contains the new application binary, might differ depending on the use case. This communication interface could be SPI I2C, Ethernet, and USB, etc. For this scenario, use an Ethernet communication interface. The bootloader provided will work as a TFTP client and the DHCP feature will be used.

freescale semiconductor

2.1 Bootloader architecture

The bootloader is placed into the first 48 KB of the on-chip flash memory, but the actual size of the bootloader can be smaller. The user application can be placed into the flash area above the bootloader flash area. The last 4 KB of the flash memory must be reserved for the bootloader's parameters storage, which is defined by the size of the erase-page of the on-chip flash memory.

The following figure shows the memory map and the bootloader memory footprint. The bootloader code is located in the first address of the flash memory, 0x0000_0000 to 0x0000_BFFF. The bootloader parameters are located in the first address of the last logical-block of the flash memory, 0x0007_F000 to 0x0008_0000 (in Kinetis MK60n512).

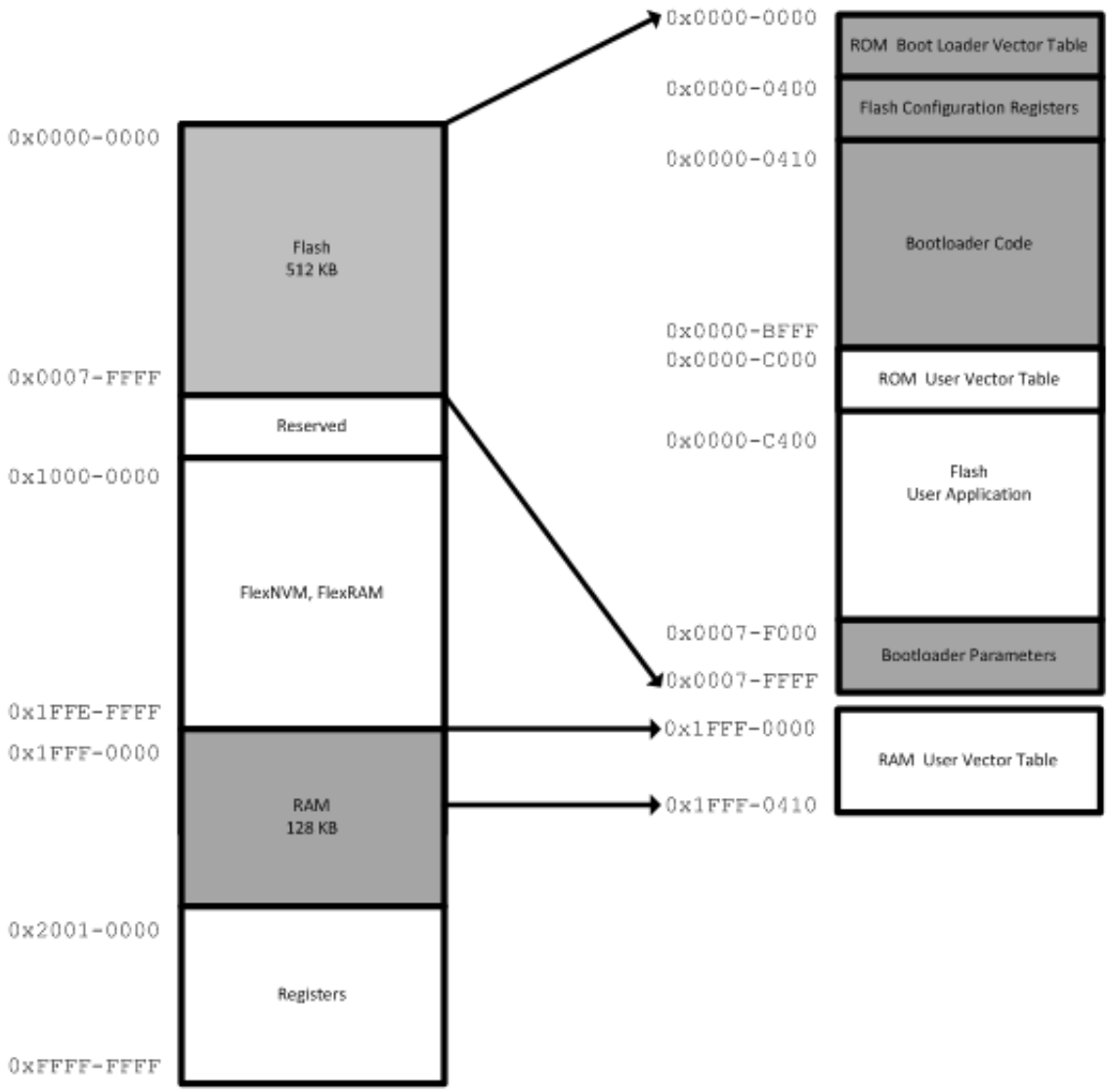


Figure 1. Memory map

2.2 Bootloader sequence

The following figure shows the bootloader sequence, after the power-on reset, a boot value is read. This boot value is the one that determines if the application will run after the reset ‘go’, execute a programmed script ‘script’ or the bootloader enters in a ‘stop’ mode.

The default boot value is stop. In this mode after the power-on reset the bootloader just executes. You can modify certain parameters of the bootloader like the image that will be downloaded; for instance, you can program a script that can be executed after a reset as long as the ‘script’ mode is set.

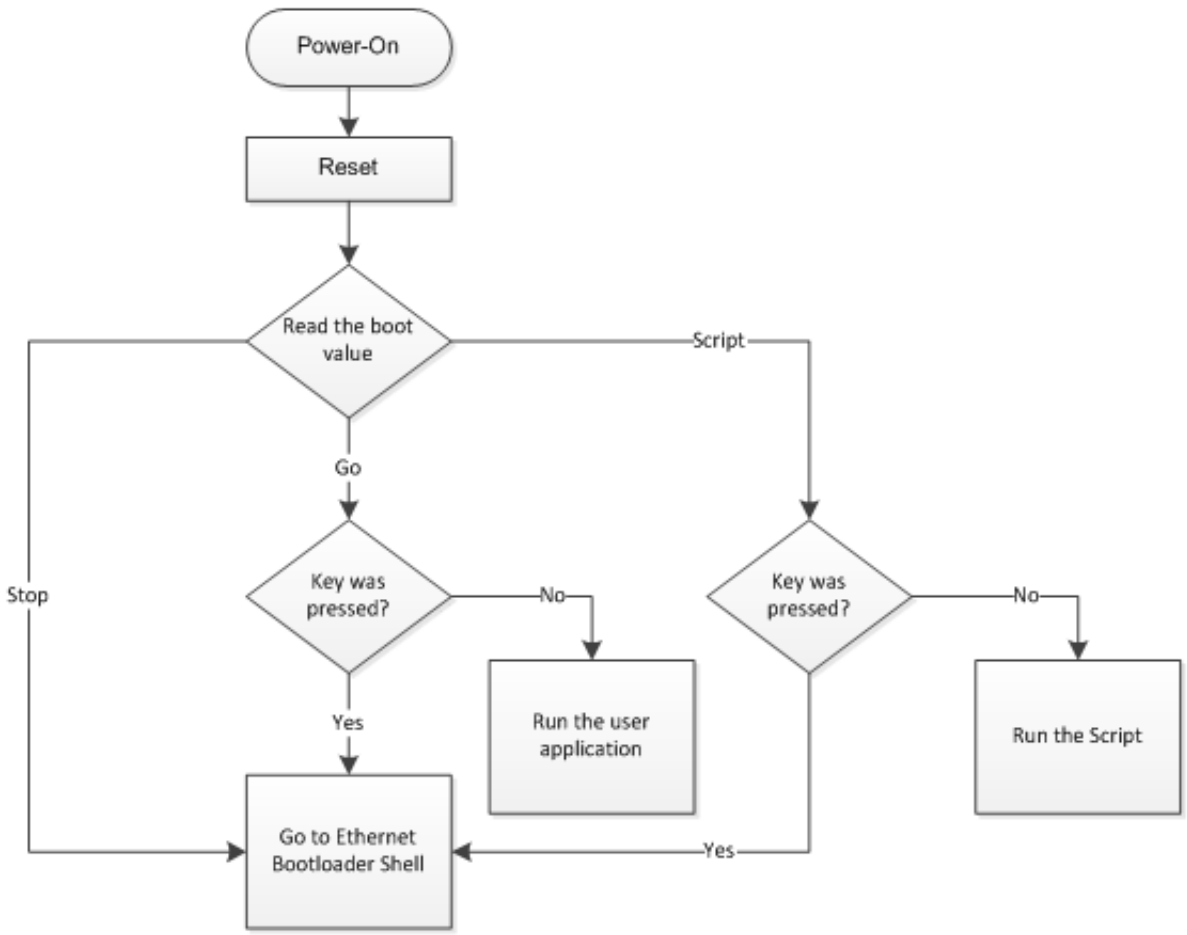


Figure 2. Ethernet bootloader sequence

In 'script' mode, you can program commands like: ‘**dhcp; erase all; tftp; set boot go; save; go**’

A brief description of this script is explained in "Running the ethernet bootloader." The script is either executed or not if a key is pressed within 5 seconds after reset.

The ‘go’ mode forces the MCU to enter into the user application, if any key is pressed within 5 seconds.

3 Developing an application with Ethernet bootloader support

This section details how to modify the application linker command file (.lcf for CW and .icf for IAR) to reserve space in the flash area where the bootloader will reside. The application code must be placed in a different section of flash area. This is where it becomes necessary to modify the LCF or the ICF for CW and IAR, respectively.

3.1 Developing a bareboard application in IAR with Ethernet bootloader support

At this point you can use your own project in IAR or an example project that you can search at the www.freescale.com for KINETIS512_SC.

This case will use a “Hello World” project that can be found at this path …\kinetis-sc\ build\iar\hello_world. The following steps show how to modify the application linker command file.

1. Open hello_world.eww workspace.
2. Change the target to FLASH_512KB_PFLASH, and open 512KB_Pflash.icf.

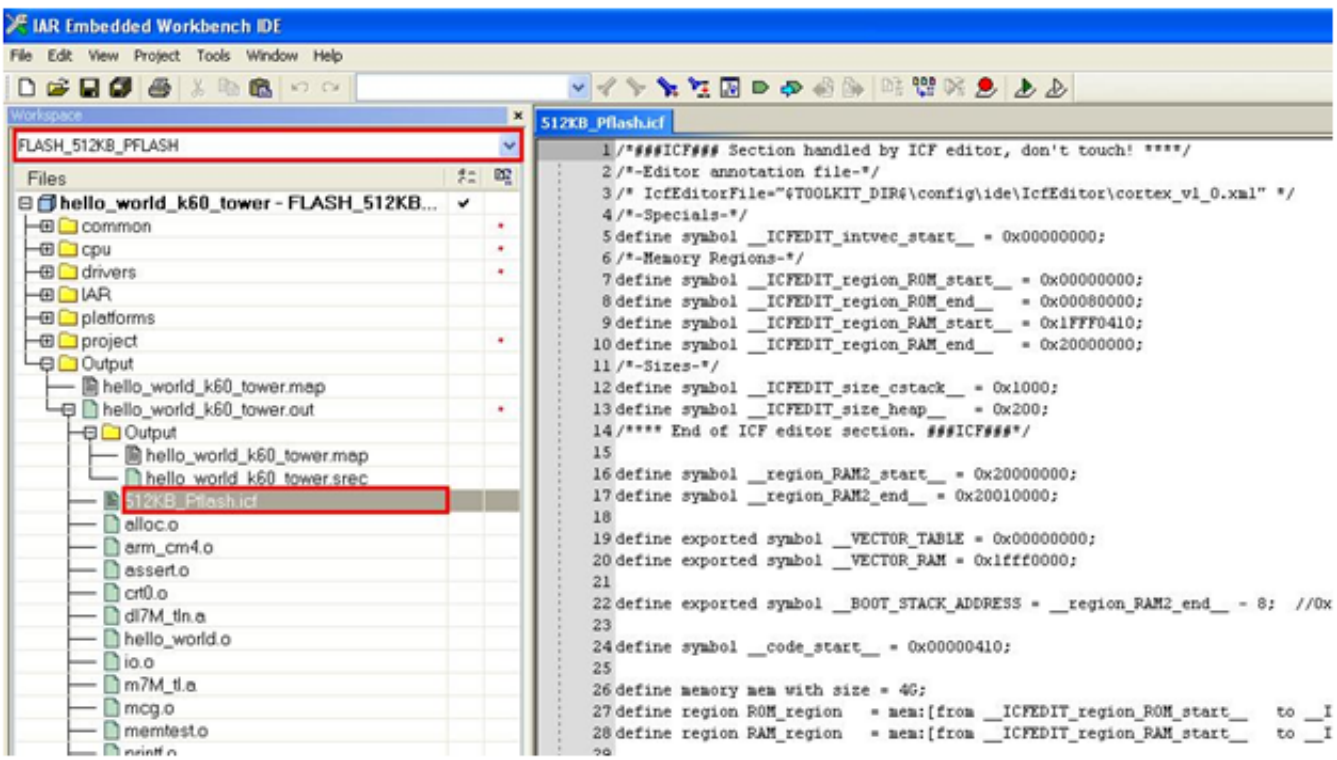


Figure 3. Linker command (.icf) file without notifications

3. Modifying the .icf link file.
   a. Modify the start address in flash of your application from 0x0000_0000 to 0x0000_C000, including ROM Vector Table and the code start.

```
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. Convert the split RAM section into only one RAM section by deleting section RAM2 and increasing RAM section up to 0x2001_0000.

```
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. Create a Flash section for the bootloader parameters. This section is needed only if the application is going to change the bootloader parameters. First you will need to define the start and end address of the section. So you need to define two symbols that represent those values:

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

Then, create the ROM region into which the application will be placed.

```
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
```

An entire modified .icf file is shown in Appendix A.

4. Compile and link the project. Read the .map file to check whether the application code is located in the expected flash address.

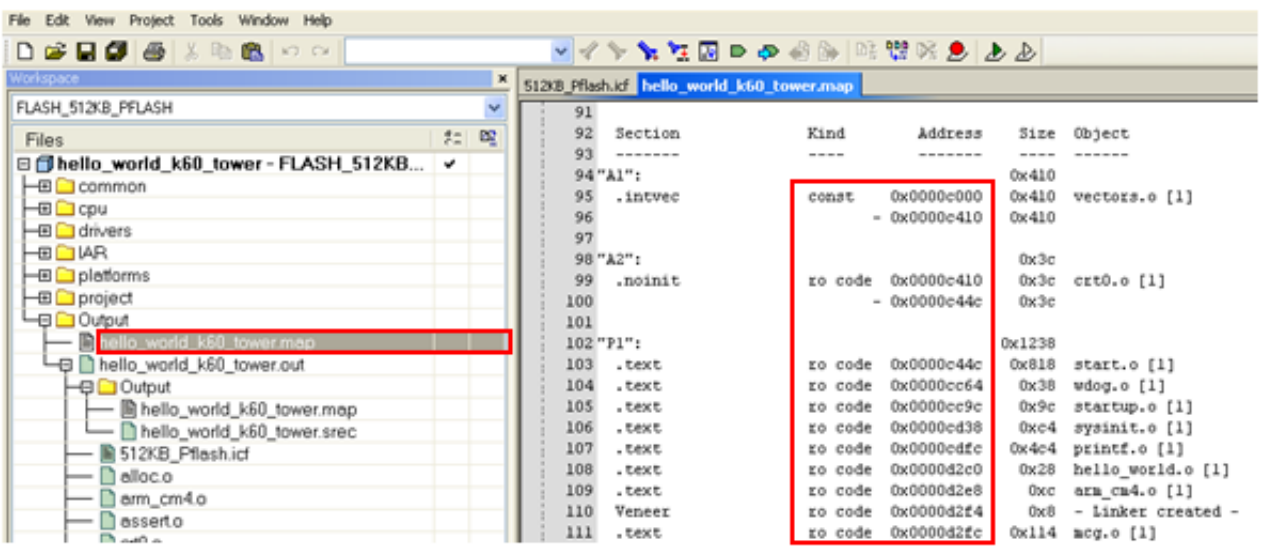


Figure 4. Memory map output file after linker command file is modified.

Now the generated .srec file is ready to be downloaded to your MCU through the Ethernet bootloader and is available in the output folder of the project. You can find it as shown in the following figure. This file is needed for section 4.

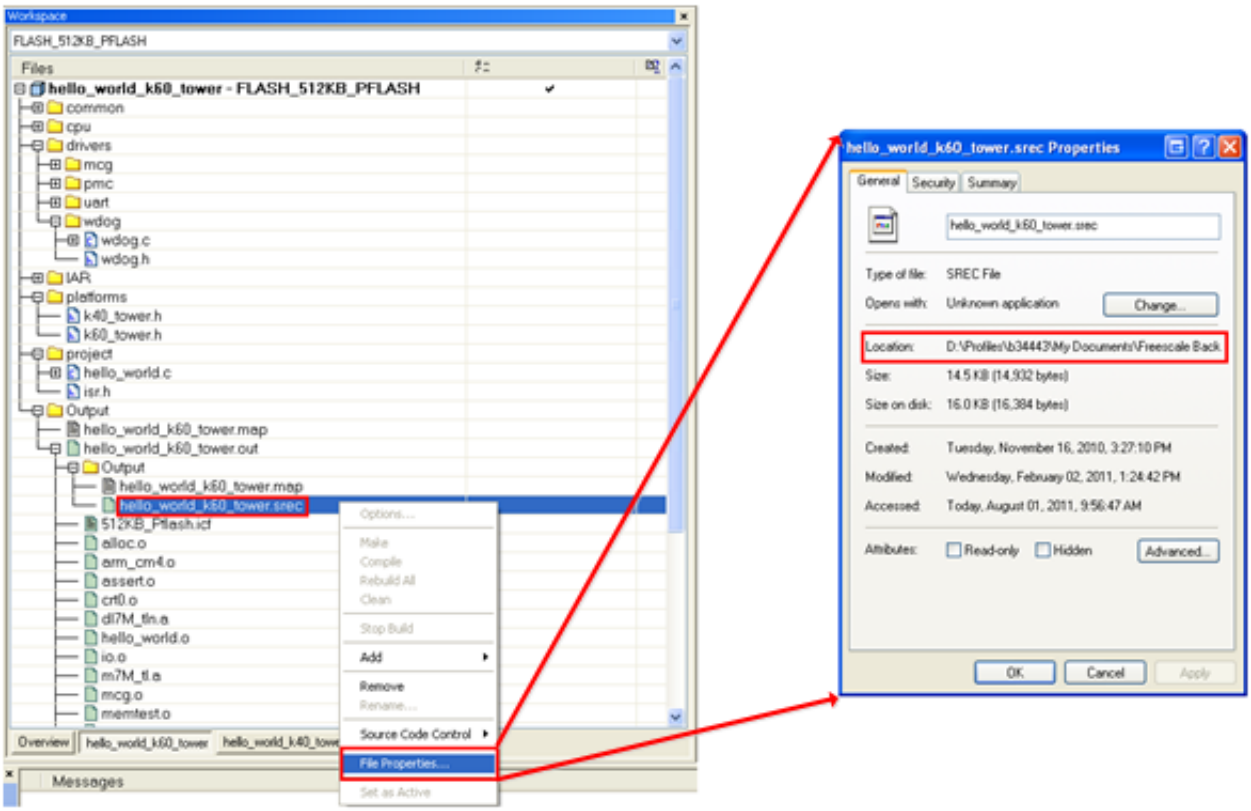


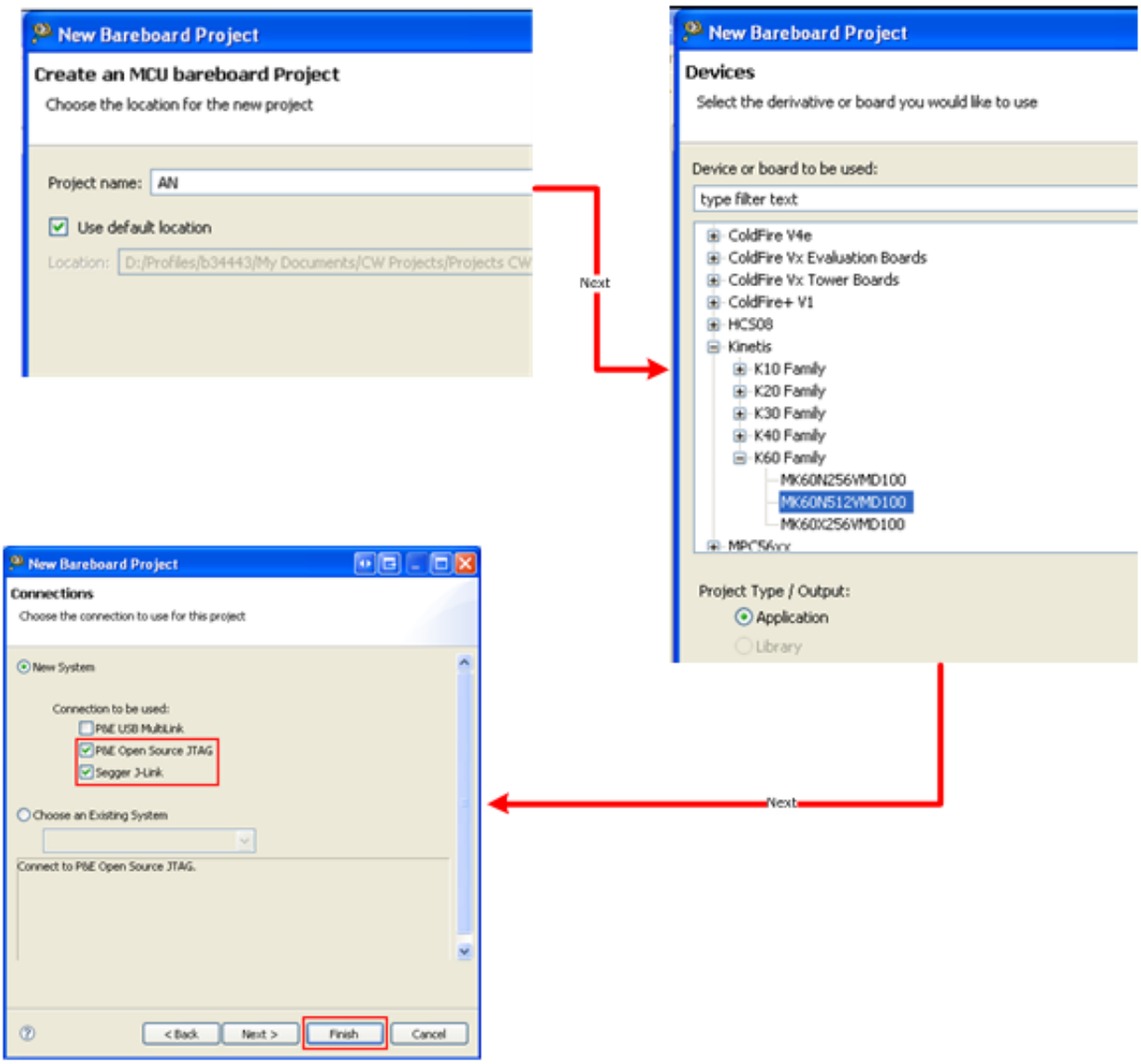
Figure 5. Location of the s-rec file.

3.2 Developing a bareboard application in CodeWarrior for MCU 10.1 with Ethernet bootloader support

In order to create a bareboard application, you can either create a new project in CodeWarrior (CW) 10.1, or use an existing project. For the purpose of this application note a new project is created.

1. Create a stationary bareboard project for MK60N512 going to File >New >Bareboard Project and follow the procedure shown below.

Figure 6. Creating a barebord project in CodeWarrior.



2. Change the target to MK60N512VMD100_INTERNAL_FLASH, and open MK60N512VMD100_flash.lcf

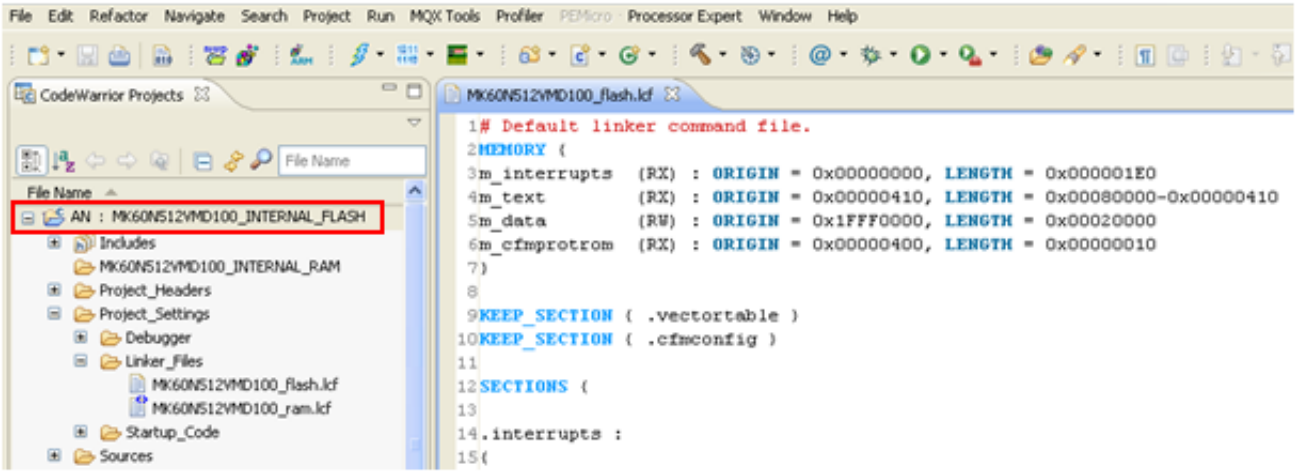


Figure 7. Internal flash target in CW.

3. Modify the .lcf file
   a. Modify the start address in flash of your application from 0x0000 to 0xC000, including ROM Vector Table.

```
# 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. Remove the m_cfmprtrom region, because you will not use this area. Because it is reserved for flash protection registers, the bootloader will handle this section.

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

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

   c. Create a flash section for the bootloader parameters.

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

   An entire modified .lcf file is shown in Appendix B.

4. Compile the project. By default, CodeWarrior does not generate a link map file; you will need to enable this feature as shown in the following figure.

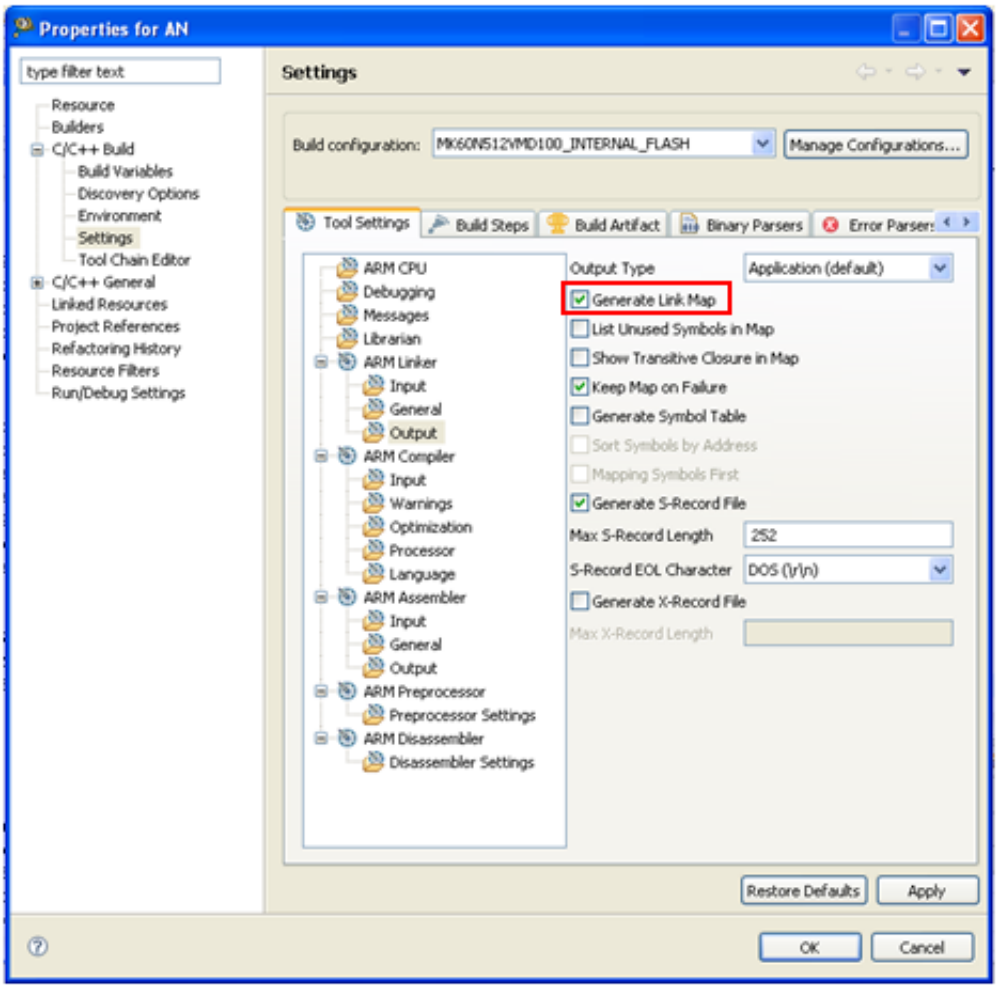


Figure 8. Enabling link map output.

5. Read the .xmap file to check whether the application code is located in the expected flash address.

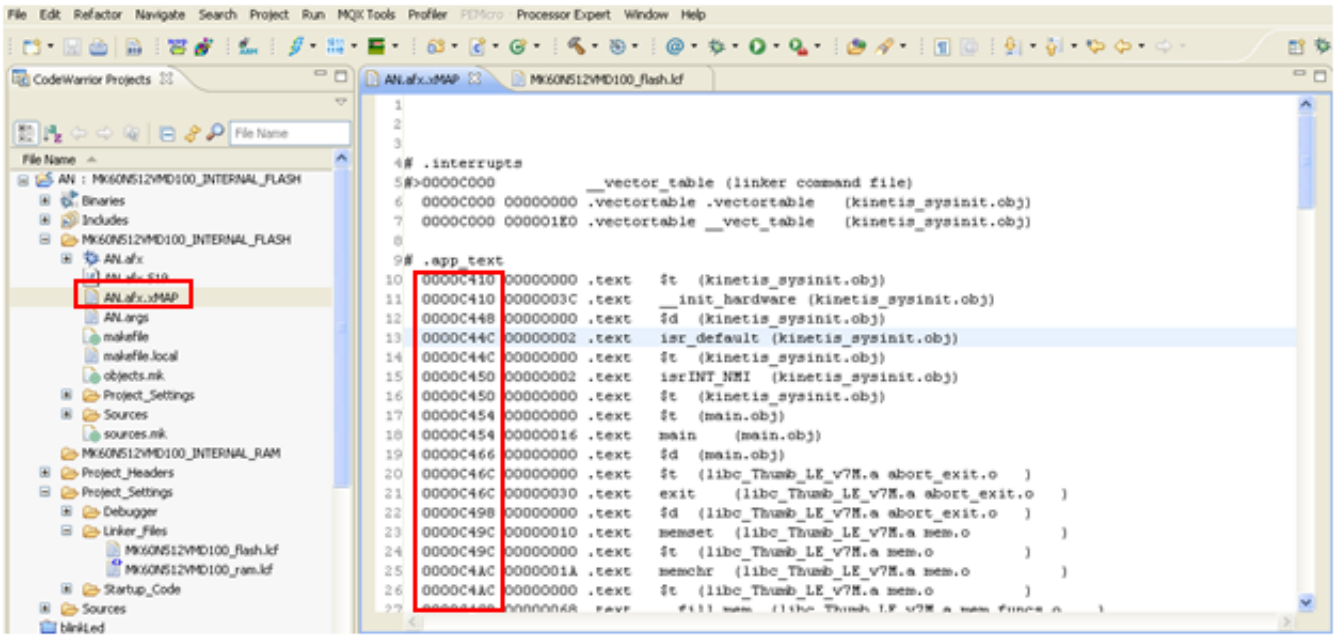


Figure 9. Memory map output file after linker command file is modified in CW.

Now the generated.S19 file is ready to be downloaded to your MCU through the Ethernet bootloader. You can find .S19 file as shown in the following figure. This file is needed in section 4.

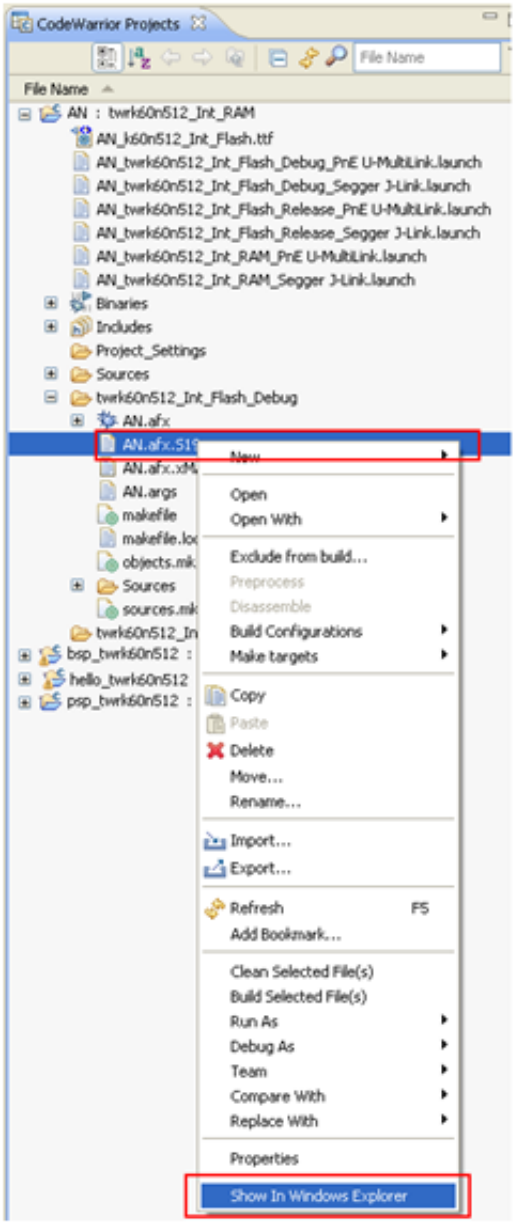


Figure 10. Location of the .S19 file.

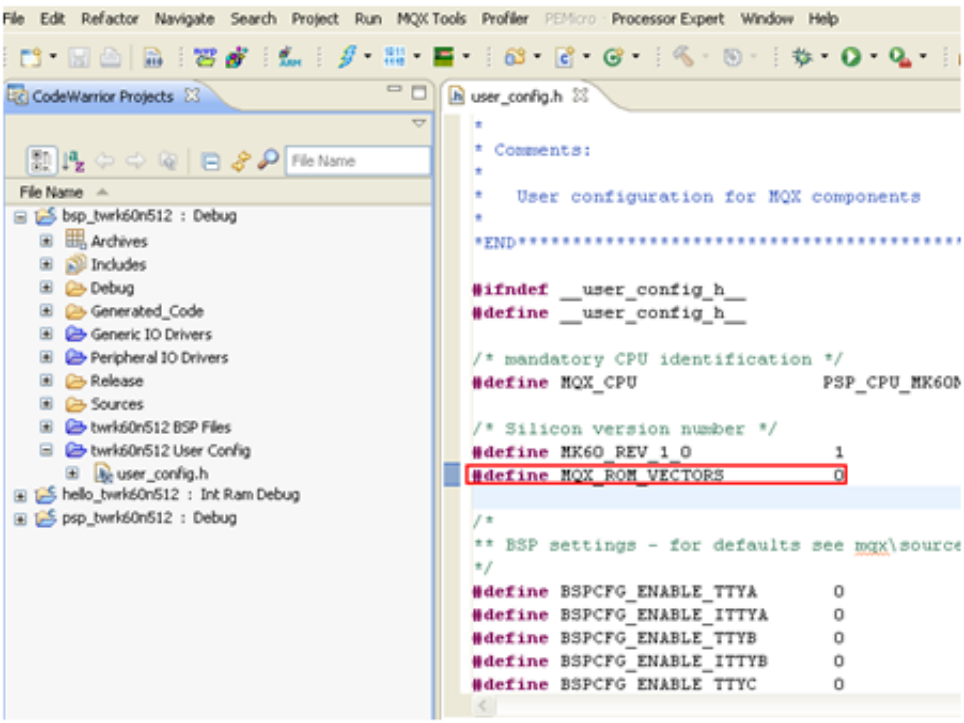
3.3 Developing an MQX application in CodeWarrior for MCU 10.1 with Ethernet bootloader support

This scenario uses a “Hello World” project that can be found at C:\Program Files\Freescale\Freescale MQX 3.7\mqx \examples\hello\cw10\hello_twrk60n512.

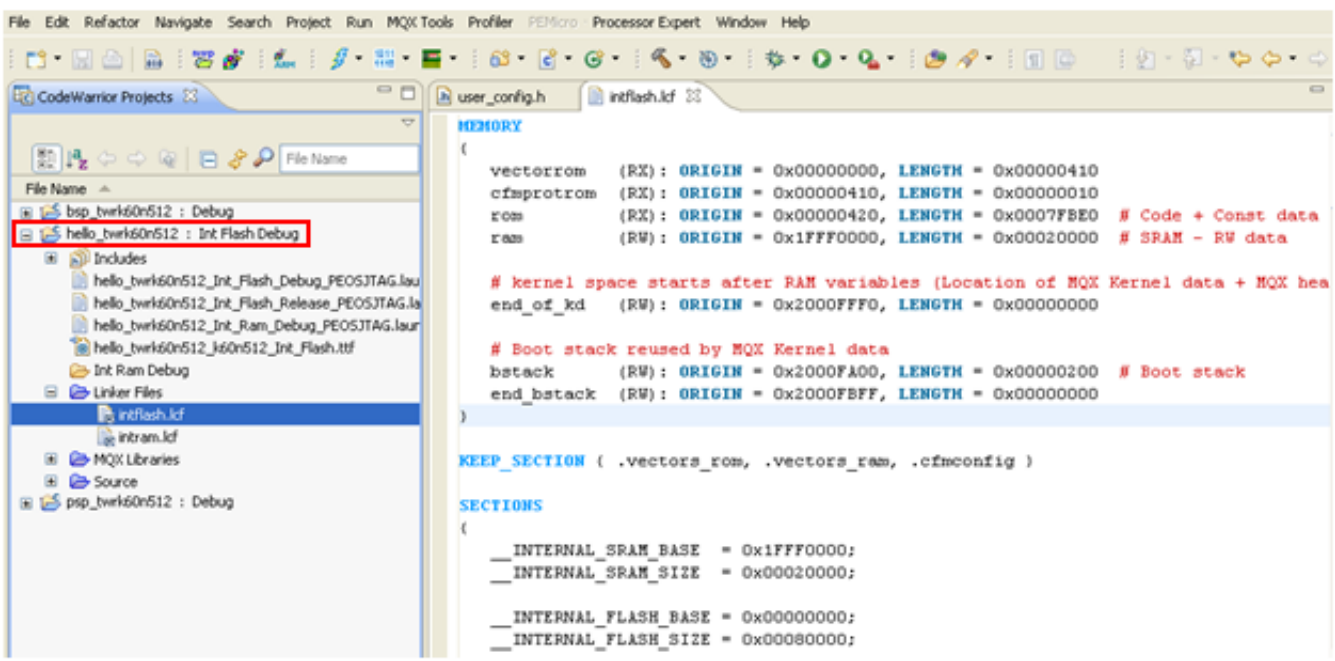
1. Drag and drop .project file of the hello_twrk60n512 project into the CodeWarrior Projects window.
2. By default, MQX allocates the vector table at the first address of the flash memory, 0x0000-0000. Because this region of flash memory is used by the Ethernet bootloader, it will be necessary to reallocate the vector table in a RAM section. To reallocate, open the libraries (bsp_twrk60n512 psp_twrk60n512), add the following macro into user_config.h, and recompile both libraries. You can find more information about this procedure in the section 2.4 Freescale CodeWarrior

Development Studio version 10.1 from the “Getting Started” document C:\Program Files\Freescale\Freescale MQX 3.7\doc\FSL_MQX_Getting_Started.pdf.

```
#define MQX_ROM_VECTORS           0
```



3. Change the target to MK60N512VMD100_INTERNAL_FLASH, and open MK60N512VMD100_flash.lcf



4. Modify the .lcf file
   a. Modify the start address in flash of your application from 0x0000_0000 to 0x0000_C000, including ROM Vector Table.

```
# 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. Remove the m_cfmprtrom region, because you will not use this area. Because it is reserved for flash protection registers, the bootloader will handle this section. Comment the m_cfmprotrom section as shown below.

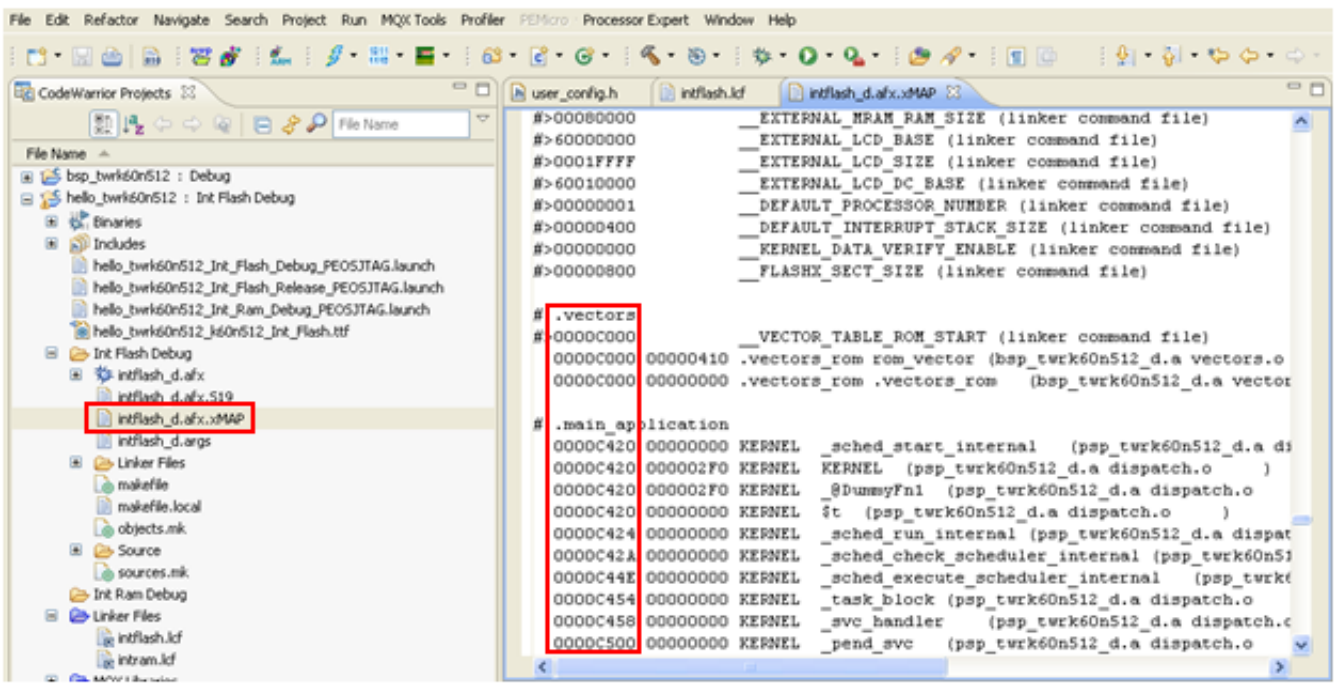
```
# m_cfmprotrom  (RX) : ORIGIN = 0x00000400, LENGTH = 0x00000010

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

c. Create a flash section for the bootloader parameters.

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

An entire modified .lcf file is shown in Appendix C.

5. Read the .xmap file to check whether the application code is located in the expected flash address.



Now the generated .afx.S19 file is ready to be downloaded to your MCU through the Ethernet bootloader. You can find the .S19 file in this path C:\Program Files\Freescale\Freescale MQX 3.7\mqx\examples\hello \cw10\hello_twrk60n512\Int Flash Debug. This file is needed for the next section.

4 Running Ethernet bootloader

For the purpose of this document the following tools and hardware were used:

- IAR Systems Embedded Workbench 6.0.3
- Terminal program (Windows Hyperterminal)
- TFTP Server and Client (Tftpd version 3.33 http://tftpd32.jounin.net/)
- Ethernet bootloader source code, coming with this document
- TWR-K60N512-KIT

The following steps show you how to run the Ethernet bootloader in the MK60N512:

1. Open the Ethernet bootloader project by drag and drop the twrk60n512_boot.ewp file located in C:\Program Files \Freescale\FNET 1.0.0\fnet_demos\mk60n512\boot\iararm6.1 into IAR IDE.
2. Compile using FLASH_512KB_PFLASH target and program the device.

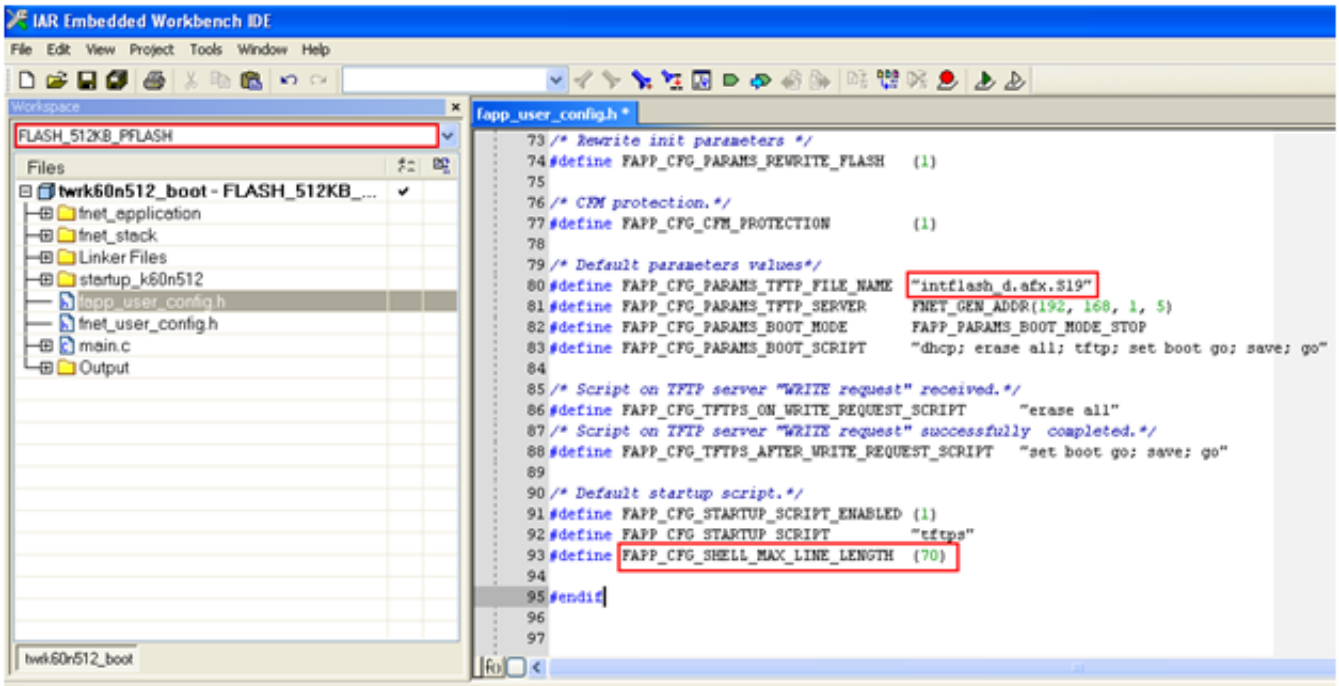
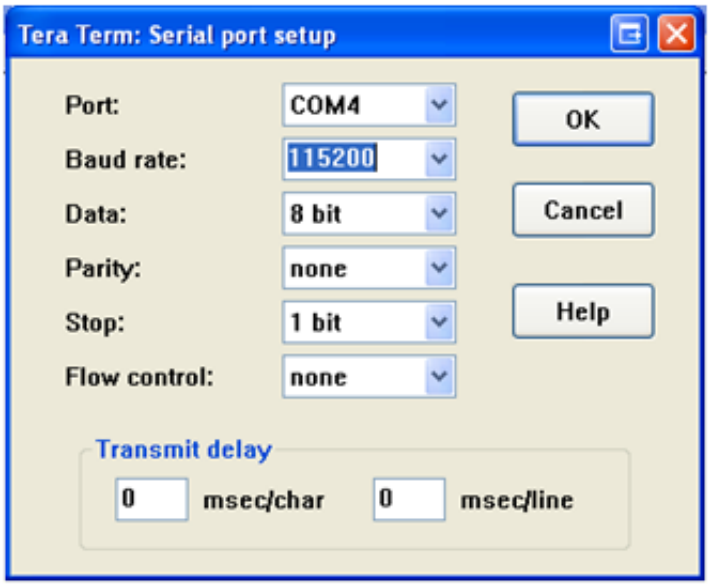


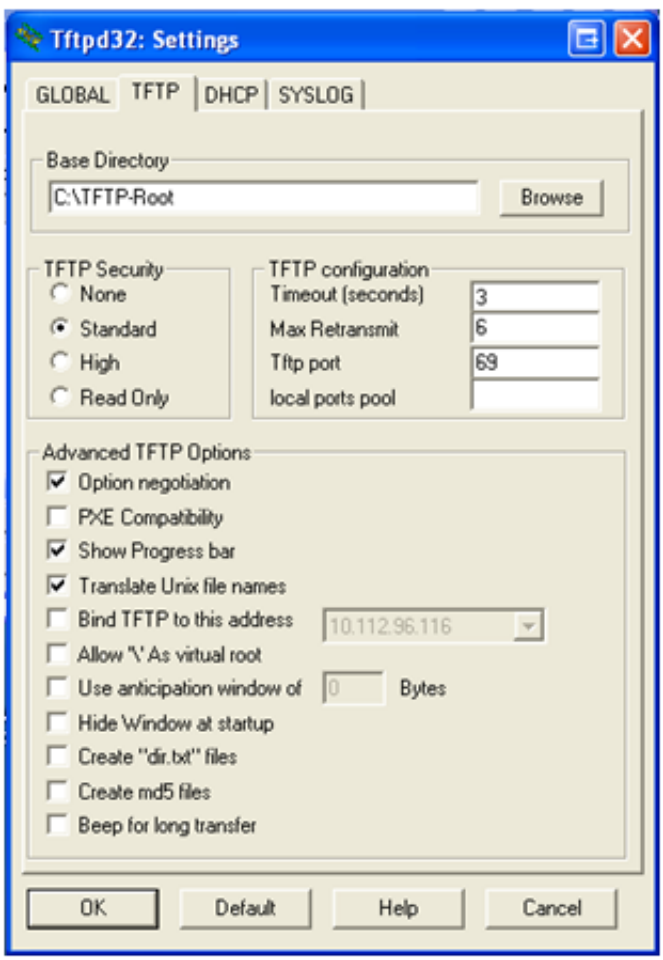
Figure 11. Default bootloader configurations.

Note: You can change the default parameters of the Ethernet bootloader, such as the name of the s-rec file (.s19 or .srec), the maximum length of string shell, and the scripts. Perform these changes by modifying fapp_user_config.h. The following figure shows the values used in this document.

3. Prepare your TWR-K60N512-KIT according to the TWR-K60N512 User's Manual.
4. Use the RS-232 male/female DB-9 serial cable to connect your PC (through COM port) to the tower board, using the TWR-SER.
5. Connect the tower board to an LAN network.
6. Invoke the terminal program on the PC to which the tower board is connected and configure it to:
   a. Bits per second: 115200.
   b. Data bits: 8.
   c. Parity: none.
   d. Stop bits: 1.
   e. Flow control: None.



7. Start TFTP Server:
   a. Start TFTP Server application. Go to [Start]->[Programs]->[Tftpd32]->[Tftpd32]
   b. Press the [Settings] button and set the TFTP server base directory (for example C:\TFTP-Root):



8. Power off and power on back your board. You can also push the SW5 to reset the board.

9. In your terminal program window you should see the Ethernet bootloader welcome message as shown below:

```
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. The Ethernet bootloader is now ready to accept user commands. Enter the “help” command to display a list of all commands available within the shell of the bootloader:

```
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>
```

Type “dhcp” to get an IP address. In case you don’t have a DHCP client the MCU will have the default IP address that can be modified in the fnet_user_config.h file in the next line:

```
#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. Type "**set tfpt <IP address>**" to set tfpt firmware server. In this case, use 10.112.96.116.

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

12. Type "**set script 'dhcp\; erase all\; tftp\; set boot go\; save\; reset**'" to change the script parameters.

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

13. When all parameters are set correctly for your system, set the "**boot**" parameter to the script value. To do this type "**set boot script**" to set the script value.

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

14. Enter the "**save**" command to save all run-time parameters to a non-volatile memory (it is located in the last erase page of the on-chip flash memory).
15. Type "**reset"**.
16. In your terminal program window you should see the Ethernet bootloader with the scrip already loaded and ready to run the user application:

```
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. If any key was pressed within 5 seconds after reset, the Ethernet bootloader will execute the script programmed before.

```
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. At this point, the application resides in the flash area, and if any key is pressed after 5 seconds the application will run until a reset is executed.

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

5 Conclusion

The Ethernet bootloader includes 3 different scenarios. The explained options are:

- Bareboard application in IAR with Ethernet bootloader support.
- Bareboard application in CodeWarrior for MCU 10.1 with Ethernet bootloader support.
- MQX application in CodeWarrior for MCU 10.1 with Ethernet bootloader support.

Each option is explained in detail, providing the developer the needed steps to embed the Ethernet bootloader in any product. This variety of implementations offers the developer the possibility to choose the best option for their applications.

The usage guide of the bootloader is also provided. This gives the users specific information in order to use the Ethernet bootloader.

This application note helps the developers to include Ethernet bootloaders in their products, using a quick and easy approach.

6 Appendix A: IAR linker configuration file (.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 Appendix B: Linker command file (.lcf) in CodeWarrior bareboard project

```
# 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 Appendix C. Linker command file (.lcf) in CodeWarrior MQX project

```
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:
www.freescale.com

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

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

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

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

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

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

Document Number: AN4367
Rev. 0, 09/2011

freescale semiconductor