

Building a Flash Embedded Application in an i.MX System

by: Florent Auger

1 Abstract

This application note is for users who program embedded applications on an i.MX ADS system (MC9328MX1/MC9328MXL).

Most programs contain variables that change during program execution. In addition, most applications usually operate from ROM. This leaves the programmer with the question of where in RAM to transfer initialized variables and other created variables during program execution. The free location for new variables and the exact location address of global variables in RAM must be known during the link procedure of the program. This means that a complete software description of the memory location must be known before flashing a program. To accomplish this, a scatter and init file must be created to enable a program to move from a ROM-only execution to a ROM + RAM or full RAM execution. Figure 1 illustrates this hardware organization.

The user will not find every scenario in this application note to build and install an embedded program.

Contents

1 Abstract	1
2 Overview	3
3 Method 1: With stack.s and heap.s	3
4 Method 2: Without stack.s and heap.s	10
5 Where are the Differences?	15
6 Others Files to Run the Example	16
7 How to Proceed in CodeWarrior	16
8 Appendices	16



Abstract

However, users will find basic and sufficient examples to create many others. Flashing procedures will not be described, as it depends on the Flash memory used.

The intent of this document is to provide a reference program that operates from flash memory, and which uses the interrupts of the RTC module to make 2 LED's of an i.MX Application Development System (i.MX ADS) board blink. This application note describes the hardware and software requirements.

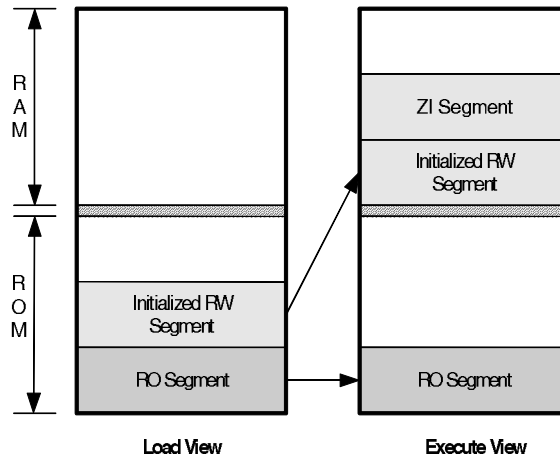


Figure 1. A Scattered Memory Map

1.1 Related Documents

[1] ADS_DeveloperGuide_D, ARM Ltd. (ARM DUI 0056D)

[2] ADS_LinkerGuide_A, ARM Ltd. (ARM DUI 0151A)

[3] ADS_AssemblerGuide_B, ARM Ltd. (ARM DUI 0068B)

[4] MC9328MX1 Integrated Portable System Processor Reference Manual, (order number MC9328MX1RM/D)

[5] MC9328MXL Integrated Portable System Processor Reference Manual, (order number MC9328MXLRM/D)

1.2 Abbreviations Used in This Document

i.MX ADS	Application Development System for either MC9328MX1ADS or MC9328MXLADS
CW	Metrowerks CodeWarrior
ELF	Executable and linking format
Heap	The portion of computer memory that can be used for creating new variables.
IRQ	Interruption request
Stack	The portion of memory that is used to record the return address of code that calls a subroutine. The stack can also be used for parameters and temporary variables.
ZI	Zero Initialized. R/W memory used to hold variables that do not have an initial value. The memory is normally set to zero on reset.

2 Overview

The following sections provide the user with the hardware examples for the given scenario. Two different software methods describe how to build the embedded application.

Users may copy and paste the files contained in this text and the files in the appendices for either of the two methods to easily reproduce the example. However, hardware requirements must be respected.

The two methods are virtually the same, the differences are in the way to use and set the values for some of the needed parameters.

2.1 Hardware

Freescale recommends using the i.MX ADS board for the i.MX processor. However, any proprietary hardware design can be used for the purposes outlined in this document if it is equipped with its own memory.

A Multi-ICE[®] or a serial cable is required to transfer the code to be flashed, and the flash program. If only a serial cable is used, a flashing utility can be found in the BSP provided for the i.MX ADS boards at: <http://www.freescale.com/imx>.

Typical Hardware required for the purposes outlined in this document are:

- i.MX ADS board
- Multi-ICE or serial cable

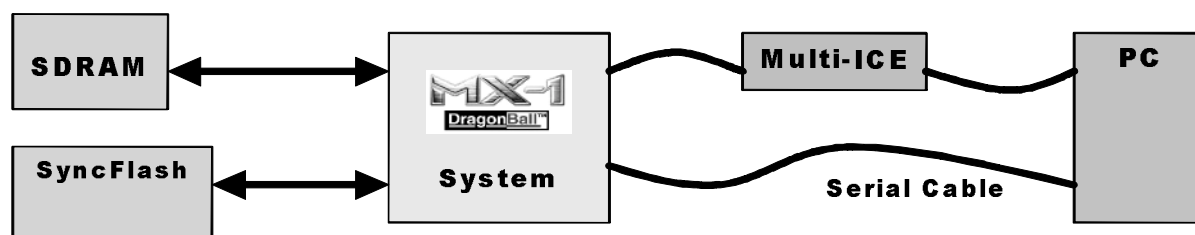


Figure 2. Typical System Development

2.2 Software

This section describes what files the user needs, and what the files must contain. The Freescale's reference design i.MX ADS will be used as the example for all the following methods.

2.3 Development Environment

Either the ARM[®] Developer Suite[™] ver. 1.2 or the Metrowerks CodeWarrior[™] ver. 1.2 for the i.MX processor is required. With either development system, use a debugger through a Multi-ICE connection to transfer the code and download the Flash program. As stated previously, a serial cable can do the same.

3 Method 1: With stack.s and heap.s

In method 1, all the memory characteristics of the program are specified in the scatter description file.

3.1 Scatter File

This is the recommended method for placing regions at required locations in the memory map.

The scatter file is important because it specifies to the linker the exact memory location of the following regions:

- ROM contains all the invariable code and constants
- RAM includes Zero Initialized region, RW region, and more particularly heap and stack areas:
 - ZI region is used to store the zero initialized variables.
 - The heap is used for creating dynamic, global, and static variables.
 - The stack stores local variables, parameters passing into functions, return addresses from subroutines, and so on.

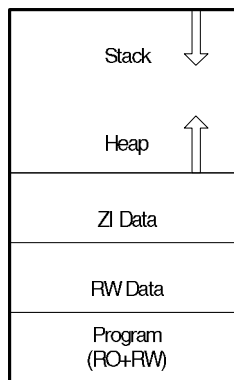


Figure 3. Scatter File Flow Diagram

Given the goal to have an embedded application, there is the need to have two types of memories:

1. One to store the ROM code—that is, Flash. The ROM code contains an interruption vector table, initialization of RAM memory, and the remainder of the program.
2. One to store RAM part—that is, SDRAM or SRAM, which contains stack and heap areas.

The ROM region is mapped at the base address of Flash for the i.MX ADS board version 2.0 which is 0x1000 0000, and the base address 0x0800 0000 is used for the SDRAM to store the RAM part.

The scatter file is shown in Example 1.

Example 1. Scatter File

```
ROM_LOAD 0x0C000000
{
    ROM_EXEC 0x0C000000
    {
        vector.o (Vect, +First)
        * (+RO)
    }
    RAM 0x08000000
}
```

```

        * (+RW,+ZI)
    }

    HEAP +0 UNINIT
    {
        heap.o (+ZI)
    }
    STACK 0x088FFFC0 UNINIT
    {
        stack.o (+ZI)
    }
}

```

NOTE

The start address for ROM, RAM, HEAP, and STACK must be changed to suit the design needs—that is, to handle large variables or large amounts of data or to optimize the memory allocation. The most convenient location for heap is usually just after the RW+ZI region. The top of stack should be placed far enough from the bottom of the heap to avoid overlap of these two regions.

In this application, Vector.o is the file that contains the vector's table. This is a key file for an application running IRQ, and should be located at the beginning of a region where it can jump the CPU. For the i.MX processor, it should be located at the address 0x0, which can be remapped to another relative address. Please, refer to the i.MX reference manuals (See Section 1.1, "Related Documents" on page 2).

Heap.s and Stack.s files described in the next section are used in the scatter description file, which associates an address in the RAM to the symbols created in these files.

3.2 Heap.s File

This file allows the user to create a symbol to define the start address of the heap region. This address is called *bottom_of_heap*. The scatter file uses this file to set the value for the heap base address. For example, directly after the RW, ZI regions. The linker automatically sets this value, as it should know the size of the RAM region.

Example 2. Heap.s

```

;;; Copyright ARM Ltd 2001. All rights reserved.
AREA    Heap, DATA, NOINIT

        EXPORT bottom_of_heap

; Create dummy variable used to locate bottom of heap

bottom_of_heap    SPACE    1

        END

```

3.3 Stack.s File

This file allows the user to create a symbol to define the top address of the stack region. This address is called `top_of_stacks`. The scatter file is used to set the value for the stack top address, for example, `0x088FFFC0`. This file is needed by the `init.s` file to initiate the `top_of_stacks` variable.

Example 3. Stack.s

```
;;; Copyright ARM Ltd 2001. All rights reserved.

        AREA    Stacks, DATA, NOINIT

        EXPORT top_of_stacks

; Create dummy variable used to locate stacks in memory

top_of_stacks    SPACE    1

        END
```

3.4 Init.s File

The `init.s` file is one of the most important files to run an application on an ARM core based processor such as the i.MX processor. The example 6.4 on page 6-17 of the `ADS_Developer_Guide` is the base of the `init.s` file that is used.

Because the program will be embedded, the user must add the assembly command to this reference file to initialize the SDRAM. Indeed, before beginning, the RAM areas must be ready. DPLL and different clocks can also be initialized there. The remaining code is the same as the example used.

Given this, the file used in most of the templates related to the i.MX processor is shown in Example 4.

Example 4. Init.s

; On reset, the ARM core starts up in Supervisor (SVC) mode, in ARM state, with IRQ and FIQ disabled.

```
        AREA    Init, CODE, READONLY

; --- Standard definitions of mode bits and interrupt (I & F) flags in PSRs

Mode_USR            EQU        0x10
Mode_FIQ            EQU        0x11
Mode_IRQ            EQU        0x12
Mode_SVC            EQU        0x13
Mode_ABT            EQU        0x17
Mode_UND            EQU        0x1B
Mode_SYS            EQU        0x1F ; available on ARM Arch 4 and later

I_Bit                EQU        0x80 ; when I bit is set, IRQ is disabled
F_Bit                EQU        0x40 ; when F bit is set, FIQ is disabled
```

```

; --- Amount of memory (in bytes) allocated for stacks

Len_FIQ_Stack          EQU      256
Len_IRQ_Stack          EQU      256
Len_ABT_Stack          EQU      256
Len_UND_Stack          EQU      256
Len_SVC_Stack          EQU      8192
; Len_USR_Stack          EQU      16384

; Add lengths >0 for FIQ_Stack, ABT_Stack, UND_Stack if you need them.
; Offsets will be loaded as immediate values.
; Offsets must be 8 byte aligned.

Offset_FIQ_Stack       EQU      0
Offset_IRQ_Stack       EQU      Offset_FIQ_Stack + Len_FIQ_Stack
Offset_ABT_Stack       EQU      Offset_IRQ_Stack + Len_IRQ_Stack
Offset_UND_Stack       EQU      Offset_ABT_Stack + Len_ABT_Stack
Offset_SVC_Stack       EQU      Offset_UND_Stack + Len_UND_Stack
; Offset_USR_Stack       EQU      Offset_SVC_Stack + Len_SVC_Stack

                IMPORT          top_of_stacks

                ENTRY

                EXPORT  Reset_Handler

Reset_Handler

; --- Initialize stack pointer registers
                ; Enter each mode in turn and set up the stack pointer

                LDR      r0, =top_of_stacks

; Enter FIQ mode and set up the FIQ stack pointer
                MSR      CPSR_c, #Mode_FIQ:OR:I_Bit:OR:F_Bit ; No interrupts
                SUB      sp, r0, #Offset_FIQ_Stack

; Enter IRQ mode and set up the IRQ stack pointer
                MSR      CPSR_c, #Mode_IRQ:OR:I_Bit:OR:F_Bit ; No interrupts
                SUB      sp, r0, #Offset_IRQ_Stack

                MSR      CPSR_c, #Mode_ABT:OR:I_Bit:OR:F_Bit ; No interrupts
                SUB      sp, r0, #Offset_ABT_Stack

                MSR      CPSR_c, #Mode_UND:OR:I_Bit:OR:F_Bit ; No interrupts
                SUB      sp, r0, #Offset_UND_Stack

```

Method 1: With stack.s and heap.s

```
; Enter SVC mode and set up the SVC stack pointer
    MSR    CPSR_c, #Mode_SVC:OR:I_Bit:OR:F_Bit ; No interrupts
    SUB    sp, r0, #Offset_SVC_Stack

; -- SDRAM controller constants.

SDRAMC_BA            EQU            0x221000
SDRAMC_BA            EQU            0x8000000

SDRAMC_CMDPRE       EQU            0x92120200
SDRAMC_PREREA       EQU            0x08200000

SDRAMC_AUTORE       EQU            0xA2120200
SDRAMC_AUTREA       EQU            0x08000000

SDRAMC_MDEREG       EQU            0xB2120200
SDRAMC_MDEREA       EQU            0x08111800

SDRAMC_NRMLMO       EQU            0x82124200

; --- Initialize SDRAM memory

    LDR    r0, =SDRAMC_CMDPRE
    LDR    r1, =SDRAMC_BA
    STR    r0, [r1, #0]
    MOV    r0, #0x1
    MOV    r1, #0x8200000
    STR    r0, [r1, #0]
    LDR    r0, =SDRAMC_AUTORE
    LDR    r1, =SDRAMC_BA
    STR    r0, [r1, #0]
    MOV    r0, #0x1
    LDR    r1, =SDRAMC_BA
    STR    r0, [r1, #0]
    STR    r0, [r1, #0]
    STR    r0, [r1, #0]
    STR    r0, [r1, #0]
    STR    r0, [r1, #0]
    STR    r0, [r1, #0]
    STR    r0, [r1, #0]
    STR    r0, [r1, #0]
    LDR    r0, =SDRAMC_MDEREG
    LDR    r1, =SDRAMC_BA
    STR    r0, [r1, #0]
```



```

MOV            r0,#0x1
LDR            r1,=SDRAMC_MDEREA
STR            r0,[r1,#0]
LDR            r0,=SDRAMC_NRMLMO
LDR            r1,=SDRAMC_BA
STR            r0,[r1,#0]

; --- Initialize PLL & Clocks

                ;Restart MCU_PLL,PRES=1,BCLK=1,
CSCR_VAL      EQU            0x2F200403
CSCR_ADD      EQU            0x21B000
MPCTL0_VAL    EQU            0x04632410
MPCTL0_ADD    EQU            0x21B004

                ;Program MCU PLL at 150Mhz
LDR            r0,=MPCTL0_VAL
LDR            r1,=MPCTL0_ADD
STR            r0,[r1,#0]

                ;HCLK = 48Mhz for SDRAM
LDR            r0,=CSCR_VAL
LDR            r1,=CSCR_ADD
STR            r0,[r1,#0]

                ;Set asynchronous mode
MRC            p15,0,r0,c1,c0,0
MOV            r2, #0xC0000000
ORR            r0,r2,r0
MCR            p15,0,r0,c1,c0,0

IMPORT    __main

; --- Now enter the C code
B    __main    ; note use B not BL, because an application will never return this way

END

```

3.5 Retarget.c File

This file is mainly used in such application to set heap base, heap limit, stack limit, stack base address. The heap base address is imported from heap.s file. The structure used to set all these values is shown in Example 5.

Example 5. Retarget.c

```
extern unsigned int bottom_of_heap;
__value_in_regs struct __initial_stackheap __user_initial_stackheap(
    unsigned R0, unsigned SP, unsigned R2, unsigned SL)
{
    struct __initial_stackheap config;

    ; // defined in heap.s and placed by scatterfile
    config.heap_base = (unsigned int)&bottom_of_heap

    config.heap_limit = (unsigned int)&bottom_of_heap +
    0x100000;
    config.stack_limit = SP - 9214;
    config.stack_base = SP;    // inherit SP from the
    execution environment

    return config;
}
```

NOTE

The fields config.heap_limit and config.stack_limit are not mandatory, there are specified here to give an example. However, they avoid overlap between these two regions.

4 Method 2: Without stack.s and heap.s

The alternate method to create an embedded application does not require the creation of heap.s and stack.s files. The locations of the stack and heap areas are described in the retarget.c file rather than the scatter file.

4.1 Scatter File

This is the recommended method for placing regions at required locations in the memory map.

- ROM contains all the invariable code and constants.
- RAM includes Zero Initialized region, RW region, and more particularly heap and stack areas:
 - ZI region is used to store the zero initialized variables.
 - The heap is used for creating dynamic, global, and static variables.
 - The stack stores local variables, parameters passing into functions, return addresses from subroutines, and so on.

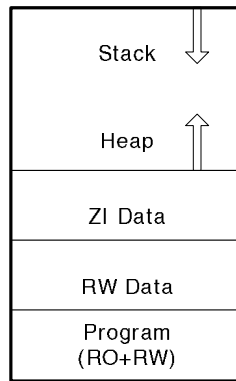


Figure 4. Scatter File Flow Diagram

Given the goal to have an embedded application two types of memories are required:

1. One to store the ROM code—that is, Flash.
2. One to store RAM part—that is, SDRAM or SRAM.

Therefore, the ROM region will be mapped at the base address 0x1000 0000, and the base address 0x0800 0000 is used for the SDRAM to store the RAM part.

The scatter file is shown in Example 6.

Example 6. Scatter File

```
ROM_LOAD 0x0C000000
{
    ROM_EXEC 0x0C000000
    {
        vector.o (Vect, +First)
        * (+RO)
    }
    RAM 0x08000000
{
    * (+RW,+ZI)
}
}
```

NOTE

The start address for ROM and RAM must be changed to suit the design needs—that is, to handle large variables or large amounts of data or to optimize the memory allocation.

In this application, Vector.o is the file that contains the vector table. This is a key file for an application running IRQ, and should be located at the beginning of a region where it can jump the CPU. For the i.MX processor, it should be located at the address 0x0, which can be remapped to another relative address. Please, refer to the i.MX processor reference manual.

4.2 Init.s File

The init.s file is one of the most important files to run an application on an ARM core based processor such as the i.MX processor. The example 6.4 on page 6-17 of the ADS_Developer_Guide is the base of the init.s file that is used.

Because the program will be embedded, the user must add the assembly command that initializes the SDRAM to the init.s reference file. Indeed, before beginning the RAM areas must be ready. DPLL and different clocks can also be initialized there. The rest of the code is the same as the example used.

Given this, the file used in most of the templates related to the i.MX processor is shown in Example 7.

Example 7. Init.s File

; On reset, the ARM core starts up in Supervisor (SVC) mode, in ARM state, with IRQ and FIQ disabled.

```

        AREA    Init, CODE, READONLY

; --- Standard definitions of mode bits and interrupt (I & F) flags in PSRs

Mode_USR            EQU    0x10
Mode_FIQ            EQU    0x11
Mode_IRQ            EQU    0x12
Mode_SVC            EQU    0x13
Mode_ABT            EQU    0x17
Mode_UNDEF          EQU    0x1B
Mode_SYS            EQU    0x1F        ;available on ARM Arch 4 and later

I_Bit               EQU    0x80        ; when I bit is set, IRQ is disabled
F_Bit               EQU    0x40        ; when F bit is set, FIQ is disabled

; --- System memory locations

RAM_Limit           EQU    0x08F00000    ; For unexpanded ARM Integrator board
SVC_Stack           EQU    RAM_Limit    ; 4096 byte SVC stack at top of memory
USR_Stack           EQU    SVC_Stack-4096 ; followed by IRQ stack
IRQ_Stack           EQU    USR_Stack-4096 ; followed by SVC stack
FIQ_Stack           EQU    IRQ_Stack-4096 ; followed by USR stack

ROM_Start           EQU    0x00000000    ; Base address of ROM after remapping
Instruct_2          EQU    ROM_Start + 4 ; Address of second instruction in ROM

        ENTRY

        EXPORT  Reset_Handler

Reset_Handler

; --- Initialise stack pointer registers

```

```

; Enter SVC mode and set up the SVC stack pointer
MSR    CPSR_c, #Mode_SVC:OR:I_Bit:OR:F_Bit ; No interrupts
LDR    SP, =SVC_Stack

; Enter IRQ mode and set up the IRQ stack pointer
MSR    CPSR_c, #Mode_IRQ:OR:I_Bit:OR:F_Bit ; No interrupts
LDR    SP, =IRQ_Stack

; Enter FIQ mode and set up the FIQ stack pointer
MSR    CPSR_c, #Mode_FIQ:OR:I_Bit:OR:F_Bit ; No interrupts
LDR    SP, =FIQ_Stack

; -- SDRAM controller constants.

SDRAMC_BA            EQU            0x221000
SDRAMC_BA            EQU            0x8000000

SDRAMC_CMDPRE        EQU            0x92120200
SDRAMC_PREREA        EQU            0x08200000

SDRAMC_AUTORE        EQU            0xA2120200
SDRAMC_AUTREA        EQU            0x08000000

SDRAMC_MDEREG        EQU            0xB2120200
SDRAMC_MDEREA        EQU            0x08111800

SDRAMC_NRMLMO        EQU            0x82124200

; --- Initialize SDRAM memory

LDR    r0, =SDRAMC_CMDPRE
LDR    r1, =SDRAMC_BA
STR    r0, [r1, #0]
MOV    r0, #0x1
MOV    r1, #0x8200000
STR    r0, [r1, #0]
LDR    r0, =SDRAMC_AUTORE
LDR    r1, =SDRAMC_BA
STR    r0, [r1, #0]
MOV    r0, #0x1
LDR    r1, =SDRAMC_BA
STR    r0, [r1, #0]
STR    r0, [r1, #0]
STR    r0, [r1, #0]
STR    r0, [r1, #0]
STR    r0, [r1, #0]

```

Method 2: Without stack.s and heap.s

```
        STR            r0, [r1, #0]
        STR            r0, [r1, #0]
        STR            r0, [r1, #0]
        LDR            r0, =SDRAMC_MDEREG
        LDR            r1, =SDRAMC_BA
        STR            r0, [r1, #0]
        MOV            r0, #0x1
        LDR            r1, =SDRAMC_MDEREA
        STR            r0, [r1, #0]
        LDR            r0, =SDRAMC_NRMLMO
        LDR            r1, =SDRAMC_BA
        STR            r0, [r1, #0]

; --- Initialize PLL & Clocks
        ;Restart MCU_PLL, PRESC=1, BCLK=1,
CSCR_VAL EQU          0x2F200403
CSCR_ADD EQU          0x21B000
MPCTL0_VAL EQU        0x04632410
MPCTL0_ADD EQU        0x21B004

        ;Program MCU PLL at 150Mhz
        LDR            r0, =MPCTL0_VAL
        LDR            r1, =MPCTL0_ADD
        STR            r0, [r1, #0]

        ;HCLK = 48Mhz for SDRAM
        LDR            r0, =CSCR_VAL
        LDR            r1, =CSCR_ADD
        STR            r0, [r1, #0]

        ;Set asynchronous mode
        MRC            p15, 0, r0, c1, c0, 0
        MOV            r2, #0xC0000000
        ORR            r0, r2, r0
        MCR            p15, 0, r0, c1, c0, 0

; Finally, Re-Enter SVC mode
        MSR            CPSR_c, #Mode_SVC:OR:I_Bit:OR:F_Bit ; No interrupts
        LDR            SP, =SVC_Stack

        IMPORT    __main

; --- Now enter the C code
        B            __main ; note use B not BL, because an application will never return this way
        END
```

NOTE

The `RAM_limit` value corresponds to the location of the top of the stack. This value must be set manually by the user. However, it is automatically set in Method 1.

4.3 Retarget.c File

This file is primarily used to set heap base, heap limit, stack limit, and stack base address. The stack and heap base addresses are mandatory, however both limit addresses are not. With this method, this file does not call any function or variable from `stack.s` or `heap.s` files. The structure used to set these values is shown in Code Listing 8.

Example 8. Retarget.c

```
__value_in_regs struct __initial_stackheap __user_initial_stackheap(
    unsigned R0, unsigned SP, unsigned R2, unsigned SL)
{
    struct __initial_stackheap config;

    config.heap_base = 0x08100000;
                                config.stack_base = SP;    // inherit SP from the execution
environment

    return config;
}
```

5 Where are the Differences?

Method 1, using `heap.s` and `stack.s`, is the more efficient method:

- The association of both the scatter file and the `bottom_of_heap` symbol defined in `heap.s` allows the heap base to be set automatically in `retarget.c`.
- The association of both the scatter file and the `top_of_stack` symbol defined in `stack.s` allows the stack base address to be set automatically in `init.s`.

This method is more convenient because the user is only required to make efficient use of the scatter file and is not required to change anything in the `init.s` or `retarget.c` files. All values, to match the specific design, are set in the scatter file.

Method 2, without `heap.s` and `stack.s`, is a basic alternative:

- The heap base address is set manually in `retarget.c`, referred to as `config.heap_base`.
- The stack base address is set manually in the `init.s`, referred to as `RAM_Limit`.

This method requires the user to modify and manage three files (`init.s`, `retarget.c`, and scatter file) as opposed to managing one file as in method 1.

6 Others Files to Run the Example

The following files are also used in the example and are available at the end of the document.

- Main.c
- Handler.c
- RTC_routines.c
- Memory_Map.h
- Vector.s
- Macro.c
- Led.c
- Prototype.h

7 How to Proceed in CodeWarrior

When the code has been written, the user will compile, link, and post link it to obtain a binary file and will store this file onto the Flash targeted memory.

The procedure to achieve this is as follows:

1. Project Setting: Target -> Target Settings window -> Post-Linker: select ARM from ELF.
2. Go to Linker -> ARM from ELF -> Output format: select Plain Binary. Set an output file name for the binary file, project_name.bin for example.
3. **make** the project to generate the binary file and program it to the Flash with an appropriate Flash program.

8 Appendices

8.1 Main.c

Example 9. Main.c

```

/*****
C   M O D U L E   F I L E
(c) Copyright Freescale Semiconductors Toulouse Limited 2001-2002
ALL RIGHTS RESERVED
*****/

Project Name : An Embedded Application
File Name    : Main.c
Description  : This file describes the main.
*****/

#include <stdio.h>
#include "Common.h"
#include "Memory_Map.h"
#include "Prototype.h"

U32 gLed_Status,gcount,hourmin_val_m,sec_val_m,min_val_m,hour_val_m;

void main()
{

```



```
gLed_Status = ON;
    EnableIntSource(17);
    EnableIRQ();

    Rtc_softreset();
    Rtc_disable();
    Rtc_setup();
    Rtc_enable();

    while(1);
}
```

Figure 5 provides the Flow charts of the main.c program.

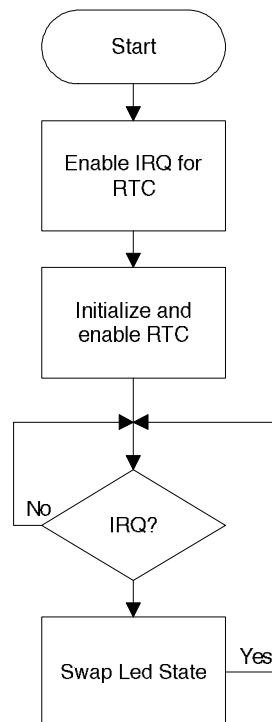


Figure 5. main.c Flow Diagram

8.2 Vectors

Example 10. Vector.s

```

;;; Copyright ARM Ltd 1999. All rights reserved.

        AREA Vect, CODE, READONLY

; These are example exception vectors and exception handlers

; Where there is ROM fixed at 0x0 (build_b & build_c), these are hard-coded at 0x0.
; Where ROM/RAM remapping occurs (build_d), these are copied from ROM to RAM.
; The copying is done automatically by the C library code inside __main.

; *****
; Exception Vectors
; *****

; Note: LDR PC instructions are used here because branch (B) instructions
; could not simply be copied (the branch offsets would be wrong). Also,
; a branch instruction might not reach if the ROM is at an address >32MB).

        ENTRY

        LDR    PC, Reset_Addr
        LDR    PC, Undefined_Addr
        LDR    PC, SWI_Addr
        LDR    PC, Prefetch_Addr
        LDR    PC, Abort_Addr
        NOP                                ; Reserved vector
        LDR    PC, IRQ_Addr
        LDR    PC, FIQ_Addr

        IMPORT Reset_Handler             ; In init.s

Reset_Addr          DCD    Reset_Handler
Undefined_Addr      DCD    Undefined_Handler
SWI_Addr            DCD    SWI_Handler
Prefetch_Addr      DCD    Prefetch_Handler
Abort_Addr          DCD    Abort_Handler
DCD    0             ; Reserved vector
IRQ_Addr            DCD    IRQ_Handler
FIQ_Addr            DCD    FIQ_Handler

; *****
; Exception Handlers
; *****

```

```

IMPORT      C_Undefined_Handler
IMPORT      C_SWI_Handler
IMPORT      C_Prefetch_Handler
IMPORT      C_Abort_Handler
IMPORT      C_IRQ_Handler
IMPORT      C_FIQ_Handler

```

; The following dummy handlers do not do anything useful in this example.
; They are set up here for completeness.

```

Undefined_Handler
    B      C_Undefined_Handler
SWI_Handler
    B      C_SWI_Handler
Prefetch_Handler
    B      C_Prefetch_Handler
Abort_Handler
    B      C_Abort_Handler
IRQ_Handler
    B      C_IRQ_Handler
FIQ_Handler
    B      C_FIQ_Handler

END

```

8.3 Handler.c

Example 11. Handler.c

```

/*****
C  M O D U L E   F I L E
(c) Copyright Freescale Semiconductors Toulouse Limited 2001-2002
ALL RIGHTS RESERVED
*****/
Project Name : An Embedded Application
File Name    : Handler.c
Description  : This file describes the different handlers used by the iMX1/L.
*****/
#include <stdio.h>
#include "Common.h"
#include "Memory_Map.h"
#include "Prototype.h"

```



```

switch( swi_num )
{
    case 0:
        break;
        default:
            break;
}
}

void C_Undefined_Handler(void)
{
    //read in ITC status reg
    //test bit for dispatch
    //call user ISR
    return;
}

void C_Prefetch_Handler(void)
{
    //read in ITC status reg
    //test bit for dispatch
    //call user ISR
    return;
}

void C_Abort_Handler(void)
{
    //read in ITC status reg
    //test bit for dispatch
    //call user ISR
    return;
}

/*****
* IRQ_Handler *
* This function handles IRQ interrupts. *
*****/
void __irq C_IRQ_Handler(void)
{
    short vectNum;
    vectNum = NIVECSR >> 16; // determine highest pending normal interrupt
    vect_IRQ[vectNum](); // find the pointer to correct ISR in the look up
table
}

```

Appendices

```

/*****
 * FIQ_Handler
 * This function handles IRQ interrupts.
 *****/
void __irq C_FIQ_Handler(void)
{
    short vectNum;
    vectNum = FIVECSR & 0x0000003F;          // determine highest pending fast
interrupt
    vect_FIQ[vectNum]();                    // find the pointer to correct ISR in the look up
table
}

/*****
 *
 * Interrupt Handlers (64 fast int sources and 64 normal int sources)
 *
 *****/
void fast_dummy_isr(void){}
void norm_dummy_isr(void){}

extern U32 gLed_Status;

void RTC_scr17_isr(void)
{
    if((* (P_U32)RTC_RTCISR & 0x10) == 0x10)
    {
        *(P_U32)RTC_RTCISR = 0x10;
        if(gLed_Status == ON)
        {
            LED2_off();
            LED3_on();
            gLed_Status = OFF;
        }
        else
        {
            LED2_on();
            LED3_off();
            gLed_Status = ON;
        }
    }
}
}

```

8.4 Macro.c

Example 12. Macro.c

```

/*****
C   M O D U L E   F I L E
(c) Copyright Freescale Semiconductors Toulouse Limited 2001-2002
ALL RIGHTS RESERVED
*****/

Project Name : An Embedded Application
File Name    : Macros.c
Description  : This file has all the macros and inline function for iMX1/L.
*****/

#include <stdio.h>
#include "Common.h"
#include "Memory_Map.h"
#include "Prototype.h"

// IRQ handling function for the ARM core.
// Enable IRQ
void EnableIRQ (void)
{
    __asm
    {
        MRS r1, CPSR
        BIC r1, r1, #0x80
        MSR CPSR_c, r1
    }
}

// Disable IRQ
void DisableIRQ (void)
{
    __asm
    {
        MRS r1, CPSR
        ORR r1, r1, #0x80
        MSR CPSR_c, r1
    }
}

// Macro to set the Interrupt Enable Number Register
void EnableIntSource (U32 IntNum)
{
    IntNum &= 0x0000003F;
}

```

```

        *(P_U32)AIRC_INTENNUM = (U32)IntNum;
    }

```

8.5 RTC_Routines.c

Example 13. RTC_Routines.c

```

/*****
C   M O D U L E   F I L E
(c) Copyright Freescale Semiconductors Toulouse Limited 2001-2002
ALL RIGHTS RESERVED
*****/
Project Name : An Embedded Application
File Name    : RTC_Routines.c
Description  : Basic functions to program the Real Time Clock Module
*****/
#include <stdio.h>
#include "Memory_Map.h"
#include "Prototype.h"

extern  U32          hourmin_val_m;
extern  U32          sec_val_m;
extern  U32          min_val_m;
extern  U32          hour_val_m;

/*****
void Rtc_softreset()
{
    *((P_U32)RTC_RTCCTL) = 0x01;
}
*****/
void Rtc_enable()
{
    *((P_U32)RTC_RTCCTL) |=0x80;
}
/*****
void Rtc_disable()
{
    *((P_U32)RTC_RTCCTL) &=0xfffffff7f;
}
*****/
void Rtc_setup()
{

```



```

        //set prescaler = 01 -> means 32Hz
* ((P_U32) RTC_RTCCTL) |= 0x20;                                //set bit 5

        //Clear all the bit of the rtc status register
* ((P_U32) RTC_RTCISR) = 0xFFFF;

        //Enable the 1Hz interrupt
* ((P_U32) RTC_RTCIENR) = 0x10;

* (P_U32) RTC_HOURMIN = 0x0;
* (P_U32) RTC_SECOND = 0x0;
}

```

8.6 Led.c

Example 14. Led.c

```

/*****
C  M O D U L E   F I L E
(c) Copyright Freescale Semiconductors Toulouse Limited 2001-2002
ALL RIGHTS RESERVED
*****/
Project Name : An Embedded Application
File Name    : Led.c
Description  : Basic functions to switch the LED state
*****/
#include "Memory_Map.h"
#include "Prototype.h"

void LED2_on(void)
{
    *(P_U32) PTA_GIUS |= 0x4;
    *(P_U32) PTA_OCR1 |= 0x30;
    *(P_U32) PTA_PUEN &= 0xfffffff;
    *(P_U32) PTA_DDIR |= 0x4;
    *(P_U32) PTA_DR  |= 0x4;
}

void LED2_off(void)
{
    *(P_U32) PTA_GIUS |= 0x4;
    *(P_U32) PTA_OCR1 |= 0x30;
    *(P_U32) PTA_PUEN &= 0xfffffff;
    *(P_U32) PTA_DDIR |= 0x4;
}

```

Appendices

```
        *(P_U32) PTA_DR &= 0xffffffffb;
    }

void LED3_on(void)
{
    *(P_U32) PTA_GIUS |= 0x800000;
    *(P_U32) PTA_OCR2 |= 0xc000;
    *(P_U32) PTA_PUEN &= 0xff7fffff;
    *(P_U32) PTA_DDIR |= 0x800000;
    *(P_U32) PTA_DR |= 0x800000;
}

void LED3_off(void)
{
    *(P_U32) PTA_GIUS |= 0x800000;
    *(P_U32) PTA_OCR1 |= 0xc000;
    *(P_U32) PTA_PUEN &= 0xff7fffff;
    *(P_U32) PTA_DDIR |= 0x4;
    *(P_U32) PTA_DR &= 0xff7fffff;
}
```

8.7 Memory_Map.h

Example 15. Memory_Map.h

```
/*
*****
C H E A D E R F I L E
(c) Copyright Freescale Semiconductors Toulouse Limited 2001-2002
ALL RIGHTS RESERVED
*****
Project Name : An Embedded Application
File Name : Memory_Map.h
Description : Definition of needed Memory Map address for iMX processor
*****/

// ;-----;
// ; AITC ;
// ; $0022_3000 to $0022_3FFF ;
// ;-----;

#define AITC_BASE_ADDR 0x00223000
#define AITC_INTCNTL (AITC_BASE_ADDR+0x00) //Interrupt Control Register
#define AITC_NIMASK (AITC_BASE_ADDR+0x04) //Normal Interrupt Mask Register
#define AITC_INTENNUM (AITC_BASE_ADDR+0x08) //Interrupt Enable Number Register
```

```

#define AITC_INTDISNUM          (AITC_BASE_ADDR+0x0C)//Interrupt Disable Number Register
#define AITC_INTENABLEH        (AITC_BASE_ADDR+0x10) //Interrupt Enable Register High
#define AITC_INTENABLEL        (AITC_BASE_ADDR+0x14) //Interrupt Enable Register Low
#define AITC_INTTYPEH          (AITC_BASE_ADDR+0x18)
#define AITC_INTTYPEL          (AITC_BASE_ADDR+0x1C)
#define AITC_NIPRIORITY7       (AITC_BASE_ADDR+0x20)
#define AITC_NIPRIORITY6       (AITC_BASE_ADDR+0x24)
#define AITC_NIPRIORITY5       (AITC_BASE_ADDR+0x28)
#define AITC_NIPRIORITY4       (AITC_BASE_ADDR+0x2C)
#define AITC_NIPRIORITY3       (AITC_BASE_ADDR+0x30)
#define AITC_NIPRIORITY2       (AITC_BASE_ADDR+0x34)
#define AITC_NIPRIORITY1       (AITC_BASE_ADDR+0x38)
#define AITC_NIPRIORITY0       (AITC_BASE_ADDR+0x3C)
#define AITC_NIVECSR           (AITC_BASE_ADDR+0x40)
#define AITC_FIVECSR           (AITC_BASE_ADDR+0x44)
#define AITC_INTSRCH           (AITC_BASE_ADDR+0x48)
#define AITC_INTSRCL           (AITC_BASE_ADDR+0x4C)
#define AITC_INTFRCH           (AITC_BASE_ADDR+0x50)
#define AITC_INTFRCL           (AITC_BASE_ADDR+0x54)
#define AITC_NIPNDH            (AITC_BASE_ADDR+0x58)
#define AITC_NIPNDL            (AITC_BASE_ADDR+0x5C)
#define AITC_FIPNDH            (AITC_BASE_ADDR+0x60)
#define AITC_FIPNDL            (AITC_BASE_ADDR+0x64)

// ;-----;
// ; GPIO - PTA                      ;
// ; $0021_C000 to $0021_C0FF        ;
// ;-----;
#define PTA_BASE_ADDR          0x0021C000
#define PTA_DDIR                PTA_BASE_ADDR
#define PTA_OCR1                (PTA_BASE_ADDR+0x04)
#define PTA_OCR2                (PTA_BASE_ADDR+0x08)
#define PTA_ICONFA1             (PTA_BASE_ADDR+0x0C)
#define PTA_ICONFA2             (PTA_BASE_ADDR+0x10)
#define PTA_ICONFB1             (PTA_BASE_ADDR+0x14)
#define PTA_ICONFB2             (PTA_BASE_ADDR+0x18)
#define PTA_DR                  (PTA_BASE_ADDR+0x1C)
#define PTA_GIUS                (PTA_BASE_ADDR+0x20)
#define PTA_SSR                 (PTA_BASE_ADDR+0x24)
#define PTA_ICR1                (PTA_BASE_ADDR+0x28)
#define PTA_ICR2                (PTA_BASE_ADDR+0x2C)
#define PTA_IMR                 (PTA_BASE_ADDR+0x30)
#define PTA_ISR                 (PTA_BASE_ADDR+0x34)

```

Appendices

```
#define PTA_GPR                (PTA_BASE_ADDR+0x38)
#define PTA_SWR                (PTA_BASE_ADDR+0x3C)
#define PTA_PUEN              (PTA_BASE_ADDR+0x40)

// ;-----;
// ; RTC                        ;
// ; $0020_4000 to $0020_4FFF    ;
// ;-----;
#define RTC_BASE_ADDR          0x00204000
#define RTC_HOURMIN            RTC_BASE_ADDR
#define RTC_SECOND             (RTC_BASE_ADDR+0x04)
#define RTC_ALARM_HM          (RTC_BASE_ADDR+0x08)
#define RTC_ALARM_SEC         (RTC_BASE_ADDR+0x0C)
#define RTC_RTCCTL            (RTC_BASE_ADDR+0x10)
#define RTC_RTCISR            (RTC_BASE_ADDR+0x14)
#define RTC_RTCIENR           (RTC_BASE_ADDR+0x18)
#define RTC_STPWCH            (RTC_BASE_ADDR+0x1C)
#define RTC_DAYR              (RTC_BASE_ADDR+0x20)
#define RTC_DAYALARM          (RTC_BASE_ADDR+0x24)
#define RTC_TEST1             (RTC_BASE_ADDR+0x28)
#define RTC_TEST2             (RTC_BASE_ADDR+0x2C)
#define RTC_TEST3             (RTC_BASE_ADDR+0x30)
```

8.8 Prototype.h

Example 16. Prototype.h

```
/*-----*/
C H E A D E R   F I L E
(c) Copyright Freescale Semiconductors Toulouse Limited 2001-2002
ALL RIGHTS RESERVED
/*-----*/
Project Name : An Embedded Application
File Name    : ProtoType.h
Description  : ProtoType definition header for iMX processor
/*-----*/
#ifndef Proto_INC
#define Proto_INC
/*-----*/
/*-----*/ Typedefs for integer types /*-----*/
typedef unsigned char   U8;    /* unsigned 8 bit data */
typedef unsigned short  U16;   /* unsigned 16 bit data */
typedef unsigned int    U32;   /* unsigned 32 bit data */
```

```

typedef char      S8;          /* signed 8 bit data */
typedef short    S16;         /* signed 16 bit data */
typedef int      S32;         /* signed 32 bit data */

typedef U8 *     P_U8;        /* unsigned 8 bit data */
typedef U16 *    P_U16;       /* unsigned 16 bit data */
typedef U32 *    P_U32;       /* unsigned 32 bit data */
typedef S8 *     P_S8;        /* signed 8 bit data */
typedef S16 *    P_S16;       /* signed 16 bit data */
typedef S32 *    P_S32;       /* signed 32 bit data */

/***** Constants *****/
#define ON 1
#define OFF 0

/***** Function prototype (S) *****/
void EnableIRQ (void);
void DisableIRQ (void);
void EnableFIQ (void);
void DisableFIQ (void);
void MaskIRQLevel(U32 MaskLevel);
void EnableIntSource(U32 IntNum);
void DisableIntSource(U32 IntNum);
void SetIntType(U32 IntNum, U8 type);
void SetIrqPriority(U8 IntNum, U8 level);
void ForceOneInt(U32 IntNum);
void ForceAllInt(void);

void SysInit(void);
void LED2_on(void);
void LED2_off(void);
void LED3_on(void);
void LED3_off(void);

void Rtc_softreset(void);
void Rtc_enable(void);
void Rtc_disable(void);
void Rtc_setup(void);

/* Interrupt Service Routines function prototypes */
void fast_dummy_isr(void);
void norm_dummy_isr(void);

void RTC_scr17_isr(void);

#endif

```

How to Reach Us:

USA/Europe/Locations Not Listed:

Freescale Semiconductor Literature Distribution Center
P.O. Box 5405
Denver, Colorado 80217
1-800-521-6274 or 480-768-2130

Japan:

Freescale Semiconductor Japan Ltd.
Technical Information Center
3-20-1, Minami-Azabu, Minato-ku
Tokyo 106-8573, Japan
81-3-3440-3569

Asia/Pacific:

Freescale Semiconductor Hong Kong Ltd.
2 Dai King Street
Tai Po Industrial Estate
Tai Po, N.T., Hong Kong
852-26668334

Home Page:

www.freescale.com

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.

Learn More: For more information about Freescale products, please visit www.freescale.com.

Freescale™ and the Freescale logo are trademarks of Freescale Semiconductor, Inc. ARM, the ARM POWERED logo, and Multi-ICE are the registered trademarks of ARM Ltd. ARM Developer Suite is a trademark of ARM Ltd. All other product or service names are the property of their respective owners.

© Freescale Semiconductor, Inc. 2004. All rights reserved.