Freescale Semiconductor
Application Note

Document Number:AN4552
Rev. 0, 10/2012

# Porting Your Code to the uVision Environment

by: Chris Brown

Contents

1 Introduction....1
2 Overview of the KEIL Compiler....1
3 Common uVision Errors and Pitfalls....2
4 Creating a new Keil project from the Kinetis Sample Code....2
5 File Modifications....13
6 Checking your newly compiled project....26
7 Conclusion....26

## 1 Introduction

This application note is written for those who wish to use the Kinetis family sample code (KINETIS512_SC, located at http://www.freescale.com) in the Keil uVision environment. It covers some general pitfalls that the Freescale Kinetis application team has discovered when trying to port to Keil and outlines one step-by-step example of porting a project from the Kinetis sample code to the Keil environment (targeted for the TWR-K60N512).

The current uVision release at the time of writing this application note is v4.23. The uVision IDE, offered by KEIL, is strictly ANSI C99 and CMSIS compliant. This can in turn cause compatibility issues with Freescale's sample code (or your code) as it is not CMSIS compliant.

## 2 Overview of the KEIL Compiler

KEIL uVision is an integrated development environment that combines project management, make facilities, source code editing, program debugging, and complete simulation in one powerful environment. KEIL uVision is a tool developed by ARM and has been supporting Freescale Kinetis products

© 2012 Freescale Semiconductor, Inc.

since 2010. Because KEIL has close ties to ARM, KEIL is completely CMSIS compliant. More information about uVision can be found at http://www.keil.com/uvision/.

# 3 Common uVision Errors and Pitfalls

The following is a list of common errors that have been encountered by this group in the course of creating uVision projects.

- uVision strictly adheres to the C99 (ISO/IEC 9899) standard. Therefore, main must return a parameter of type int. In the current sample code base, main has a void return. uVision will not allow this and the definition of main must be changed to return an integer value.
- Keil does support inline assembly code. Setting up this function in uVision is difficult and not recommended. Many of the inline assembly functions throughout the sample code base must be changed to the CMSIS compliant intrinsic functions defined on Keil's website.
- If a variable or function is declared as an extern, it must be explicitly defined elsewhere in the code. If it is not, the uVision compiler will not be able to find the variable and errors will occur during the compilation.
- uVision by default does not allow anonymous union declarations, like those used in the current device specific header files. To enable the use of anonymous unions, you must use the instruction #pragma anon_unions to indicate to uVision that you will be using anonymous union declarations.
- The __main function of uVision initializes the stack and heap, as well as initializing variables. While you can call functions in the *.s file before calling __main, you cannot have a function that jumps to another function (the stack is not setup). Also, you cannot depend on variables being set to a value in a function that was called before __main, because __main may clear this variable.

# 4 Creating a new Keil project from the Kinetis Sample Code

Currently, no Keil projects exist in the Kinetis sample code package. Therefore, you will need to create a folder, named Keil, under the KINETIS512_SC\build folder. Next, you will need to create subfolders for your projects (the location for these folders will be KINETIS512_SC\build\Keil\<project name>). After you have manually created your new project folder(s) in the location specified above, follow the step by step instructions to create your new uVision project.

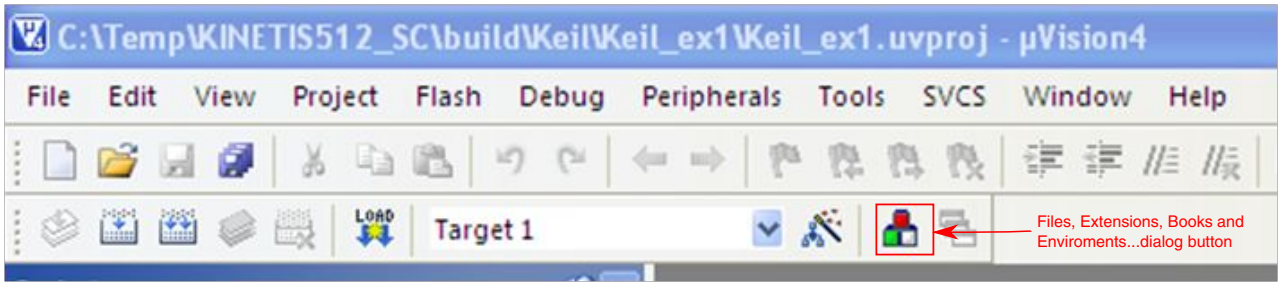
1. Navigate to Project->New uVision Project….
2. Point the dialog box to the project folder you just created and save your new project there.
3. Select the appropriate Freescale MCU from the Generic CPU database (MK60N512VMD100 for this example). Do not be concerned that you can only select one part. There will be a chance to add other parts (or change the current part) later.

When prompted to copy the recommended startup_<device>.s file to the Project Folder and add the file to the project, click no. We will add our own *.s file to the project later.

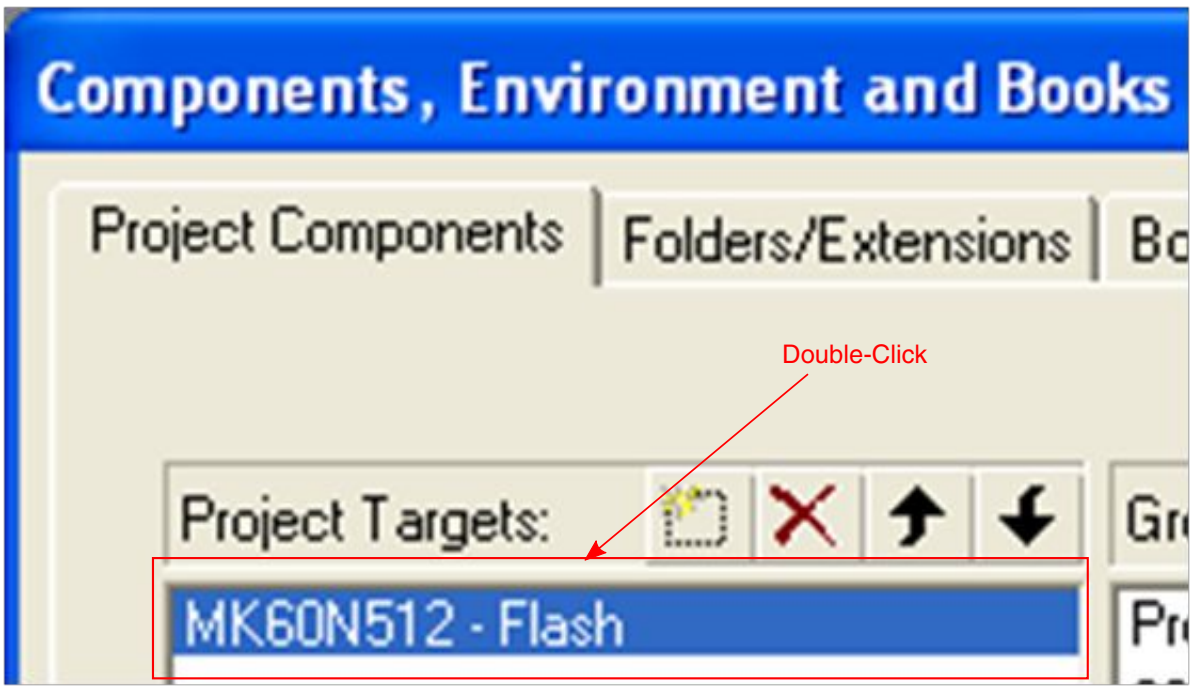
## 4.1 Creating new targets and groups

Now you must add the project files to your project. But first, correctly setup the groups to organize your workspace. Follow these instructions to setup the workspace.

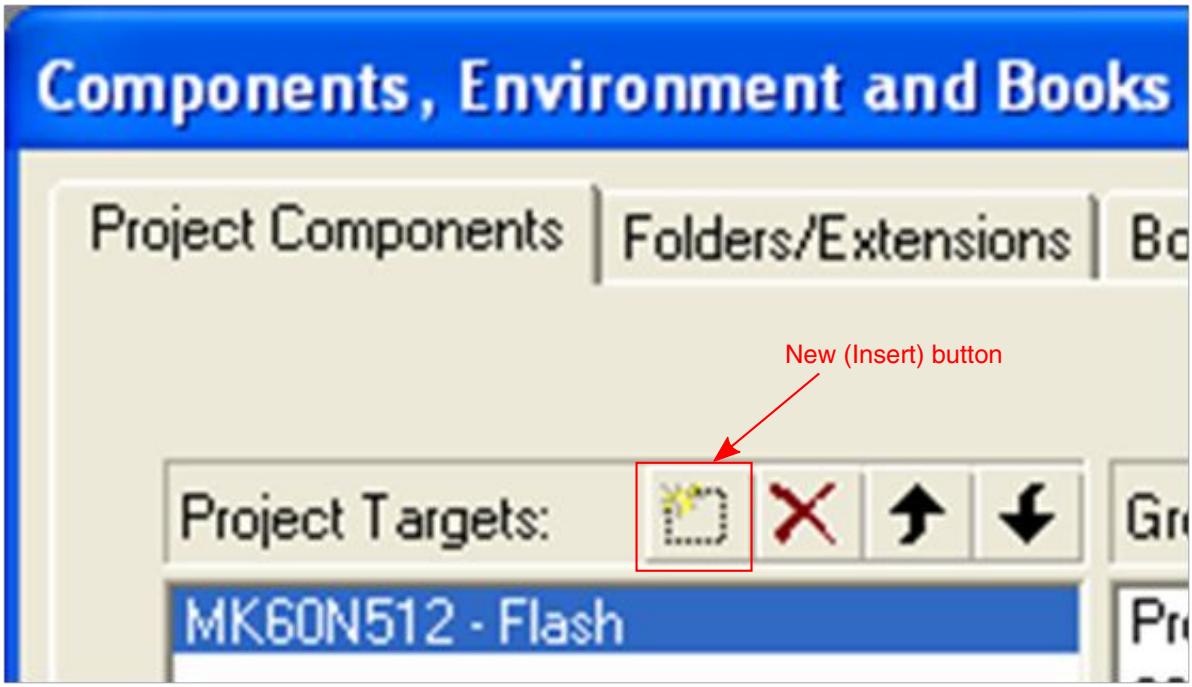
1. Click the Files, Extensions, Books, and Environments… dialog button.




2. A dialog box opens. Here you can add or modify targets and groups, as well as add or remove files to or from the groups. For convenience and organization, rename Target 1 to MK60N512 – Flash. Do this by double-clicking Target 1.

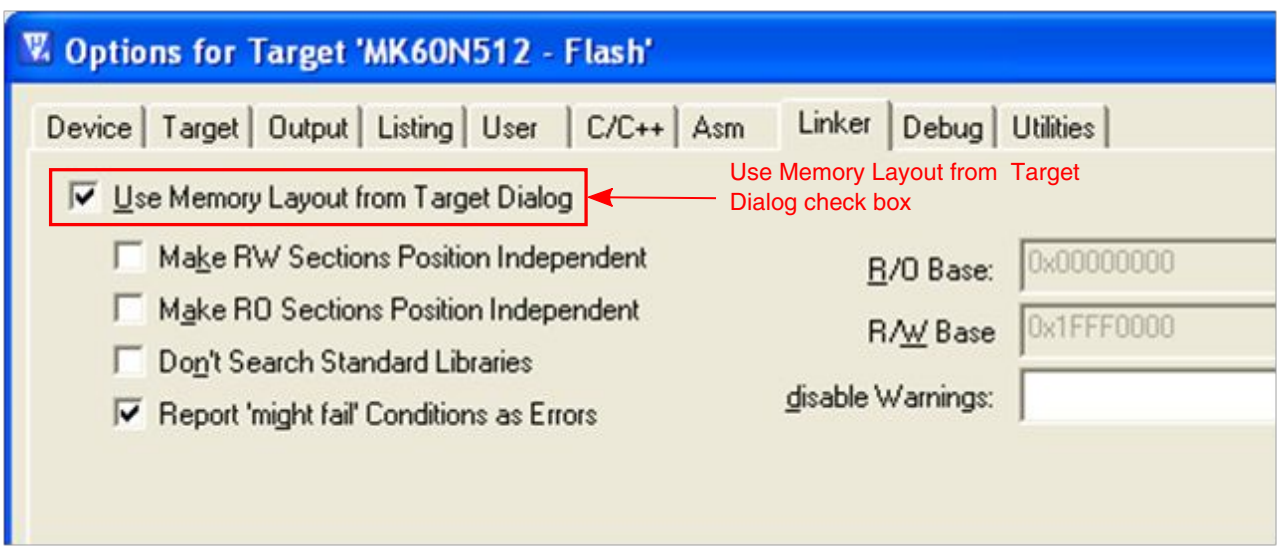



3. Rename Source Group 1 to Project, the same way that Target 1 was renamed. Before adding files to the groups, add all of the appropriate groups. The groups added are as follows:
   - common
   - drivers
   - cpu
   - platforms
4. Create a new project target (named MK60N512 – RAM) by clicking the "New (Insert)" button and name the newly created target appropriately. After the new target is created, it is necessary to perform the target setup actions described in Memory configuration for a RAM target for this target.
5. Click Ok to exit the Components, Environment and Books dialog box.

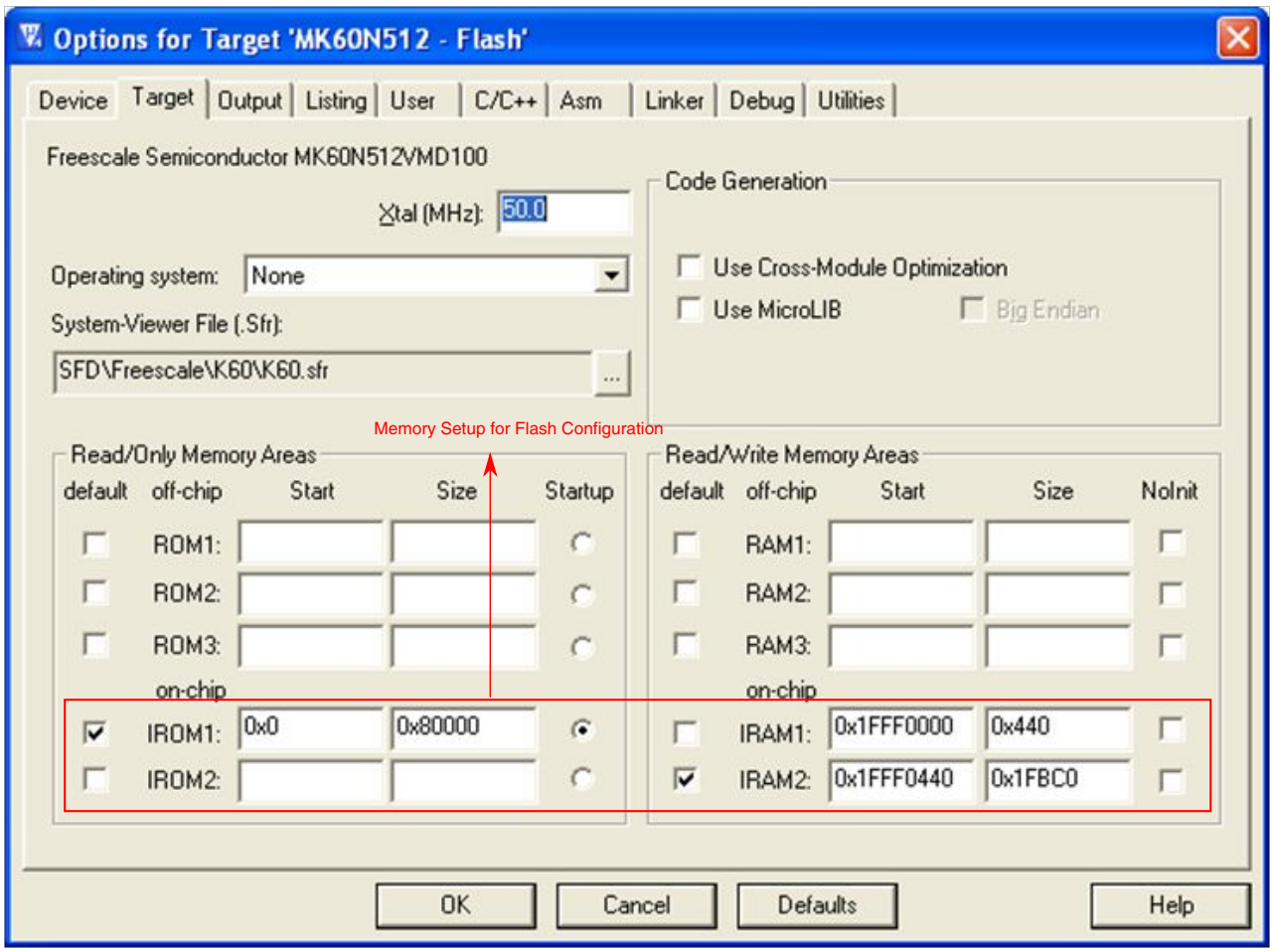



## 4.2 Memory configuration for a Flash target

1. Navigate to the Linker tab of the Target Options dialog box
2. Select Use Memory Layout from Target Dialog check box.



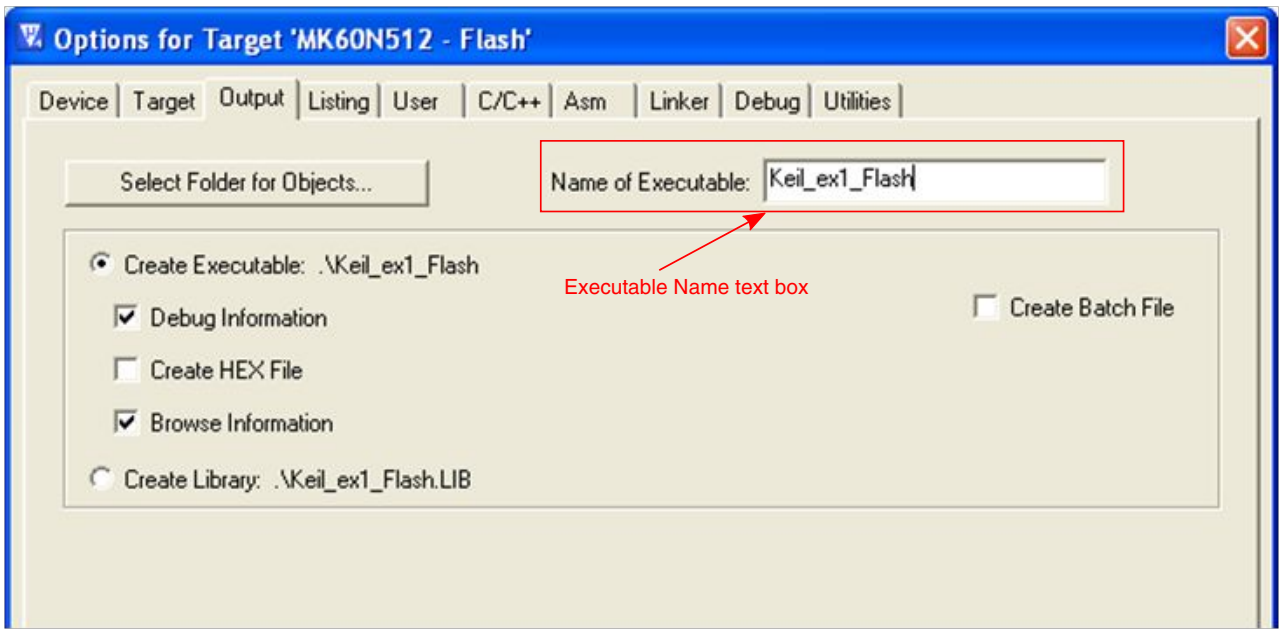

3. Select the Target tab. Ensure that the setup is the desired setup. For a Flash target, the flash memory addresses must be located in the IROM1 text boxes and the RAM memory locations must be entered in the IRAM1 and IRAM2 text boxes as displayed below.




**NOTE**

This memory configuration does not divide the RAM memory into two equal sections as described in the reference manual. If you need to maintain this structure or just need a third RAM section in general, consult the section on creating your own scatter loading file in this application note.

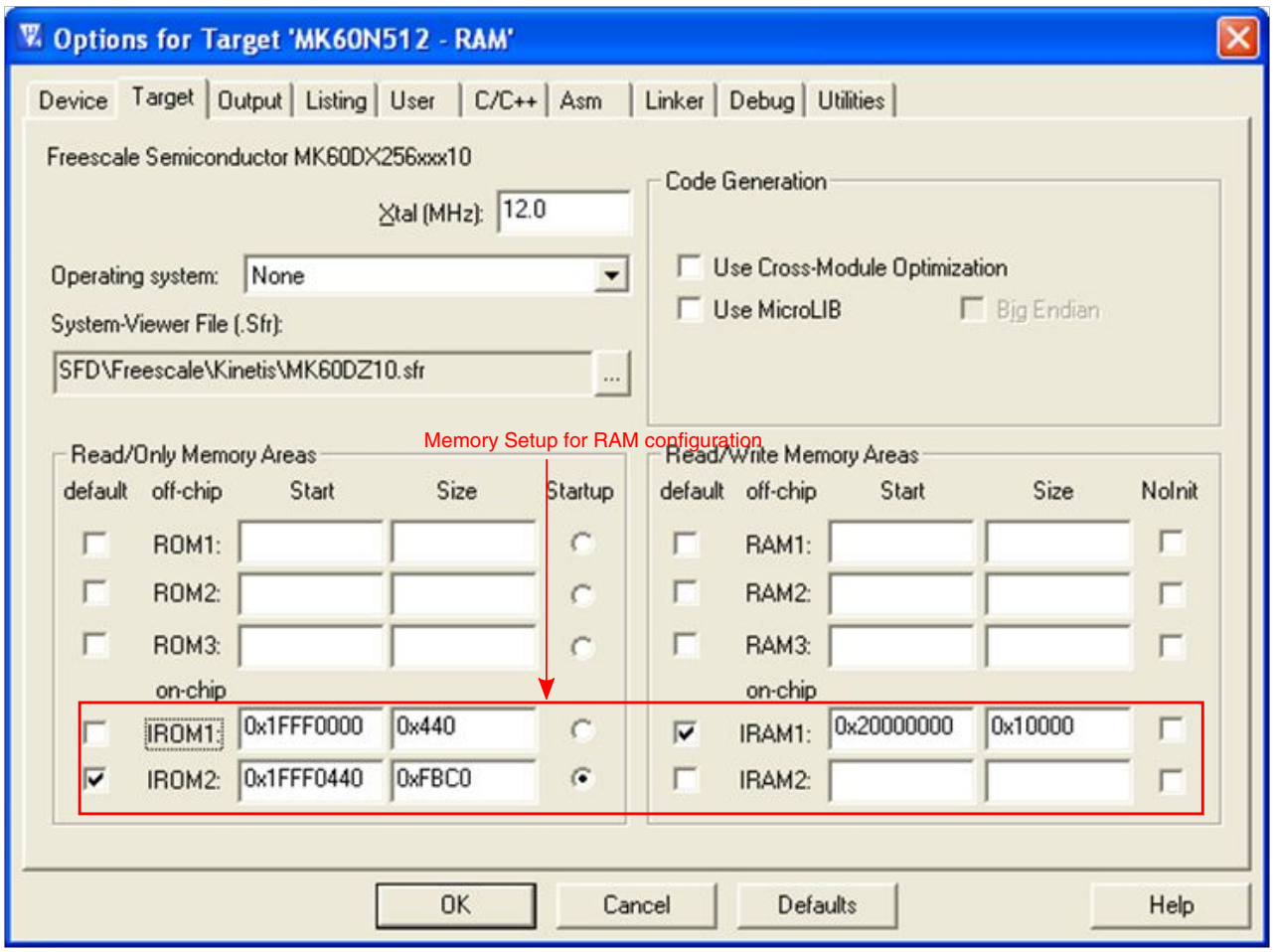
4. Change the name of the executable produced when the project is compiled. This is executed mainly for convenience. Select the Output tab in the dialog box. In the Name of Executable text box, enter the name of the project followed by _<Target>, where <Target> is the particular target type (that is, Flash or RAM). An example is shown in the following figure.



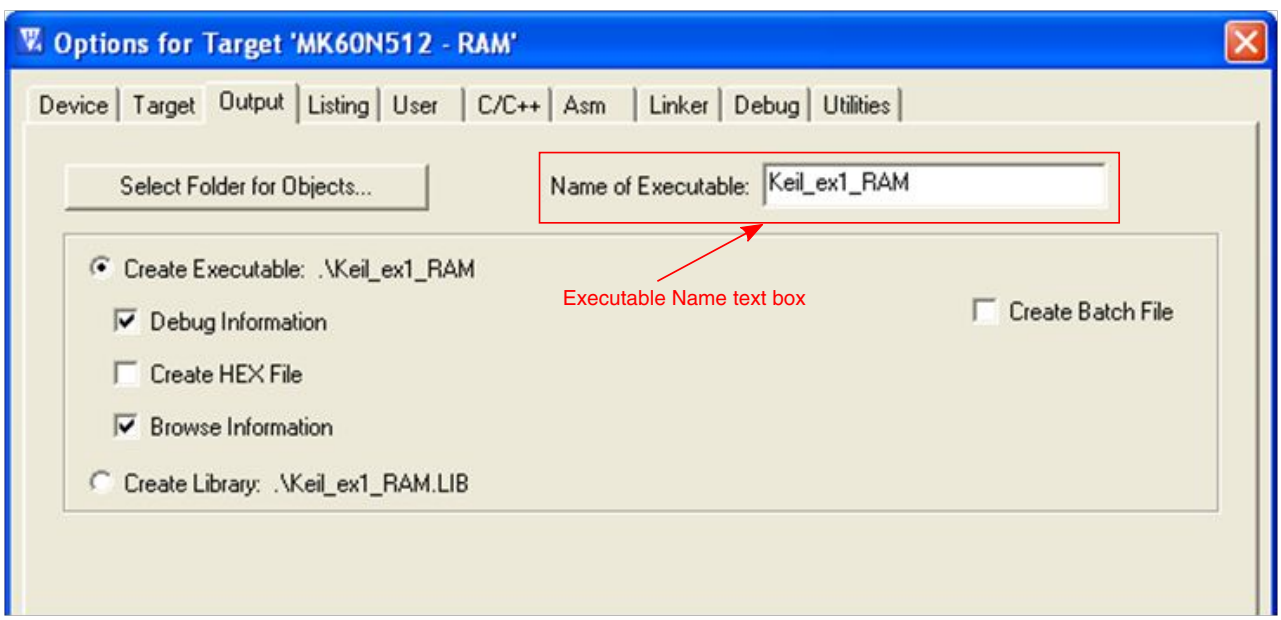

5. Click Ok to exit the Target Options dialog box.

## 4.3 Memory configuration for a RAM target

1. If you wish to run out of RAM, then you will need to modify these text boxes to the desired program RAM space entered in IROM1 and the other RAM memory locations are in IRAM1. For example, if you want to run this code from the lower half of RAM, the memory setup boxes need to be modified as follows.




2. Change the name of the executable produced when the project is compiled. This is executed mainly for convenience. Select the Output tab in the dialog box. For the Name of Executable text box, enter the name of the project followed "_<Target>", where <Target> is the particular target type (that is, Flash or RAM). An example is shown below.




3. Click Ok to exit the Target Options dialog box.

## 4.4 Adding files and include paths

Now that the groups are correctly setup, add the appropriate files to the groups.

Follow these instructions:

1. Right click on the group folder icon (listed in the group column of Table 1 below) in the Project pane and click the Add Files to Group '<folder name'... button.
2. Then browse to the appropriate folder in the Add Files to Group dialog box, select the files in that folder, and click the Add button.

**NOTE**

By default, uVision is set to find only source files, and finds header files automatically. Therefore, you will not need to add the header files, though you may want to add some header files manually if the file is modified often. The following table lists all of the files that are located in the sample project folder structure. Not all of these files will be used for every project and it is recommended that only necessary files be added to a project (the unnecessary files for the hello_world example are in italics in the table).

**Table 1. Project file groupings**

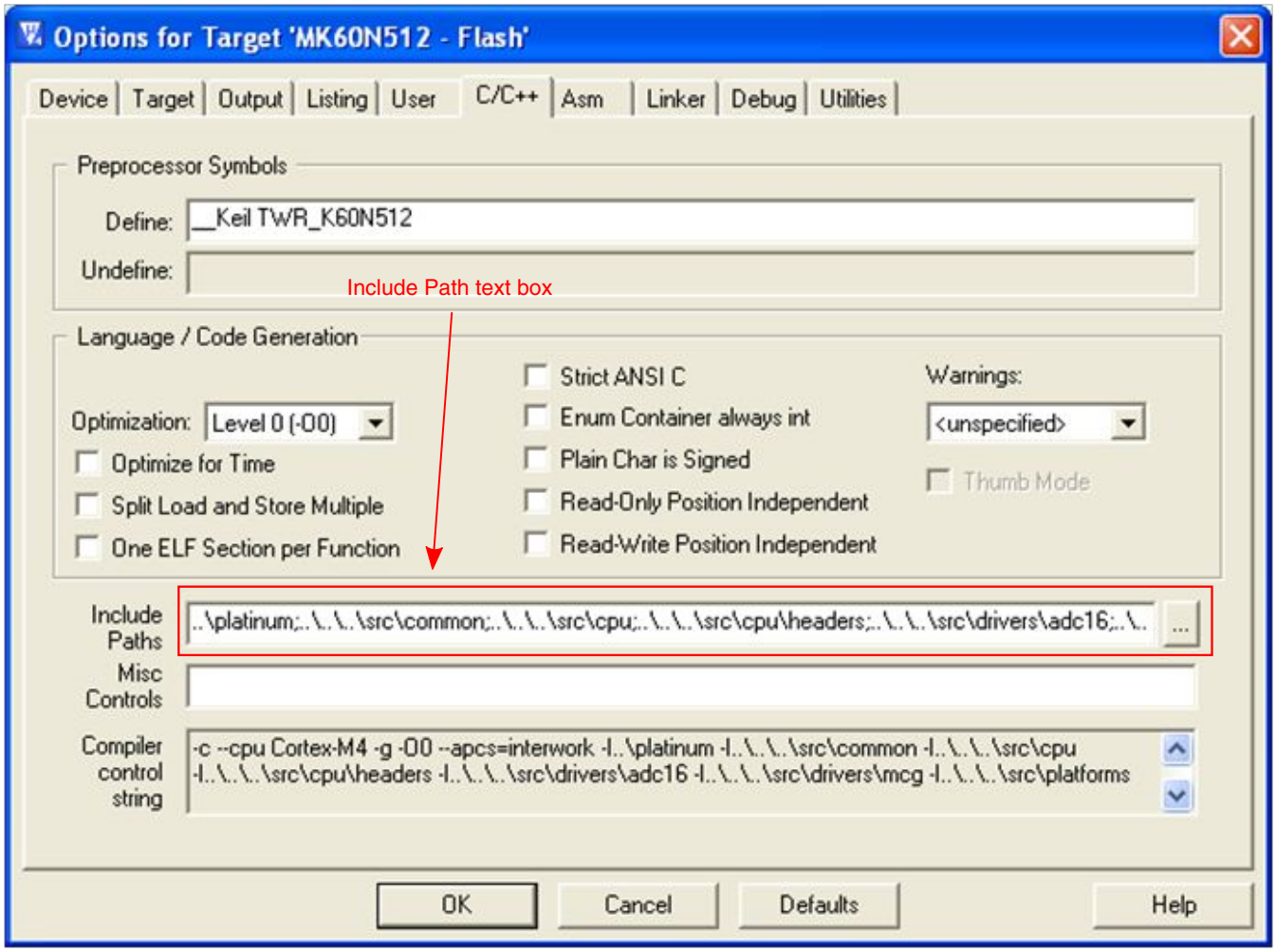
| Group | Files |
|---|---|
| Project | hello_world.c |
| common | *uif.c* |
| | alloc.c |
| | assert.c |
| | io.c |
| | *memtest.c* |
| | *printf.c* |
| | queue.c |
| | *startup.c* |
| | stdlib.c |

*Table continues on the next page...*

**Table 1. Project file groupings (continued)**

| Group | Files |
|---|---|
| drivers | *adc16.c*<br>*nbuf.c*<br>*enet.c*<br>*eth_phy.c*<br>*mii.c*<br>*lptmr.c*<br>mcg.c<br>pmc.c<br>*rtc.c*<br>uart.c<br>wdog.c |
| cpu | arm_cm4.c<br>start.c<br>sysinit.c<br>vectors.c |
| platforms | k60_tower.h<br>k40_tower.h<br>k53_tower.h |

3. In addition, you will need to copy Retarget.c and Serial.c to the UART driver folder (found at <Sample Code Root Directory>\src\drivers\uart) and add these files to your project. These files are necessary for the hardware UART to work correctly. Retarget.c is included with your uVision download and can be found at <Keil Installation Directory> \v4_23\ARM\Startup. Serial.c is included with this application note.
4. After groups are created and the appropriate files have been added, setup the Include Paths, so that uVision is able to find all of the appropriate files, by typing the project relative paths into the Include Paths box (in a semicolon delimited list) in the C/C++ tab of the target options dialog box.




This setting will need to be added on a per target basis. The include paths that should be added are as follows:

..\<project_name>;

..\..\..\src\common;

..\..\..\src\cpu;

..\..\..\src\cpu\headers;

..\..\..\src\drivers\adc16;

..\..\..\src\drivers\mcg;

..\..\..\src\platforms;

..\..\..\src\projects\<project_name>;

..\..\..\src\drivers\uart;

..\..\..\src\drivers\wdog;

..\..\..\src\drivers\pmc;

..\..\..\src\drivers\enet;

..\..\..\src\drivers\mcg;

..\..\..\src\drivers\rtc;

..\..\..\src\drivers\adc16;

..\..\..\src\drivers\lptmr

**Tip**

Here are the paths as an in-line list for easy copying into the include paths text box:

..\Keil_ex1;..\..\..\src\common;..\..\..\src\cpu;..\..\..\src\cpu\headers;..\..\..\src\drivers\adc16;..\..\..\src\drivers\mcg;..\..\..\src\platforms;..\..\..\src\projects\hello_world;..\..\..\src\drivers\uart;..\..\..\src\drivers\wdog;..\..\..\src\drivers\pmc;..\..\..\src\drivers\lptmr

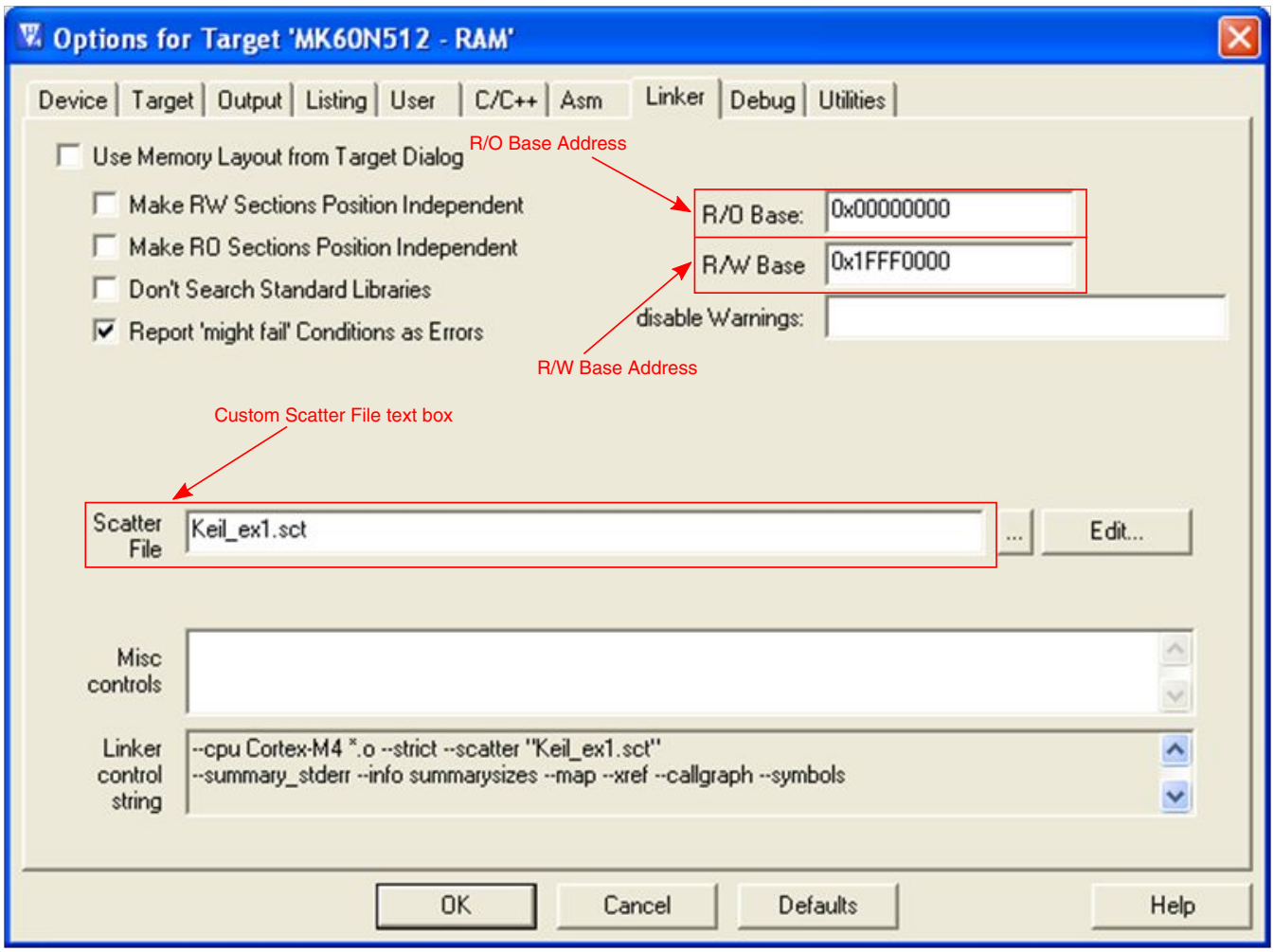
**NOTE**

These file paths can be quickly added to your project by copying and pasting each of these into the target options or by copying and pasting the links from the example project included in this application note.

## 4.5 Creating a scatter file

Scatter files are used by the uVision environment to know where to place sections of code and variables. The scatter file is analogous to the linker file for other IDEs. These files can be generated automatically (as already shown here) or you can designate a custom scatter file for use by the IDE. If you have a scatter file you would like to use, go to the linker tab of the target options dialog box, uncheck “Use Memory Layout from Target Dialog and point uVision to the desired scatter file by clicking the “…” and browsing to the location of your scatter file.

Note that when your own custom scatter file, you must also ensure that the R/O Base address and the R/W Base address are correct.




For more information on creating your own scatter file, consult the uVision help files. An example of a three region scatter file written for an MK60X256 Flash target is shown below and provided with this application note.

```
; *************************************************************
; *** Scatter-Loading Description File generated by uVision ***
; *************************************************************

LR_IROM1 0x00000000 0x00040000  {    ; load region size_region
  ER_IROM1 0x00000000 0x00040000  {  ; load address = execution address
   *.o (RESET, +First)
   *(InRoot$$Sections)
   .ANY (+RO)
  }
  RW_IRAM1 0x1FFF8440 0x00007BF0  {  ; RW data
   .ANY (+RW +ZI)
   mcg.o (+RO +ZI +RW)
  }
  RW_IRAM3 0x20000000 0x00008000  {
   .ANY (+RW +ZI)
  }
  RW_IRAM2 0x1FFF8000 0x00000440  {
    .ANY (+ZI +RW)
  }
}

LR_IROM2 0x10000000 0x00040000  {
  ER_IROM2 0x10000000 0x00040000  {  ; load address = execution address
   .ANY (+RO)
  }
}
```

# 5 File Modifications

## 5.1 Device header file modifications

The current header file in this sample code base is not compatible with the uVision compiler. The uVision compiler does not by default allow the use of anonymous unions. Therefore, a #pragma statement is required to allow the compiler to use the anonymous unions located throughout the header files. Therefore, in the part header file (MKXXDZ10.h), some modifications are be required and these modifications are noted in the table below.

**Table 2. Device Header file modifications**

| Line Number | Statements to add |
|---|---|
| 248 | #elif defined (__Keil)<br>#pragma push<br>#pragma anon_unions |
| 15566 | #elif defined (__Keil)<br>#pragma pop |

With these modifications, lines 241-253 should be written as shown below.

```
00236
00237 /*
00238 ** Start of section using anonymous unions
00239 */
00240
00241 #if defined(__CWCC__)
00242   #pragma push
00243   #pragma cpp_extensions on
00244 #elif defined(__GNUC__)
00245   /* anonymous unions are enabled by default */
00246 #elif defined(__IAR_SYSTEMS_ICC__)
00247   #pragma language=extended
00248 #elif defined (__Keil)
00249   #pragma push
00250   #pragma anon_unions
00251 #else
00252   #error Not supported compiler type
00253 #endif
00254
00255 /* ---------------------------------------------------------------------------
00256    -- ADC
00257    --------------------------------------------------------------------------- */
```

And lines 15560-15570 should be written as shown below.

```
15555
15556 /*
15557 ** End of section using anonymous unions
15558 */
15559
15560 #if defined(__CWCC__)
15561   #pragma pop
15562 #elif defined(__GNUC__)
15563   /* leave anonymous unions enabled */
15564 #elif defined(__IAR_SYSTEMS_ICC__)
15565   #pragma language=default
15566 #elif defined (__Keil)
15567   #pragma pop
15568 #else
15569   #error Not supported compiler type
15570 #endif
15571
15572 /**
15573  * @}
15574  */ /* end of group Peripheral_defines */
15575
15576
15577 /* ----------------------------------------------------------
15578    -- Backward Compatibility
15579    ----------------------------------------------------------
15580
```

## 5.2 CPU folder file modifications

The CPU files include vectors.c, arm_cm4.c, start.c, crt0.s, and sysinit.c. Begin with the modifications to start.c.

### 5.2.1 Start.c

The call to __main in uVision copies non-root (RO and RW) execution regions from their load addresses to their execution addresses, uncompresses necessary data sections, zeroes ZI regions, and then branches to __rt_entry (which sets up the stack and heap). Therefore, most of the common_startup() function is already in process and this function can be skipped using an #ifndef statement. However, __main does not copy the interrupt vectors, so you will need to add this function to start(). Therefore, lines 24-37 must now be as follows:

```
022 void start(void)
023 {
024     #ifndef __Keil
025     /* Disable the watchdog timer */
026     wdog_disable();
027
028     /* Copy any vector or data sections that need to be in RAM */
029     common_startup();
030     #endif
031
032     #ifdef __Keil
033
034     Relocate_ISRs();
035
036     #endif
037
```

The Relocate_ISRs() function can be defined as a local function to start.c. The function is written as follows (and should be added to the end of start.c) and remember to add the function declaration to start.h.

```
303
304 void Relocate_ISRs(void)
305 {
306
307   extern uint32 __vector_table[];
308
309   extern uint32 __VECTOR_RAM[];
310   uint16 n;
311
312   // Copy the vector table to RAM
313   if (__VECTOR_RAM != __vector_table)
314   {
315      for(n = 0; n < 0x104; n++)
316         __VECTOR_RAM[n] = __vector_table[n];
317
318   }
319   // Point the VTOR to the new copy of the vector table
320   SCB_VTOR = ((uint32)__VECTOR_RAM);
321
322 }
323
```

In addition, skip to the call to main(). Skipping the call to main() will require some project file modifications. To skip the call to main() in the start file, modify start.c as shown in the following:

```
066
067     #ifndef __Keil
068     /* Jump to main process */
069     main();
070
071
072     /* No actions to perform after this so wait forever */
073     while(1);
074
075     #endif
076 }
077 /*****************************************************************
```

## 5.2.2 arm_cm4.c

The files arm_cm4.h and arm_cm4.c also require modification. The main function of our sample code is defined in arm_cm4.h. This has been defined as a function that returns nothing and accepts no arguments. However, uVision requires that main return an integer. Therefore, this definition will have to be changed in the declaration of main and lines 83-87 of arm_cm4.h should now be as follows:

```
80  typedef volatile uint32      vuint32; /* 32 bits */
81
82  // function prototype for main function
83  #ifdef __Keil
84  int main(void);
85  #else
86  void main(void);
87  #endif
```

In addition, change the EnableInterrupts definition and DisableInterrupts definition (which are defined in arm_cm4.h at lines 27 and 30) when using uVision. These macros must be changed because uVision does not implement inline assembly in the same manner as most IDEs in the market today. uVision uses a set of intrinsic functions defined in the CMSIS standard for implementing the inline assembly instructions that are typically used throughout the sample code. Modifying the macro defined in arm_cm4.h saves valuable time and frustration when compiling the sample code in uVision. Therefore, the new macro definitions must look as follows:

```
024
025 /*****************************************************************************/
026   /*!< Macro to enable all interrupts. */
027 #ifdef __Keil
028 #define EnableInterrupts __enable_irq();
029 #else
030 #define EnableInterrupts asm(" CPSIE i");
031 #endif
032
033   /*!< Macro to disable all interrupts. */
034 #ifdef __Keil
035 #define DisableInterrupts __disable_irq();
036 #else
037 #define DisableInterrupts asm(" CPSID i");
038 #endif
039 /*****************************************************************************/
040
```

## 5.2.3 arm_cm4.c

The file arm_cm4.c currently defines the stop() and wait() functions. These functions use inline assembly instructions which as previously described, are not allowed in the uVision environment. Therefore, you need to modify those function calls to use the CMSIS defined intrinsic functions. The modified stop() function will now appear as follows:

```
022 void stop (void)
023 {
024     /* Set the SLEEPDEEP bit to enable deep sleep mode (STOP) */
025     SCB_SCR |= SCB_SCR_SLEEPDEEP_MASK;
026
027     /* WFI instruction will start entry into STOP mode */
028     #ifdef __Keil
029     __wfi();
030     #else
031     asm("WFI");
032     #endif
033 }
```

And the wait() function will now appear as follows:

```
046 void wait (void)
047 {
048     /* Clear the SLEEPDEEP bit to make sure we go into WAIT
049      * of deep sleep.
050      */
051     SCB_SCR &= ~SCB_SCR_SLEEPDEEP_MASK;
052
053     /* WFI instruction will start entry into WAIT mode */
054     #ifdef __Keil
055     __wfi();
056     #else
057     asm("WFI");
058     #endif
059 }
060 /*************************************************************
```

## 5.2.4 crt0.s

Copy the current crt0.s file, rename it crt0_keil.s and add the newly created crt0.s file to the project. This file will need to be modified for the uVision environment. Firstly, uVision uses the semicolon (;) to denote that an entire line is a comment. This will be the first change you will want to make.

The current crt0.s file simply initializes the general purpose core registers to ‘0’ and then calls the start function. In the uVision environment, the symbols Stack_Mem and Heap_Mem must be defined. The compiler looks for these symbols, by default in the startup assembly file. Therefore, you will need to define these symbols in your crt0.s file.

In addition, you will want to define __initial_sp and assign it to a known value. This variable tells the uVision environment where to start the stack. By default, the stack will grow down, so this variable should be assigned to the end of the RAM memory space minus eight. To do this, insert the following code at the beginning of the crt0.s file.

```
015 ; <h> Stack Configuration
016 ;   <o> Stack Size (in Bytes) <0x0-0xFFFFFFFF:8>
017 ; </h>
018
019 Stack_Size      EQU     0x00000400
020
021                 AREA    STACK, NOINIT, READWRITE, ALIGN=3
022 Stack_Mem       SPACE   Stack_Size
023 __initial_sp    EQU     0x2000FFF8
024
025
026 ; <h> Heap Configuration
027 ;   <o>  Heap Size (in Bytes) <0x0-0xFFFFFFFF:8>
028 ; </h>
029
030 Heap_Size       EQU     0x00000200
031
032                 AREA    HEAP, NOINIT, READWRITE, ALIGN=3
033 __heap_base
034 Heap_Mem        SPACE   Heap_Size
035 __heap_limit
036
037
038                 PRESERVE8
039                 THUMB
```

Before defining the reset procedure, setup the reset area and let the compiler know about the interrupt vector table by importing the vector table from vectors.c (Note — vectors.c does require modifications for this to work). Insert the following code into the crt0.s file.

```
041
042 ; Vector Table Mapped to Address 0 at Reset
043
044                 AREA    RESET, DATA, READONLY
045                 IMPORT  __vector_table
046
047
048                 AREA    |.text|, CODE, READONLY
049
050
051 ; Define the location of the interrupt vectors in RAM space
052 __VECTOR_RAM            EQU     0x1fff0000
053
```

Now the Reset handler must be defined. uVision will setup the code such that the program begins at the Reset handler upon a reset. The current code in crt0.s can be used to insert the following into your crt0_keil.s file.

```
; Reset Handler

Reset_Handler    PROC
                 IMPORT  wdog_disable
                 IMPORT  __main
                 EXPORT  __initial_sp
                 EXPORT  __VECTOR_RAM
                 EXPORT  Reset_Handler             [WEAK]
                 EXPORT  __startup
__startup
        MOV      r0,#0                    ; Initialize the GPRs
        MOV      r1,#0
        MOV      r2,#0
        MOV      r3,#0
        MOV      r4,#0
        MOV      r5,#0
        MOV      r6,#0
        MOV      r7,#0
        MOV      r8,#0
        MOV      r9,#0
        MOV      r10,#0
        MOV      r11,#0
        MOV      r12,#0
        CPSIE    i                        ; Unmask interrupts
                 LDR      R0, =wdog_disable
                 BLX      R0
                 LDR      R0, =__main
                 BX       R0
                 ENDP


                 ALIGN
```

Notice that the code from the original crt0.s is in the reset handler routine. The reset handler makes itself, the boot stack address, and the startup sequence (initialization of the general purpose registers) available to other parts of the program. Then it calls the watchdog disable function located in wdog.c. Then it will branch back to this file, where __main will be called. __main is the entry point of a uVision program performs the important following functions:

1. Copies non-root (RO and RW) execution regions from their load addresses to their execution addresses. Also, if any data sections are compressed, they are decompressed from the load address to the execution address.
2. Zeroes ZI regions
3. Branches to __rt_entry

__rt_entry performs the following actions:

1. Sets up the stack and the heap by one of a number of means that include calling __user_setup_stackheap(), calling __rt_stackheap_init(), or loading the absolute addresses of scatter-loaded regions.
2. Calls __rt_lib_init() to initialize referenced library functions, initialize the locale and, if necessary, setup arc and argv for main() (user defined main). For C++, calls the constructors for any top-level objects by way of __cpp_initialize_aeabi_.
3. Calls main(), the user-level root of the application.
4. Calls exit() with the value returned by main().

Finally, the assembly file ends with special instructions if you are using the Microlib library. The section of code to perform this setup must be located at the end of this file.

```
086 ; User Initial Stack & Heap
087
088                 IF      :DEF:__MICROLIB
089
090                 EXPORT  __initial_sp
091                 EXPORT  __heap_base
092                 EXPORT  __heap_limit
093
094                 ELSE
095
096                 IMPORT  __use_two_region_memory
097                 EXPORT  __user_initial_stackheap
098 __user_initial_stackheap
099
100                 LDR     R0, =  Heap_Mem
101                 LDR     R1, =(Stack_Mem + Stack_Size)
102                 LDR     R2, = (Heap_Mem +  Heap_Size)
103                 LDR     R3, = Stack_Mem
104                 BX      LR
105
106                 ALIGN
107
108                 ENDIF
109
110
111                 END
112
113
114
```

## 5.2.5 vectors.c

Now that the assembly startup file is written (and references vectors.c), modify vectors.c for operation in the uVision environment. Modify the #if statement to include a uVision option defining the vector table array and indicate to the linker where to put this array. To do this, insert the following code into the #if statement of vectors.c found at line 16.

```
011 /*****************************************************************************
012 * Vector Table
013 *****************************************************************************/
014 typedef void (*vector_entry)(void);
015
016 #if defined(IAR)
017   #pragma location = ".intvec"
018   const vector_entry  __vector_table[] = //@ ".intvec" =
019 #elif defined(CW)
020   #pragma define_section vectortable ".vectortable" ".vectortable" ".vectortable" far_abs R
021   #define VECTOR __declspec(vectortable)
022   const VECTOR vector_entry  __vector_table[] = //@ ".intvec" =
023 #elif (defined(__Keil))
024   const vector_entry  __attribute__((section(".ARM.__at_0x00"))) __vector_table[] = //@ ".int
025 #endif
026
027 {
028   VECTOR_000,           /* Initial SP           */
029   VECTOR_001,           /* Initial PC           */
030   VECTOR_002,
```

For more information on the __attribute_ command, refer to the uVision help files.

## 5.2.6 vectors.h

The first two vectors in the vector table need to point to the initial stack pointer and the Reset_Handler. Therefore, you will need to modify vectors.h as follows:

```
022
023 extern void __startup(void);
024 extern void Reset_Handler(void);
025
026 extern unsigned long __BOOT_STACK_ADDRESS[];
027 extern unsigned long __initial_sp[];
028 extern void __iar_program_start(void);
029
030 #ifdef __Keil
031 #define VECTOR_000      (pointer*)__initial_sp
032 #define VECTOR_001      Reset_Handler
033 #else
034                                     // Address     Vector IRQ
035 #define VECTOR_000      (pointer*)__BOOT_STACK_ADDRESS  //
036 #define VECTOR_001      __startup   // 0x0000_0004 1 -          ARM
037 #endif
038 #define VECTOR_002      default_isr     // 0x0000_0008 2 -
039 #define VECTOR_003      default_isr     // 0x0000_000C 3 -
```

## 5.3 Common folder file modifications

The common files include uif.c, uif.h, alloc.c, assert.c, io.c, io.h, memtest.c, memtest.h, printf.c, queue.c, queue.h, startup.c, stdlib.c, and stdlib.h. We will begin with the modifications to alloc.c.

### 5.3.1 common.h

The header file common.h contains a selection statement that selects a compiler specific header file based on the compiler being used. To eliminate the warning when using the uVision IDE, a modification to this code is necessary. At line 61, insert the following code.

```
54 /*
55  * Include any toolchain specfic header files
56  */
57 #if (defined(CW))
58   #include "cw.h"
59 #elif (defined(IAR))
60   #include "iar.h"
61 #elif (defined(__Keil))
62 #else
63 #warning "No toolchain specific header included"
64 #endif
65
```

### 5.3.2 alloc.c

In alloc.c, there are compiler specific #if statements that define variables necessary for the malloc function. The variables __HEAP_START and __HEAP_END are populated from linker symbols. To populate these in uVision, use of the uVision specific linker symbols are needed. Therefore, insert the following code at line 85 such that lines 80-91 are as follows:

```
079     /* Get addresses for the HEAP start and end */
080     #if (defined(CW))
081       extern char __HEAP_START;
082       extern char __HEAP_END[];
083     #elif (defined(IAR))
084       char* __HEAP_START = __section_begin("HEAP");
085       char* __HEAP_END = __section_end("HEAP");
086     #elif (defined(__Keil))
087       extern uint32_t HEAP$$Base;
088       extern uint32_t HEAP$$Limit;
089       uint32_t __HEAP_START = (uint32_t) &HEAP$$Base;
090       uint32_t __HEAP_END = (uint32_t) &HEAP$$Limit;
091     #endif
092
```

## 5.4 Driver file modifications

The only driver files that need modification are the mcg.h and mcg.c files. These files have IDE specific declarations for the set_sys_dividers subroutine because this subroutine needs to be placed in RAM memory due to an errata (Errata e2448) on the Kinetis family 0M33Z mask set. Placing this function in RAM memory allows the code base to be compatible with all Kinetis family revisions. (Specifying the code location of functions are discussed in the next section.) For now, the focus is on the changes to the code in the driver files.

### 5.4.1 mcg.h

To make these files uVision compatible, add an #if statement to define the function when using uVision. In the header file, mcg.h, lines 40-47 must be written as follows:

```
39
40 #if (defined(IAR))
41     __ramfunc void set_sys_dividers(uint32 outdiv1, uint32 outdiv2, uint32 outdiv3, uint32 outdiv4);
42 #elif (defined(CW))
43     __relocate_code__
44     void set_sys_dividers(uint32 outdiv1, uint32 outdiv2, uint32 outdiv3, uint32 outdiv4);
45 #elif (defined(__Keil))
46     void set_sys_dividers(uint32 outdiv1, uint32 outdiv2, uint32 outdiv3, uint32 outdiv4);
47 #endif
48
```

```
39
40 #if (defined(IAR))
41     __ramfunc void set_sys_dividers(uint32 outdiv1, uint32 outdiv2, uint32 outdiv3, uint32 outdiv4);
42 #elif (defined(CW))
43     __relocate_code__
44     void set_sys_dividers(uint32 outdiv1, uint32 outdiv2, uint32 outdiv3, uint32 outdiv4);
45 #elif (defined(__Keil))
46     void set_sys_dividers(uint32 outdiv1, uint32 outdiv2, uint32 outdiv3, uint32 outdiv4);
47 #endif
48
```

### 5.4.2 mcg.c

In the source file, mcg.c, lines 141-148 must be written as follows:

```
140   */
141 #if (defined(IAR))
142     __ramfunc void set_sys_dividers(uint32 outdiv1, uint32 outdiv2, uint32 outdiv3, uint32 outdiv4)
143 #elif (defined(CW))
144 __relocate_code__
145 void set_sys_dividers(uint32 outdiv1, uint32 outdiv2, uint32 outdiv3, uint32 outdiv4)
146 #elif (defined(__Keil))
147     void set_sys_dividers(uint32 outdiv1, uint32 outdiv2, uint32 outdiv3, uint32 outdiv4)
148 #endif
149 (
```

In addition to these changes, the source file declares variables drs_val and dmx32_val as externs, and they have not been explicitly defined elsewhere in the program. As mentioned previously, the uVision environment does not allow this coding practice. Therefore, at line 23, the word extern must be removed from the statement and the code should now be written as follows:

```
020 extern int core_clk_khz;
021 extern int core_clk_mhz;
022 extern int periph_clk_khz;
023 char drs_val, dmx32_val;
024
```

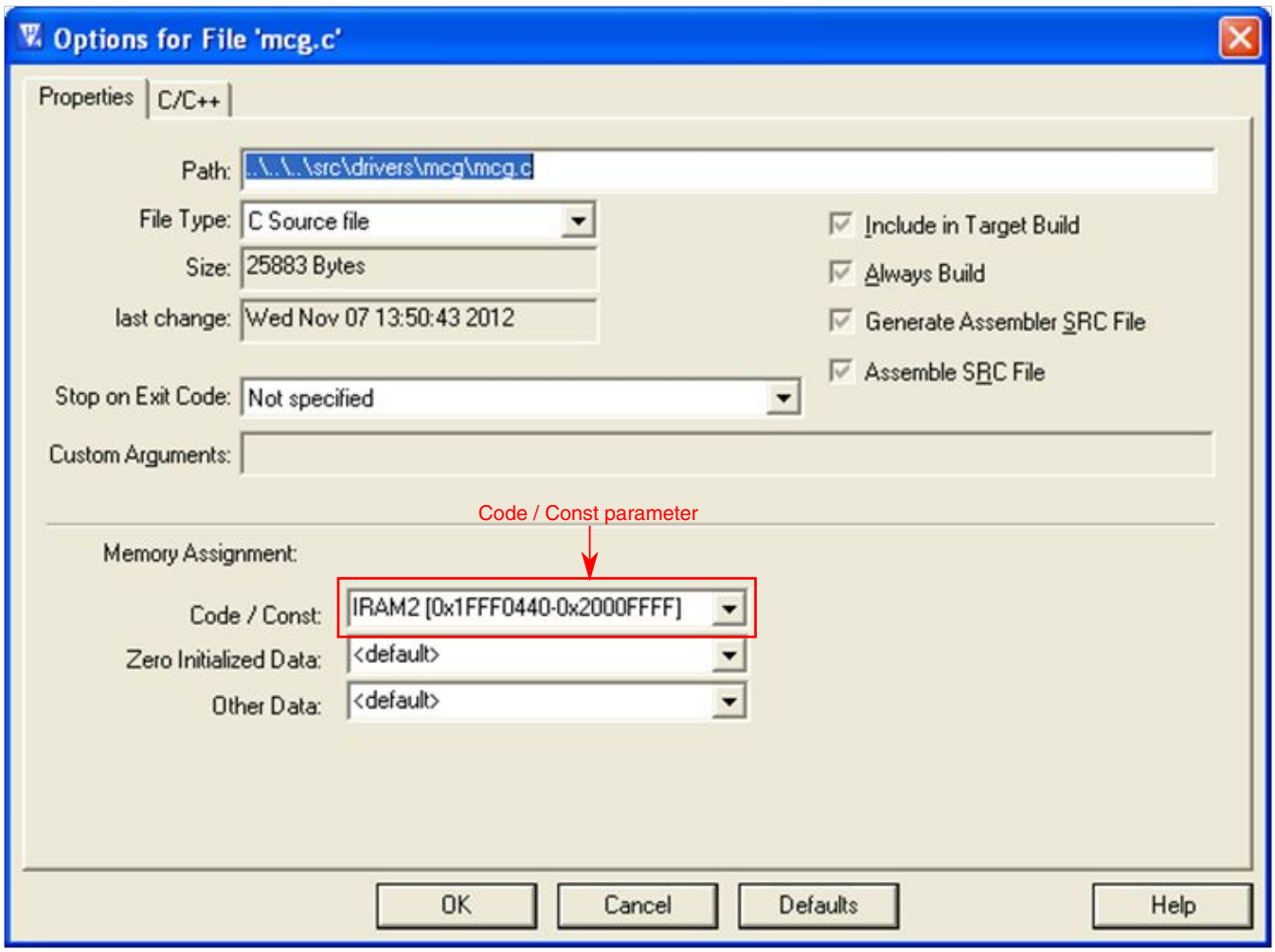
## 5.4.3 enet.c

The Ethernet driver files use an inline assembly instruction call to implement a wait function. However, as previously mentioned, the uVision environment does not accommodate these instructions. Therefore, you will need to modify the enet_reset function at line 134 in enet.c to the following:

```
132  */
133 void
134 enet_reset (int ch)
135 {
136     int i;
137
138     /* Set the Reset bit and clear the Enable bit */
139     ENET_ECR/*(ch)*/ = ENET_ECR_RESET_MASK;
140
141     /* Wait at least 8 clock cycles */
142     for (i=0; i<10; ++i)
143     {
144         #ifdef __Keil
145             __NOP;
146         #else
147             asm( "NOP" );
148         #endif
149     }
150 }
```

# 5.5 Specifying memory locations for code and variables

It is beneficial to be able to specify where specific functions and variables reside in memory. For example, one errata against the Kinetis part requires that certain functions within the MCG driver be placed in RAM memory. uVision does not allow specific functions to be placed in specific locations in memory. However, it does allow you to place entire files or variables into specific places in memory. If you are allowing uVision to create your scatter file, then you can place an entire file in RAM memory by right clicking on the file (try this with mcg.c) and selecting Options. In the Options dialog box for the file, select the desired location for the Code/Const parameter in the Memory Assignment section (shown below). In this particular case, you want to select IRAM2, this is the location of our internal RAM memory.




If you have created your own scatter file, you can specify where in memory to place that file by listing the associated *.o file in the desired section of code. Consider mcg.c again. To place this file in RAM memory when using a predefined scatter file, just list mcg.o in the desired RAM location. The following figure shows an example of this.

## 5.6 Using the Hardware UART

The uVision environment provides semi-hosting support and is configured for this by default. For the printf() function to send data to the hardware UART (and not the debugger console), the Keil provided retarget.c, Serial.h, and Serial.c must be used. These files disable semi-hosting and retarget the library functions that use semi-hosting.

### 5.6.1 Retarget.c

Retarget.c is found in <root>\Keil\vXXX\ARM\Startup\Retarget.c, where vXXX is your uVision version number. Keep in mind that it must be copied into your local folder structure when adding it to the project. This file is where semi-hosting is disabled and where most of the retargeting is performed. Minor modifications are required for proper operation. The functions fputc() and _ttywrch() should return SER_PutChar() instead of their default return values. These functions are found in Serial.c, so you must include Serial.h at the top of Retarget.c. The necessary modifications should result in the following code:

```
28
29  int fputc(int ch, FILE *f) {
30    //return (sendchar(ch));
31    return (SER_PutChar(ch));
32  }
33
```

```
45
46  void _ttywrch(int ch) {
47    //sendchar (ch);
48    SER_PutChar(ch);
49  }
50
51
```

The function, fgetc(), should return SER_GetChar() instead of its default return value. This function is found in Serial.c and the appropriate header should be included at the top of Retarget.c. The necessary modifications to this function should result in the following:

```
38  int fgetc(FILE *f) {
39    //return (getkey());
40    return (SER_GetChar());
41  }
```

## 5.6.2 Serial.c

Serial.c is provided with this application note and implements the SER_PutChar subroutine and SER_GetChar subroutines while still providing support for semi-hosting if desired. It is important to note that it implements a polled UART and the particular UART used is the UART defined by TERM_PORT. The SER_PutChar and SER_GetChar subroutines are displayed below.

```
31  /*----------------------------------------------------------------------------
32   *       SER_PutChar:  Write a character to the Serial Port
33   *----------------------------------------------------------------------------*/
34  int32_t SER_PutChar (int32_t ch) {
35  #ifdef __DBG_ITM
36    int i;
37    ITM_SendChar (ch & 0xFF);
38    for (i = 10000; i; i--);
39  #else
40    // Wait until space is available in the FIFO //
41    while (!(UART_S1_REG(TERM_PORT) & UART_S1_TDRE_MASK))
42    {}
43
44    // Now Send the character //
45    UART_D_REG(TERM_PORT) = (uint8)ch;
46
47  #endif
48    return (ch & 0xFF);
49  }
50
```

```
/*----------------------------------------------------------------------------
 *       SER_GetChar:  Read a character from the Serial Port
 *----------------------------------------------------------------------------*/
int32_t SER_GetChar (void) {
#ifdef __DBG_ITM
  if (ITM_CheckChar())
    return ITM_ReceiveChar();
#else
    // Wait until character has been received //
    while (!(UART_S1_REG(TERM_PORT) & UART_S1_RDRF_MASK));

    // Return the 8-bit data from the receiver //
    return UART_D_REG(TERM_PORT);

#endif

}
```

## 5.7 Project file modifications

Since the majority of the changes have been taken care of in the common files, there should be minimal changes to your project files. In the interest of being brief, the necessary source file modifications for all of the sample code projects are not covered here. With this in mind, the following bullet points should aid in porting your sample code to the uVision environment.

- Since the declaration of main has been changed to be ANSI C99 compliant, you will also need to change the definition of main to be C99 compliant in the source.
- The first action the main must take is to call start(). With the modifications made, the startup assembly file does not call the start function, and therefore, the clocks and the UART will not be properly initialized unless this function is called.
- Some interrupt vectors may still be defined in a way that will not be accepted by the uVision environment. If your compilation attempt returns errors due to an interrupt vector assignment, change the interrupt vector assignment so that it first undefines the desired interrupt vector number, and then redefines this vector number as the name of the interrupt routine you have written.
- The uVision console library does not automatically add a '\r' character when using the '\n' character in print statements. You may want pr need to add these characters to your print statements for formatting purposes.

# 6 Checking your newly compiled project

When developing a new application, it is often beneficial to check that the generated code is within your size and location constraints. The uVision environment does not automatically update an output folder with the linker map, as is done in many other IDEs. However, you can still view these files by navigating to the build folder and dragging the desired file into the uVision environment.

# 7 Conclusion

This application note provides the basic concepts for porting your code to the uVision environment. It has shown how to leverage uVision's built in features to be able to easily configure a project for a different target.

The examples provided are specifically aimed at the Kinetis MK60N512 part but the concepts could be easily implemented on other parts as well.

## *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



Information in this document is provided solely to enable system and software implementers to use Freescale Semiconductors 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 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 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 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 claims alleges that Freescale Semiconductor was negligent regarding the design or manufacture of the part.

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

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

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

© 2012 Freescale Semiconductor, Inc.



Document Number: AN4552
Rev. 0, 10/2012