Porting Freescale ARM Compiler-based Projects to use ARM GCC

Rev. April 15, 2013

Contents

Section number **Title** **Page**

Chapter 1
Introduction

This guide will describe how to port Freescale ARM Assembly Code to GCC ARM Assembly Code. The guide includes the following topics.

- Using GCC to Build Legacy Kinetis Projects
- Guidelines

NOTE

The GCC ARM documentation is available at \CW MCU v10.x \Cross_Tools\arm-noneeabigcc- X_Y_Z\share\doc\gcc-arm-none-eabi.

Chapter 2
Using GCC to Build Legacy Kinetis Projects

The following are the tips and tricks on how to use GCC and build legacy Kinetis projects in CodeWarrior v10.x.

2.1 Tips and Tricks to Port

The tips and tricks to port are:

- Change Tool Chain to GCC
- Replace link command file
- Add Startup Files
- Modify System init File
- Specify Application File in Debug Configuration

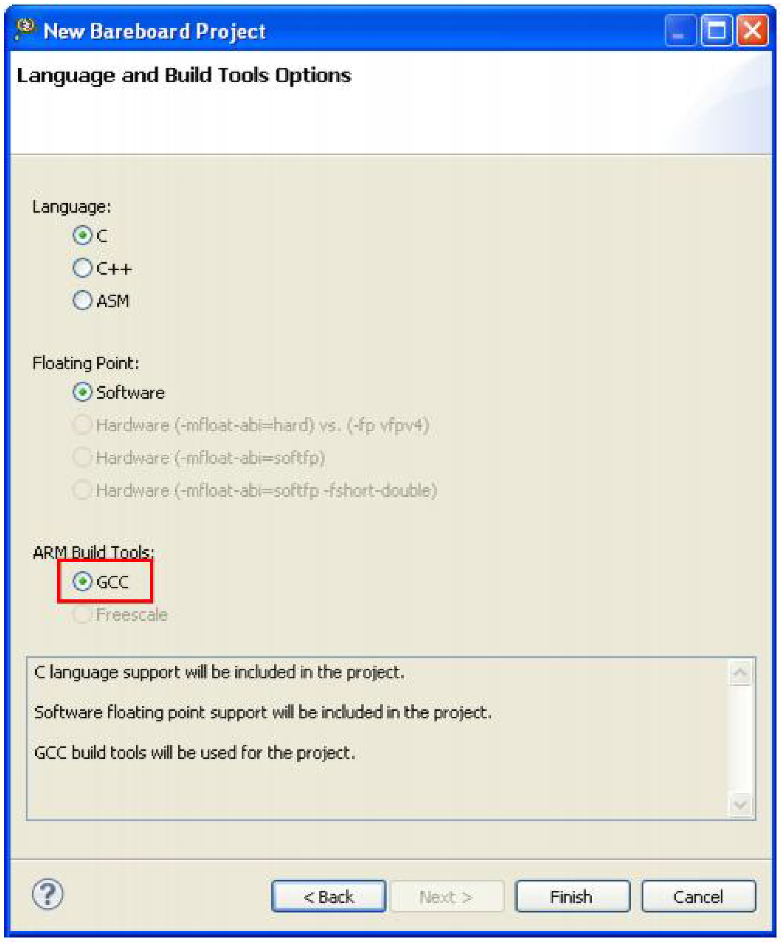
2.1.1 Change Tool Chain to GCC

To change the tool chain:

1. Import the Freescale ARM Compiler-based project into the workspace.
2. Create a new ARM GCC project using project wizard for later use.

NOTE

You need to select GCC as ARM build tool in the **Language and Build Tools Options** page of the new project wizard.



Language: [✓ C] C++ ASM

Floating Point: [✓ Software] Hardware (-mfloat-abi=hard) vs. (-fp vfpv4) Hardware (-mfloat-abi=softfp) Hardware (-mfloat-abi=softfp -fshort-double)

ARM Build Tools: [✓ GCC] Freescale

C language support will be included in the project.

Software floating point support will be included in the project.

GCC build tools will be used for the project.

Figure 2-1. Language and Build Tools Options

3. Replace the `*.cproject` file of your project with the one from the new created GCC project. `*.cproject` file is located on the top level of your project folder. This file contains project settings, such as building tools to be used, various compiler options, and so on. With this trick, the building tool of your project has been changed to GCC.

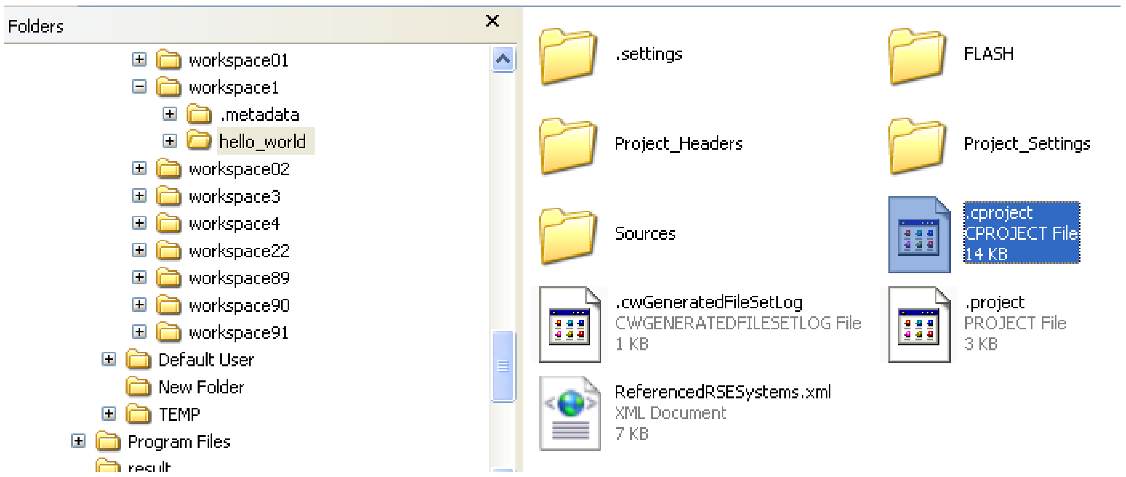
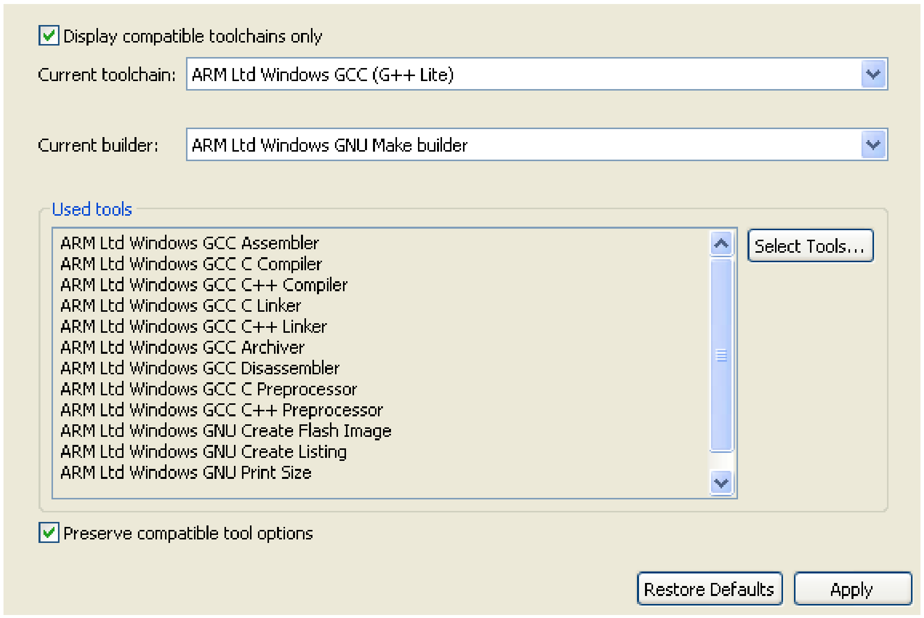


Figure 2-2. Replace the *.cproject file

4. Select your project in the project view and press F5 on the keyboard to refresh the project.
5. Select **Project > Properties** from the menu bar.
6. Navigate to **C/C++ Build > Tool Chain Editor**. In the **Tool Chain Editor** page, the current tool chain will be *ARM Ltd. Windows GCC (G++ Lite* and current builder should be *ARM Ltd. Windows GNU Make builder*.



☑ Display compatible toolchains only

Current toolchain: ARM Ltd Windows GCC (G++ Lite)

Current builder: ARM Ltd Windows GNU Make builder

Used tools

ARM Ltd Windows GCC Assembler
ARM Ltd Windows GCC C Compiler
ARM Ltd Windows GCC C++ Compiler
ARM Ltd Windows GCC C Linker
ARM Ltd Windows GCC C++ Linker
ARM Ltd Windows GCC Archiver
ARM Ltd Windows GCC Disassembler
ARM Ltd Windows GCC C Preprocessor
ARM Ltd Windows GCC C++ Preprocessor
ARM Ltd Windows GNU Create Flash Image
ARM Ltd Windows GNU Create Listing
ARM Ltd Windows GNU Print Size

☑ Preserve compatible tool options

Figure 2-3. Tool Chain Editor

7. Navigate to C/C++ **Build > Settings** . Select *[All configurations]* in Configuration option. You may need to add
   a. **Defined/Undefined symbols** in **ARM Ltd. Windows GCC Assembler > Preprocessor** page.
   b. **Defined/Undefined symbols** in **ARM Ltd. Windows GCC Compiler > Preprocessor** page.
   c. **Include paths** in **ARM Ltd. Windows GCC Assembler > Preprocessor** page.

   d. **Include paths** in **ARM Ltd. Windows GCC Compiler > Preprocessor** page.
   e. **Libraries** in **ARM Ltd. Windows GCC Linker > Libraries** page.
8. Select the **Build Artifact** tab and provide a new artifact name for your project. Ensure that you change the artifact names for different configurations.
9. Click **OK** to save the configurations.

2.1.2 Replace Link Command File

Replace the link command files of your project with those from the new created GCC project (file extension need to be `.ld`). Link command files are located in your *project's***Project Settings > Linker_Files** folder.

2.1.3 Add Startup Files

Copy `__arm_end.c, __arm_start.c, runtime_configuration.h` files from new created GCC project to your project. These three files are located in your project's **Project Settings > Startup_code** folder.

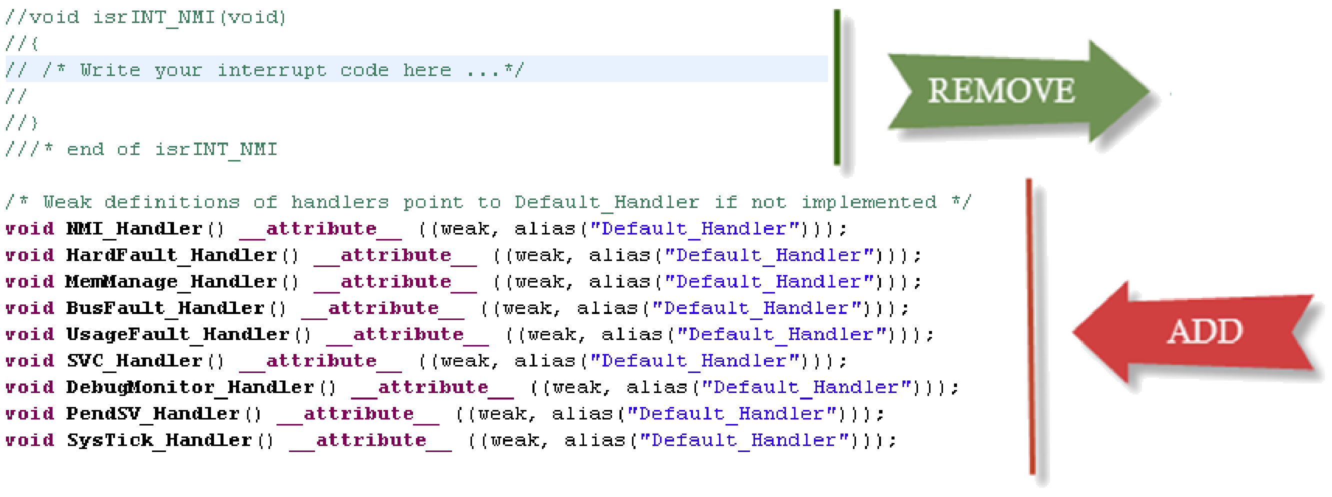
2.1.4 Modify System init File

The system initialization file is usually located in your *project's***Project_Settings > Startup_Code** folder and named *kinetis_sysinit.c*. You can modify this file by following step.

NOTE

Refer to `kinetis_sysinit.c` from the newly created GCC project, for all the code that is added here.

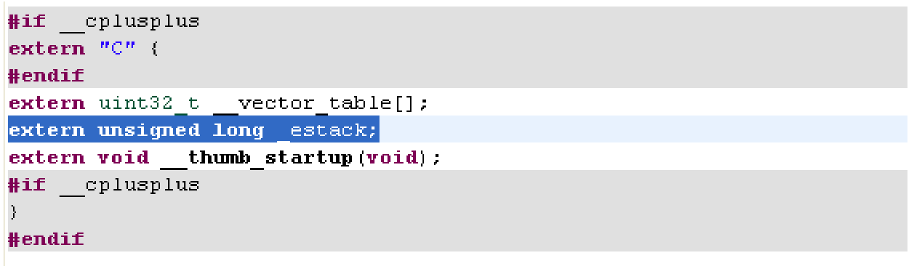
1. Add weak definitions of handlers point to `UNASSIGNED_ISR`.



```
//void isrINT_NMI(void)
//{
// /* Write your interrupt code here ...*/
//
//}
///* end of isrINT_NMI

/* Weak definitions of handlers point to Default_Handler if not implemented */
void NMI_Handler() __attribute__ ((weak, alias("Default_Handler")));
void HardFault_Handler() __attribute__ ((weak, alias("Default_Handler")));
void MemManage_Handler() __attribute__ ((weak, alias("Default_Handler")));
void BusFault_Handler() __attribute__ ((weak, alias("Default_Handler")));
void UsageFault_Handler() __attribute__ ((weak, alias("Default_Handler")));
void SVC_Handler() __attribute__ ((weak, alias("Default_Handler")));
void DebugMonitor_Handler() __attribute__ ((weak, alias("Default_Handler")));
void PendSV_Handler() __attribute__ ((weak, alias("Default_Handler")));
void SysTick_Handler() __attribute__ ((weak, alias("Default_Handler")));
```

Figure 2-4. Add Weak Definition

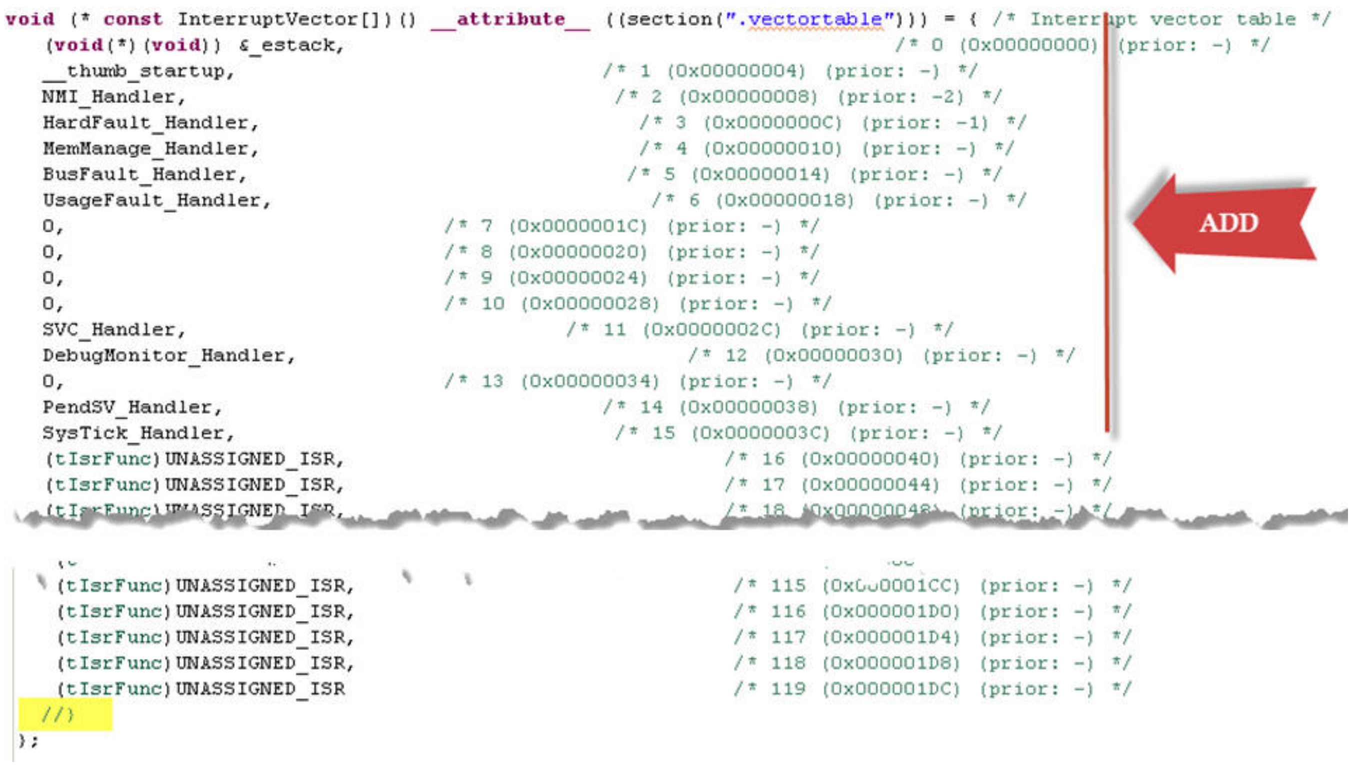
2. Define external `variable _estack.`



```
#if __cplusplus
extern "C" {
#endif
extern uint32_t __vector_table[];
extern unsigned long _estack;
extern void __thumb_startup(void);
#if __cplusplus
}
#endif
```

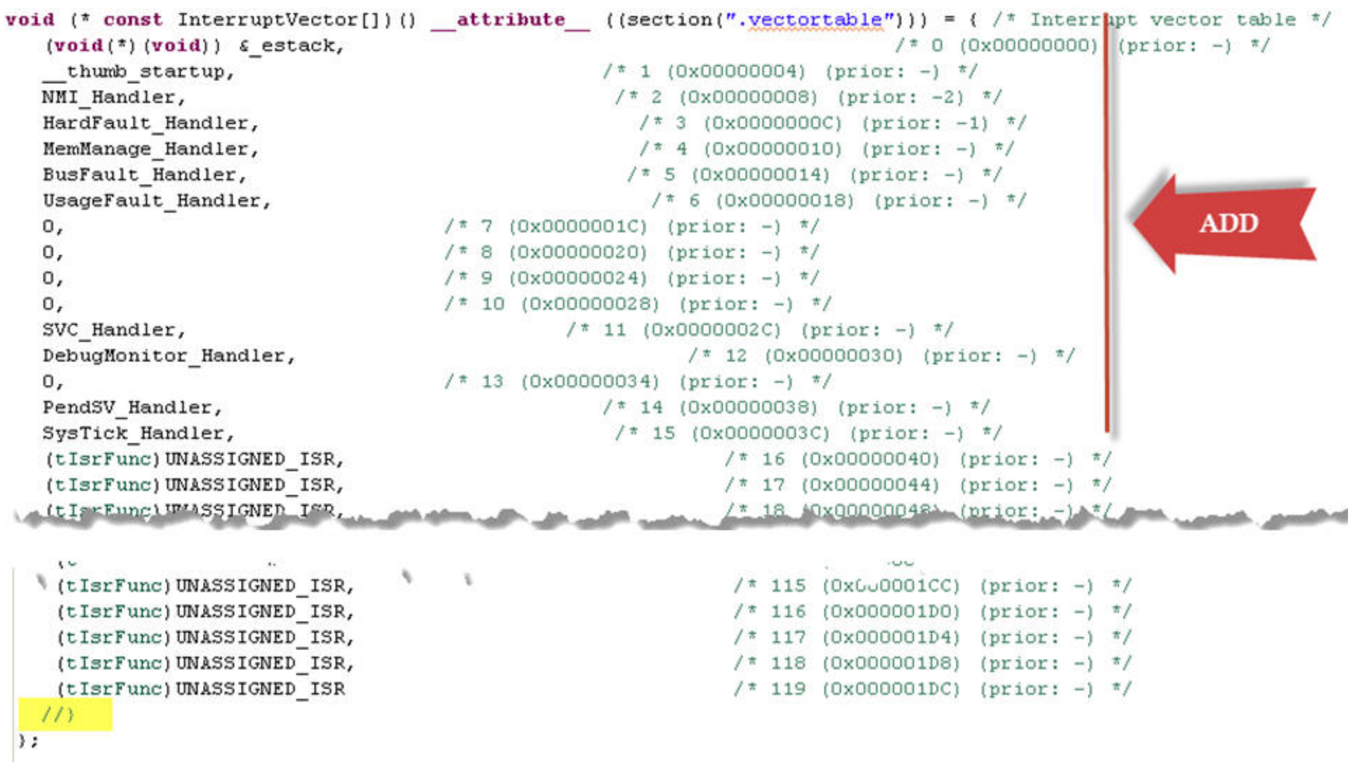
Figure 2-5. Define External Variable

3. Modify interrupt vector table.



```
void (* const InterruptVector[])() __attribute__ ((section(".vectortable"))) = { /* Interrupt vector table */
   (void(*)(void)) &_estack,                                          /* 0 (0x00000000) (prior: -) */
   __thumb_startup,                              /* 1 (0x00000004) (prior: -) */
   NMI_Handler,                                   /* 2 (0x00000008) (prior: -2) */
   HardFault_Handler,                               /* 3 (0x0000000C) (prior: -1) */
   MemManage_Handler,                               /* 4 (0x00000010) (prior: -) */
   BusFault_Handler,                               /* 5 (0x00000014) (prior: -) */
   UsageFault_Handler,                               /* 6 (0x00000018) (prior: -) */
   0,                               /* 7 (0x0000001C) (prior: -) */
   0,                               /* 8 (0x00000020) (prior: -) */
   0,                               /* 9 (0x00000024) (prior: -) */
   0,                               /* 10 (0x00000028) (prior: -) */
   SVC_Handler,                               /* 11 (0x0000002C) (prior: -) */
   DebugMonitor_Handler,                               /* 12 (0x00000030) (prior: -) */
   0,                               /* 13 (0x00000034) (prior: -) */
   PendSV_Handler,                               /* 14 (0x00000038) (prior: -) */
   SysTick_Handler,                               /* 15 (0x0000003C) (prior: -) */
   (tIsrFunc)UNASSIGNED_ISR,                               /* 16 (0x00000040) (prior: -) */
   (tIsrFunc)UNASSIGNED_ISR,                               /* 17 (0x00000044) (prior: -) */
   (tIsrFunc)UNASSIGNED_ISR,                               /* 18 (0x00000048) (prior: -) */

    (tIsrFunc)UNASSIGNED_ISR,                               /* 115 (0x000001CC) (prior: -) */
    (tIsrFunc)UNASSIGNED_ISR,                               /* 116 (0x000001D0) (prior: -) */
    (tIsrFunc)UNASSIGNED_ISR,                               /* 117 (0x000001D4) (prior: -) */
    (tIsrFunc)UNASSIGNED_ISR,                               /* 118 (0x000001D8) (prior: -) */
    (tIsrFunc)UNASSIGNED_ISR                                /* 119 (0x000001DC) (prior: -) */
   //}
};
```

Figure 2-6. Interrupt Vector Table - Before



```
void (* const InterruptVector[])() __attribute__ ((section(".vectortable"))) = { /* Interrupt vector table */
   (void(*)(void)) &_estack,                                          /* 0 (0x00000000) (prior: -) */
   __thumb_startup,                              /* 1 (0x00000004) (prior: -) */
   NMI_Handler,                                   /* 2 (0x00000008) (prior: -2) */
   HardFault_Handler,                               /* 3 (0x0000000C) (prior: -1) */
   MemManage_Handler,                               /* 4 (0x00000010) (prior: -) */
   BusFault_Handler,                               /* 5 (0x00000014) (prior: -) */
   UsageFault_Handler,                               /* 6 (0x00000018) (prior: -) */
   0,                               /* 7 (0x0000001C) (prior: -) */
   0,                               /* 8 (0x00000020) (prior: -) */
   0,                               /* 9 (0x00000024) (prior: -) */
   0,                               /* 10 (0x00000028) (prior: -) */
   SVC_Handler,                               /* 11 (0x0000002C) (prior: -) */
   DebugMonitor_Handler,                               /* 12 (0x00000030) (prior: -) */
   0,                               /* 13 (0x00000034) (prior: -) */
   PendSV_Handler,                               /* 14 (0x00000038) (prior: -) */
   SysTick_Handler,                               /* 15 (0x0000003C) (prior: -) */
   (tIsrFunc)UNASSIGNED_ISR,                               /* 16 (0x00000040) (prior: -) */
   (tIsrFunc)UNASSIGNED_ISR,                               /* 17 (0x00000044) (prior: -) */
   (tIsrFunc)UNASSIGNED_ISR,                               /* 18 (0x00000048) (prior: -) */

    (tIsrFunc)UNASSIGNED_ISR,                               /* 115 (0x000001CC) (prior: -) */
    (tIsrFunc)UNASSIGNED_ISR,                               /* 116 (0x000001D0) (prior: -) */
    (tIsrFunc)UNASSIGNED_ISR,                               /* 117 (0x000001D4) (prior: -) */
    (tIsrFunc)UNASSIGNED_ISR,                               /* 118 (0x000001D8) (prior: -) */
    (tIsrFunc)UNASSIGNED_ISR                                /* 119 (0x000001DC) (prior: -) */
   //}
};
```

Figure 2-7. Interrupt Vector Table - After

4. Provide Default interrupt handler.

```
void Default_Handler()
```

```
{
  __asm("bkpt");
}
```

5. Remove the following pragmas to have clean build with no warnings.

   ```
   #pragma define_section vectortable ".vectortable" ".vectortable" ".vectortable" far_abs
   R
   ```

   and

   ```
   #pragma overload void __init_hardware();
   ```

2.1.5 Specify Application File in Debug Configuration

To specify the correct application file in the **Debug Configurations** dialog box:

1. Select your project and click the build icon on the toolbar.

   The project should be built without an error.

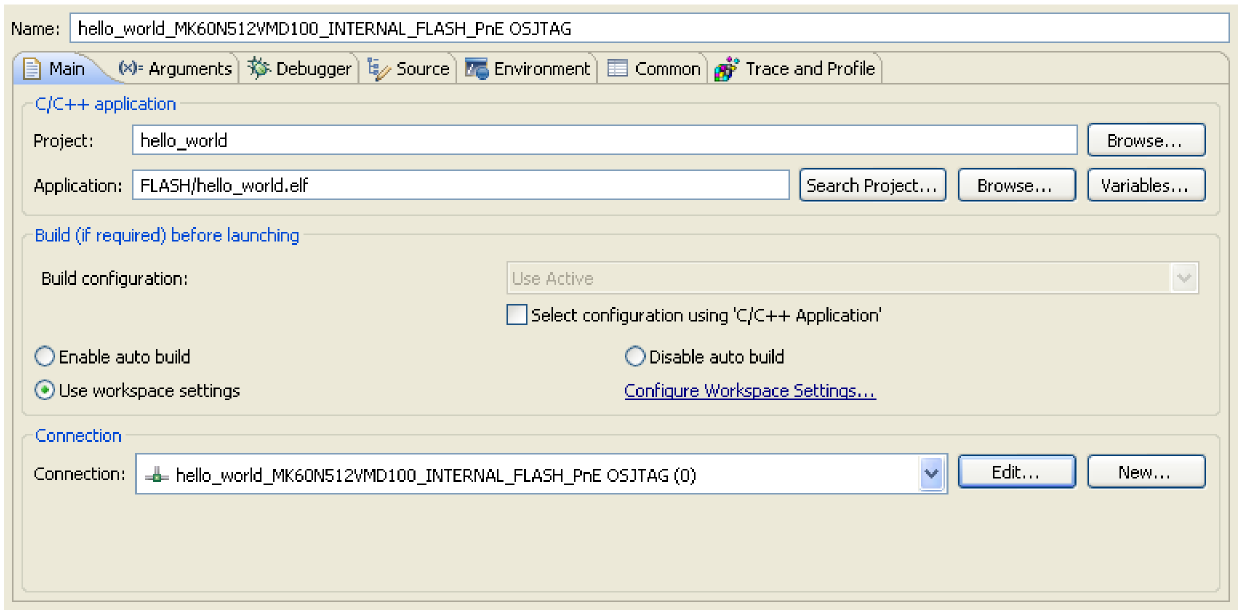
2. Select **Run** > **Debug Configurations** .

   The **Debug Configurations** dialog box appears.

3. Select the configuration you want to use. For this case, select **CodeWarrior Download > hello_world_MK60N512VMD100_INTERNAL_FLASH_PnE OSJTAG** .

   You may see the error message "The application file specified in the launch configuration does not exist".

4. Select the **Main** tab and specify the right application file for the debugger, to fix the error.


Name: hello_world_MK60N512VMD100_INTERNAL_FLASH_PnE OSJTAG

C/C++ application

Project: hello_world

Application: FLASH/hello_world.elf

Build (if required) before launching

Build configuration: Use Active

[ ] Select configuration using 'C/C++ Application'

( ) Enable auto build   ( ) Disable auto build

[✓ Use workspace settings]   Configure Workspace Settings...

Connection

Connection: hello_world_MK60N512VMD100_INTERNAL_FLASH_PnE OSJTAG (0)

Figure 2-8. Debug Configuration - Main

5. Click **Debug** button.

   The application will be downloaded to the device and debugger will start.

2.2 Custom Assembly Code

There are significant differences between the Freescale assembler and the GCC assembler, with respect to pragma's, and directives. They both recognize the standard ARM assembly code and generate functional binaries. For more information, refer to the Guidelines chapter.

Chapter 3
Guidelines

Using GCC to build existing CodeWarrior applications requires certain mappings and changes to the source code. Most of the mapping can be placed under a "prefix" file and can be included as a part of the project. This will avoid making changes to the application sources. However, some instances will require changes to the source code.

This chapter highlights the areas where mapping is possible and instances where code changes are necessary.

3.1 Built-in Macros

The built-in macros for CodeWarrior and GCC are similar.

Table 3-1. Built-in Macros for CodeWarrior and GCC

CodeWarrior	GCC
`__CWCC__`	`__GNUC__`
`__cplusplus`	`__cplusplus`
`__STDC__`	`__STDC__`
`__STDC_VERSION__`	`__STDC_VERSION__`
`__STDC_HOSTED__`	`__STDC_HOSTED__`

3.2 Directives

CodeWarrior allows certain language options and some compiler options that can be tested during build time. The options appear in the form of directives such as `__option`, `__supports`, `__has_feature`, and `__has_intrinsic`. GCC does not support testing of options during build time. But by using some of the GCC macros we will be able to imitate the CW directive. The porting process requires replacing `__option(x)` with a `#if` of the GCC equivalent macro.

The following directives should be mapped.

- `__option(Arg)`- Refer Mapping __option(Arg) Directive of CodeWarrior Driver to GCC
- `__supports(x,y)`- Refer Mapping __supports(Arg) directive of CodeWarrior Driver to GCC
- `__has_feature`- Refer Mapping __has_feature directive of CodeWarrior Driver to GCC
- `__has_intrinsic(x)`- Refer Mapping __has_intrinsic(x) directive of CodeWarrior Driver to GCC

3.2.1 Mapping __option(Arg) Directive of CodeWarrior Driver to GCC

`__option()` directive can be mapped to GCC by defining the macro `#define __option(x) x.`

Table 3-2. Treatment of x in #define __option(x) x

CodeWarrior	GCC
`__option(unsigned_char)`	`#define unsigned_char __CHAR_UNSIGNED__`
`__option(longlong)`	`#if __LONG_LONG_MAX__ #define longlong 1 #else#define longlong 0#endif`
`__option(C99)`	`__option(C99) #define C99 ( __STDC_VERSION__ >= 199901L)`
`__option(little_endian)`	`#define little_endian __ARMEL__`
`__option(dont_inline)`	`#define dont_inline __NO_INLINE__`
`__option(ANSI_strict)`	`#define ANSI_strict __STRICT_ANSI__`
`__option(k63d)`	`#define k63d 0`
`__option(bool)`	`#define bool 0`
`__option(wchar_type)`	`#ifdef __WCHAR_TYPE__#define wchar_type 1#else#define wchar_type 0#endif`
`__option(mpwc_newline)`	`#define mpwc_newline 0`
`__option(optimize_for_size)`	`#define optimize_for_size __OPTIMIZE_SIZE__`
`__option(rsqrt)`	`#define rsqrt 0`
`__option(floatingpoint)`	`#ifdef __NOFLOAT__#define floatingpoint 0#else#define floatingpoint 1#endif`
`__option(sfp_emulation)`	`#define sfp_emulation _SOFT_FLOAT`
`__option(e500_floatingpoint)`	`#define e500_floatingpoint 0`
`__option( e500v2_floatingpoint)`	`#define e500v2_floatingpoint 0`

Table continues on the next page...

Table 3-2. Treatment of x in #define __option(x) x (continued)

CodeWarrior	GCC
`__option(__thumb)`	`#define __thumb __thumb__`
`__option(exceptions)`	`#define exceptions __EXCEPTIONS`

3.2.2 Mapping __supports(Arg) directive of CodeWarrior Driver to GCC

`__supports(x,y)` directive can be mapped to GCC by defining the macro `#define __supports(x,y) 0`.

3.2.3 Mapping __has_feature directive of CodeWarrior Driver to GCC

`__has_feature` directive can be mapped to GCC by defining the macro `#define __has_feature 0`.

3.2.4 Mapping __has_intrinsic(x) directive of CodeWarrior Driver to GCC

`__has_intrinsic(x)` directive can be mapped to GCC by defining the following macro

```
#define __has_intrinsic(x) 0.
```

3.3 Pragmas

Pragma support in GCC is minimal and thus you are recommended to use function attributes. The following are the common pragmas that GCC supports with their CodeWarrior equivalent, if applicable.

Table 3-3. Equivalent Pragmas

CodeWarrior	GCC
`#pragma opt_classresults`	
`#pragma BeginErrorCheck`	
`#pragma EndErrorCheck`	
`#pragma cpp_extensions`	
`#pragma optimization_level 0|1|2|3|4`	`#pragma GCC optimize ("string"...)`
`#pragma push`	`#pragma GCC push_options`
`#pragma pop`	`#pragma GCC pop_options`
	`#pragma GCC reset_options`
`#pragma push_marcro("macro_name")`	`#pragma push_macro("macro_name")`
`#pragma pop_macro("macro_name")`	`#pragma pop_macro("macro_name")`
`#pragma message ("string")`	`#pragma message string`
	`#pragma weak symbol`
	`#pragma weak symbol1 = symbol2`
`#pragma pack(n)`	`#pragma pack(n)`
	`#pragma long_calls`
	`#pragma no_long_calls`
	`#pragma long_calls_off`

3.4 Function Attributes

The following are the function attribute mappings. GCC supports all function attributes that CodeWarrior supports.

- `__attribute__((noinline))`
- `__attribute__((nothrow))`
- `__attribute__((weak))`
- `__attribute__((naked))`
- `__attribute__((noreturn))`
- `__attribute__((const))`
- `__attribute__((always_inline))`
- `__attribute__((aligned())))`
- `__attribute__((section()))`

3.5 Macro

CodeWarrior unlike GCC supports string replacement in a macro. Therefore to support string replacement porting will involve explicitly using the complete string.

Table 3-4. String Replacement

CodeWarrior	GCC
`# define MOD_INCLUDE(str) <str##.h>`	`#ifdef __GNUC__# define MOD_INCLUDE(str) < str.h>#else# define MOD_INCLUDE(str) < str##.h>#endif`

3.6 Command-line Options

The following are the compiler command-line options for CodeWarrior and GCC.

Table 3-5. Compiler Options

CodeWarrior	GCC
`-proc cortex-m4`	`-mcpu=cortex-m4`
`-proc cortex-m0`	`-mcpu=cortex-m0`
`-prefix file`	`-include file`
`-char [un]signed`	`-f[un]signed-char`
`-Cpp_exceptions on|off`	`-f[no-]exceptions`
`-RTTI on|off`	`-f[no-]rtti`
`-O0,O1,O2,O3`	`-O0,O1,O2,O3`
`-Os`	`-Os`
`-big`	`-mbig-endian`
`-little`	`-mlittle-endian`
`-fp soft`	`-msoft-float`
`-fp vfpv4`	`-mfpu=fpv4-sp-d16 -mfloat-abi=hard`
`-[no]interworking`	`-m[no-]thumb-interwork`
`-thumb`	`-mthumb`
`-g`	`-g`

The following are the linker command-line options for CodeWarrior and GCC.

Table 3-6. Linker Options

CodeWarrior	GCC
`-dead[strip]`	`compiler: -ffunction-sections -fdata-sections / link: --gc-sections`
`-main symbol`	`-e symbol`
`-map file`	`-Map=file`
`{file.lcf}`	`-T{file.ld}`

3.7 Coding Notes

The following are the coding notes when porting from CodeWarrior to GCC.

1. User-defined sections declaration:
   - **CodeWarrior**

     ```
     _declspec(section ".foo") void foo();
     ```
   - **GCC**

     ```
     __attribute__((section(".foo"))) void foo();
     ```
2. GCC does not support the use of variable for global array initialization. However, this support is available in CodeWarrior.
   - **CodeWarrior**

     ```
     const int var=5;

     int arr[var] = {1,2,3,4,var}; // error on gcc
     ```
   - **GCC**

     ```
     int arr[5] = {1,2,3,4,5};
     ```
3. GCC does not support the use of variable for constant initialization. However, this support is available in CodeWarrior.
   - **CodeWarrior**

     ```
     const int var=1;

     const int var2 = var;
     ```
   - **GCC**

     ```
     const int var2 = 1;
     ```
4. `-n` option should be passed to gcc linker.

   The program segment gets aligned to its page size which is 0x8000 by default.

This can be removed by `-n` option. This will disable the page alignment and use the section alignment on program segment.

5. `-fextended-` identifiers are used in GCC to accept universal characters. However, in CodeWarrior universal characters are accepted by default.
6. Changes required in porting CW assembly .s files to GCC syntax:
   a. Use `.global` instead of `.public`.
   b. Comments to begin with `@`.
   c. Put `.syntax` unified near the start of your assembly source. This turns on Unified Assembly Language, which is required to get all the features of Thumb-2.
   d. `sreg01 .textequ "r1"` change to `#define sreg01 r1`.
   e. Use option `-mimplicit-it=always`.

3.8 LCF Porting

Key points to note while porting CodeWarrior (CW) Linker Command File (LCF) to GCC Linker Script File or GCC Linker Description (LD) File.

1. LCF comments start with #. Use C style (/* */) comments in case of LD file.

   Example:

   CW LCF: #Default linker command file.

   GCC LD: /*Default linker command file.*/

2. Access flags (RWX) are same between CW LCF and GCC LD. However GCC LD supports more access flags.

   CW LCF:

   MEMORY { segmentName (accessFlags) : ORIGIN = address, LENGTH = length [> fileName] } segment_1 (RWX): ORIGIN = 0x80001000, LENGTH = 0x19000 [>filename] is not compliant with GNU syntax

   GCC LD:

   MEMORY { segmentName [(attr)] : ORIGIN = address, LENGTH = length ... }
3. **AFTER(m_text)** does not work with GCC LD.
4. **KEEP_SECTION** : CW LCF command 'KEEP_SECTION' is supported outside 'SECTIONS'. GCC LD equivalent command is 'KEEP'. However this should be used inside 'SECTIONS' directive.

   Example:

CW LCF: **KEEP_SECTION** { .vectortable } GCC LD: SECTIONS { .interrupts : { __vector_table = .; **KEEP** (*(.vectortable)) . = ALIGN (0x4); } > m_interrupts }

5. GCC LD file should have section name and ':' separated by a space.

   Example:

   CW LCF: **.app_text:** GCC LD: **.app_text :** { }
6. Add *** (.text*)** and *** (.rodata*)** in .app_text in case of GCC LD.

   Example:

   .app_text : { . = ALIGN(0x4) ; * (.init) * (.text) * (.text*) . = ALIGN(0x8) ; * (.rodata) * (.rodata*) . = ALIGN(0x4) ; ___ROM_AT = .; } > m_text
7. Remove ALIGNALL command as it is not supported in GCC LD.
8. In case of GCC LD, allow a space between '.' And '=' in case of '.= ALIGN'.

   Example:

   CW LCF: . **= ALIGN(0x4);** GCCC LD: **.= ALIGN(0x4);**
9. >> is not supported in GCC LD.

   Example:

   CW LCF: .bss : { } >> m_data GCC LD: .bss : { } > m_data
10. **WRITEW** command should be replaced by **LONG** .

    Example:

    CW LCF: **WRITEW(0);** GCC LD: **LONG(0);**
11. ARM.extab section in CW LCF should be ported to GCC LD.

    Example:

    CW LCF: *(.ARM.extab) . = ALIGN(0x4) ; __exception_table_start__ = .; EXCEPTION __exception_table_end__ = .; . = ALIGN(0x4) ; GCC LD: __exidx_start = .; *(.ARM.exidx*) __exidx_end = .;
12. Remove **STATICINIT** in case of GCC LD.
13. Define segment for sections like `.ctors, .dtors, .preinit_array, init_array, .fini_array`, in GCC LD.

    Example:

    .ctors : { __CTOR_LIST__ = .; KEEP (*crtbegin.o(.ctors)) KEEP (*(EXCLUDE_FILE (*crtend.o) .ctors)) KEEP (*(SORT(.ctors.*))) KEEP (*(.ctors)) __CTOR_END__ = .; } > m_text .dtors : { __DTOR_LIST__ = .; KEEP (*crtbegin.o(.dtors)) KEEP (*(EXCLUDE_FILE (*crtend.o) .dtors)) KEEP

(*(SORT(.dtors.*))) KEEP (*(.dtors)) __DTOR_END__ = .; } >
m_text .preinit_array : { PROVIDE_HIDDEN (__preinit_array_start = .); KEEP
(*(.preinit_array*)) PROVIDE_HIDDEN (__preinit_array_end = .); } >
m_text .init_array : { PROVIDE_HIDDEN (__init_array_start = .); KEEP
(*(SORT(.init_array.*))) KEEP (*(.init_array*)) PROVIDE_HIDDEN
(__init_array_end = .); } > m_text .fini_array : { PROVIDE_HIDDEN
(__fini_array_start = .); KEEP (*(SORT(.fini_array.*))) KEEP (*(.fini_array*))
PROVIDE_HIDDEN (__fini_array_end = .); ___ROM_AT = .; } > m_text

14. INCLUDE filename

    CW LCF:

    .my_text{ INCLUDE filename }>my_text

    GCC LD:

    Equivalent syntax in GCC LD,

    INPUT(file, file, ...) INPUT(file file ...)

    Command line options to include binary file [default section as .data.]

    NOTE

    Binary filename: data.raw, Format: binary

    a. Create a stationary project.
    b. Go to **Properties > C/C++Build > Settings > ARM Ltd. Windows GCC C Linker > Miscellaneous > Other flags**
    c. Add the following options. `-Wl,-b,binary,"C:/data.raw",-b, elf32-littlearm`

       This makes linker to treat data.raw as raw binary and resume to elf32-littlearm for subsequent objects.

       The final elf will contain the following symbols for the sources to manipulate the data:

       ```
       _binary_C___data_raw_start _binary_C___data_raw_end _binary_C__data_raw_size
       ```

       The default linker section will be .data.

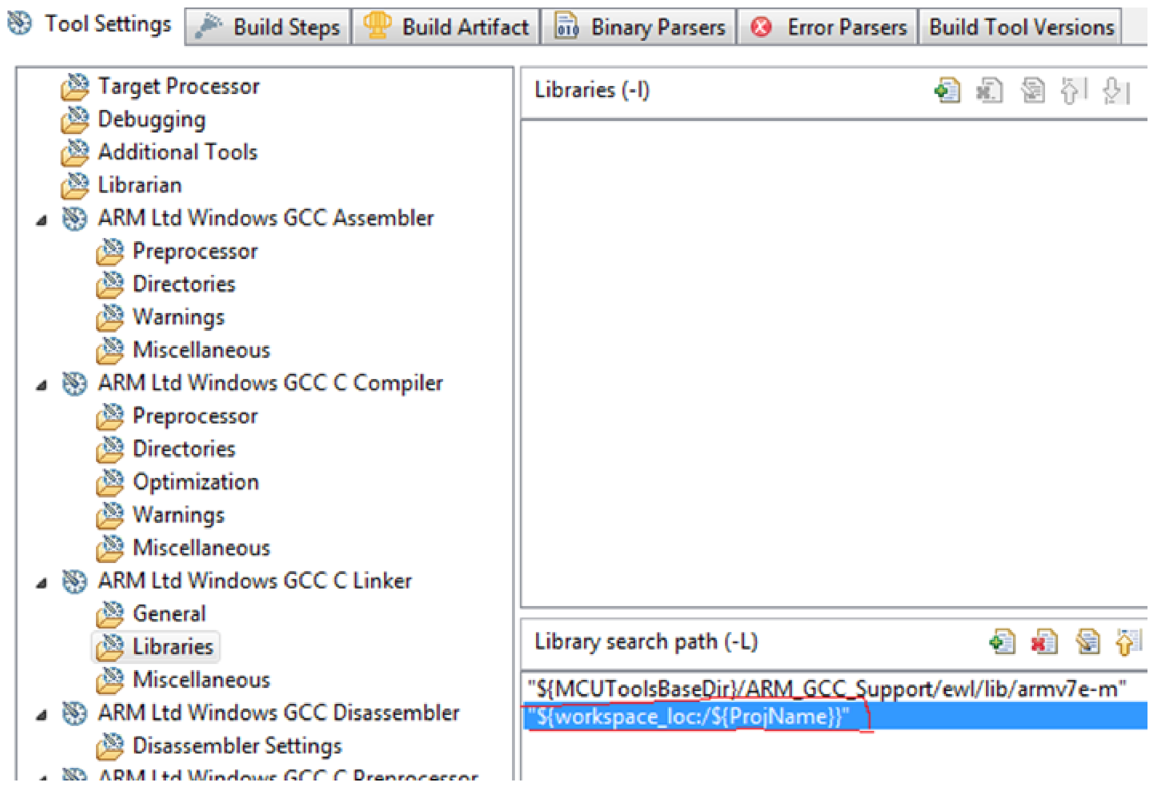
    To place the raw data in a specified linker section, define the section in the Linker command file.

    a. Define the following section.

       ```
       MEMORY{
       ..
       ```

```
m_srecord     (RX) : ORIGIN = 0x0000C000, LENGTH = 0x00004000/*define its memory
segment*/

}

TARGET(binary)/* specify the file format of binary file */

INPUT (data.raw)/* provide the file name */

OUTPUT_FORMAT(default)/* restore the out file format */

/* Define output sections */

SECTIONS

{


.srecord :

  {



 data.raw (.data)

    . = ALIGN (0x4);

  } > m_srecord

.text:

{

..

}>m_text


}
```

b. Add the path of the file added in the previous step to **ARM Ltd. GCC C Linker->Libraries->Library search path (-L)**



```
"${MCUToolsBaseDir}/ARM_GCC_Support/ewl/lib/armv7e-m"
"${workspace_loc:/${ProjName}}"
```

Figure 3-1. ARM Ltd. GCC C Linker->Libraries->Library search path (-L)

INCLUDE filename is also supported in GCC LD but for a different purpose INCLUDE filename Include the linker script filename at this point. The file will be searched for in the current directory, and in any directory specified with the '-L' option. You can nest calls to INCLUDE up to 10 levels deep. You can place INCLUDE directives at the top level, in MEMORY or SECTIONS commands, or in output section descriptions.

3.9 Miscellaneous Notes

1. Common options when building with GCC + EWL.

   ```
   compile: -nostdinc -nostdinc++ -include {ewl prefix} -ffunction- sections -fdata-
   sections
   ```

   ```
   link: -nostartfiles -nostdlib -nodefaultlib --gc-sections -T{lcf file} --start-group -lc
   -lc++ -lgcc -lhosted -lrt -lsupc++ --end- group
   ```

2. Pick the following default libraries and paths.

If you use the linker tool, (for example `powerpc-eabi-ld`) then pass the library paths, start-up files and the libraries manually.

Therefor the recommended option is to use the compiler driver (for example `powerpc-eabi-gcc`) for linking the required object files and libraries to generate the executable. The compiler driver will search and pick the required libraries (`libc.a, libgcc.a`) automatically.

3. C++ default libraries are not picked automatically even after using the compiler driver.

   Check if the user is using the correct driver (for example `powerpc-eabi-gc`c for C programs and `powerpc-eabi-g++` for C++ programs)

4. Error: Multiple definition error of start up file symbols [`` `__init', `__fini' `` etc].

   By default when linking with the compiler driver, it picks the default start-up code. To avoid using the default start up file, pass ' `-nostartfiles`' option to the compiler driver while linking.

5. Error: Skipping incompatible '<library>' when searching for '<lib.a>'.
   - Check if the user is compiling/linking with correct compiler options to pick the proper 32bit/64bit/soft-float bit libraries.
   - Check if the user is using proper 32/64bit linker description file (*.ld/*.lcf).
6. How to resolve inter-dependency between libraries.

   By default, the linker usually resolves the symbols from left to right. So if any library/object (e.g libA) is dependent on another library/object (e.g libB) then we should pass libA first. And if there is inter-dependency between these libraries, then pass these libraries between '-start-group' and '--end- group'.

7. To strip the unused symbols while generating the executable, use the following options.
   - Compiler option: `-fdata-sections -ffunction-sections` [also, set compiler optimization levels]
   - Linker option: `-Wl, --gc-sections -Wl, --strip-all`
8. Pass the appropriate library linking option `"-static"` (required for bare metal applications) or `"-shared"`.
9. Pass explicit driver option `"-std=c99"` for linking and building C99 applications.

3.10 Target Specific Notes

The following are GCC predefined macros for ARM.

Table 3-7. GCC Predefined macros for ARM

Predefined Macro	Description
__arm__	Always on for gcc arm.
__APCS_32	Always on for gcc arm.
__thumb__	Thumb is on.
__thumb2__	Thumb2 is on.
__ARMEB__	ARM big endian mode.
__ARMEL__	ARM little endian mode.
__THUMBEB__	Thumb big endian mode.
__THUMBEL__	Thumb little endian mode.
__SOFTFP__	Soft fp enabled.
__VFP_FP__	vfpu enabled.
__ARM_NEON__	ARM NEON enabled.
__THUMB_INTERWORK__	Interworking enabled.
__ARM_EABI__	Targeting arm aeabi
__ARM_ARCH_6M__	Targeting v6m architecture (cortex-m0)
__ARM_ARCH_7M__	Targeting v7m architecture (cortex-m4)

3.11 Rebuilding EWL Libraries from IDE

The following are the steps to rebuild the EWL Libraried from the IDE.

1. Open the Codewarrior IDE.
2. Import or drag and drop the .project which is present in ewl folder.

   Example: If your Kinetis product layout is in the folder `C:\Freescale\CW MCU v10.x`, then drag and drop `C:\Freescale\CW MCU v10.x\MCU\ARM_GCC_Support\ewl\.project.`
3. Right-click on the opened 'ewl: ARM GCC' project and select 'Clean Project' to clean and 'Build Project' to build the libraries.

3.12 Rebuilding EWL Libraries from Command Prompt

All library files (viz. libm.a, libc.a ...etc) are present in the ewl\lib folder.

The following steps help rebuild the EWL library files on the command prompt.

1. Open a DOS command prompt.
2. Change your working directory to the ewl folder, for example,

   cd C:\Freescale\CW MCU v10.x\MCU\ARM_GCC_Support\ewl

3. Define the ARM_TOOLS, PLATFORM and VENDOR environment variables. For example, if your Kinetis product layout is in the folder C:\Freescale\CW MCU v10.x then you can do:

   ```
   ..\ewl>set ARM_TOOLS=C:\Freescale\CW MCU v10.x\Cross_Tools\arm-none-eabi-gcc-4_7_3

   ..\ewl>set PLATFORM=arm

   ..\ewl>set VENDOR=GCC
   ```

4. '`make`' to build all libraries:

   You can find make utility at '`\CW MCU v10.x\gnu\bin\`'

   NOTE

   The make command rebuilds all the libraries required for cortex-m0+ and cortex-m4 architectures.

5. You can rebuild individual libraries for individual target.

   For example: If you want to build `libc.a` for `cortex-m0+`, do: `make TARGET=/armv6-m/ libc` .

   This builds libc.a under lib/armv6-m folder. Also you can build only C/C++/runtime libraries.

6. To build C libraries, cd to EWL_C and perform following command: `make -f EWL_C.GCC.mak`

   This builds all C libraries such as `libc.a, libc99.a and libm.a for cortex-m0+ and cortex-m4` architectures at `armv6-m` and `armv7e-m` respectively.

7. To build C++ libraries, cd to EWL_C++ directory and do: `make -f EWL_C++.GCC.mak`

   This builds C++ libraries such as libc++.a and libstdc++.a for cortex-m0+ and cortex-m4 architectures.

8. To build runtime libraries, cd to EWL_Runtime and do: `make -f EWL_Runtime.GCC.mak`

   This builds runtime libraries such as librt.a, libhosted.a, libuart.a, __arm_start.o, __arm_start- hosted.o, __arm_end.o and __arm_end-hosted.o for the both architectures.

9. To clean the libraries: `make clean`

NOTE

The clean command cleans all the libraries required for cortex-m0+ and cortex-m4 architectures.

3.13 Stream Buffer

EWL for GCC ARM is configured by default to line buffer, `_IOLBF` (line buffering: newline triggers automatic flush).

This can be modified at runtime through `setvbuf()` function.

Example:

```
setvbuf(stdout, NULL, _IONBF, 0); // _IONBF (direct write always used)

printf("Hello from Kinetis");
```

The default behavior can also be changed by rebuilding the EWL library with `EWL_BUFFERED_CONSOLE` turned off.

3.14 References

- GCC C language extension

  http://gcc.gnu.org/onlinedocs/gcc/C-Extensions.html
- GCC C++ language extension

  http://gcc.gnu.org/onlinedocs/gcc/C_002b_002b-Extensions.html#C_002b_002b-Extensions
- GCC common Pre-defined macros:

  http://gcc.gnu.org/onlinedocs/cpp/Common-Predefined-Macros.html
- GCC Pragmas:

  http://gcc.gnu.org/onlinedocs/gcc/Pragmas.html#Pragmas
- GCC Function attributes:

  http://gcc.gnu.org/onlinedocs/gcc/Function-Attributes.html#Function-Attributes

Index

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

Document Number: [No Doc ID]
Rev. April 15, 2013