CodeWarrior Development Studio for Microcontrollers V10.x Kinetis GCC Build Tools Reference Manual

Document Number: CWMCUGCCCMPREF
Rev 10.6, 02/2014

Contents

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

Chapter 1
Introduction

Chapter 2
Build Properties for ARM (GCC)

Chapter 3
EWL for Kinetis Development

Chapter 4
Using GCC to Build Legacy Kinetis Projects

Chapter 5
Porting Guidelines

Chapter 1
Introduction

The Kinetis GCC Build Tools Reference Manual for Microcontrollers describes usage of GCC build tools with Kinetis in CodeWarrior for Microcontrollers, the EWL libraries, and porting legacy Freescale Kinetis projects to use GCC.

NOTE

The GNU documentation for GCC ARM is available at `<CWInstallDir>\Cross_Tools\arm-none-eabi-gcc- X_Y_Z\share\doc\gcc-arm-none-eabi`.

The guide includes the following Chapters.

- Build Properties for ARM (GCC)
- EWL for Kinetis Development
- Using GCC to Build Legacy Kinetis Projects
- Porting Guidelines

Chapter 2
Build Properties for ARM (GCC)

The **Properties for** <*project*> dialog box shows the corresponding build properties for ARM project that supports the GCC toolchain. The properties that you specify in the **Tool Settings** panels apply to the selected build tool on the **Tool Settings** page of the **Properties for** <*project*> dialog box.

The following listed are the build properties specific to developing software for `ARM (GCC)` C Compiler:

Table 2-1. Build Properties for ARM (GCC)

Build Tool	Build Properties Panels
ARM Ltd Windows GCC C Compiler	ARM Ltd Windows GCC C Compiler > Preprocessor
	ARM Ltd Windows GCC C Compiler > Directories
	ARM Ltd Windows GCC C Compiler > Optimization
	ARM Ltd Windows GCC C Compiler > Warnings
	ARM Ltd Windows GCC C Compiler > Miscellaneous

2.1 ARM Ltd Windows GCC C Compiler

Use the **ARM Ltd Windows GCC C Compiler** panel to specify the compiler options that are specific to the ARM (GCC).

NOTE

The list of tools presented on the **Tool Settings** page can differ, based upon the toolchain used by the project.

The following figure shows the **ARM Ltd Windows GCC C Compiler** panel.

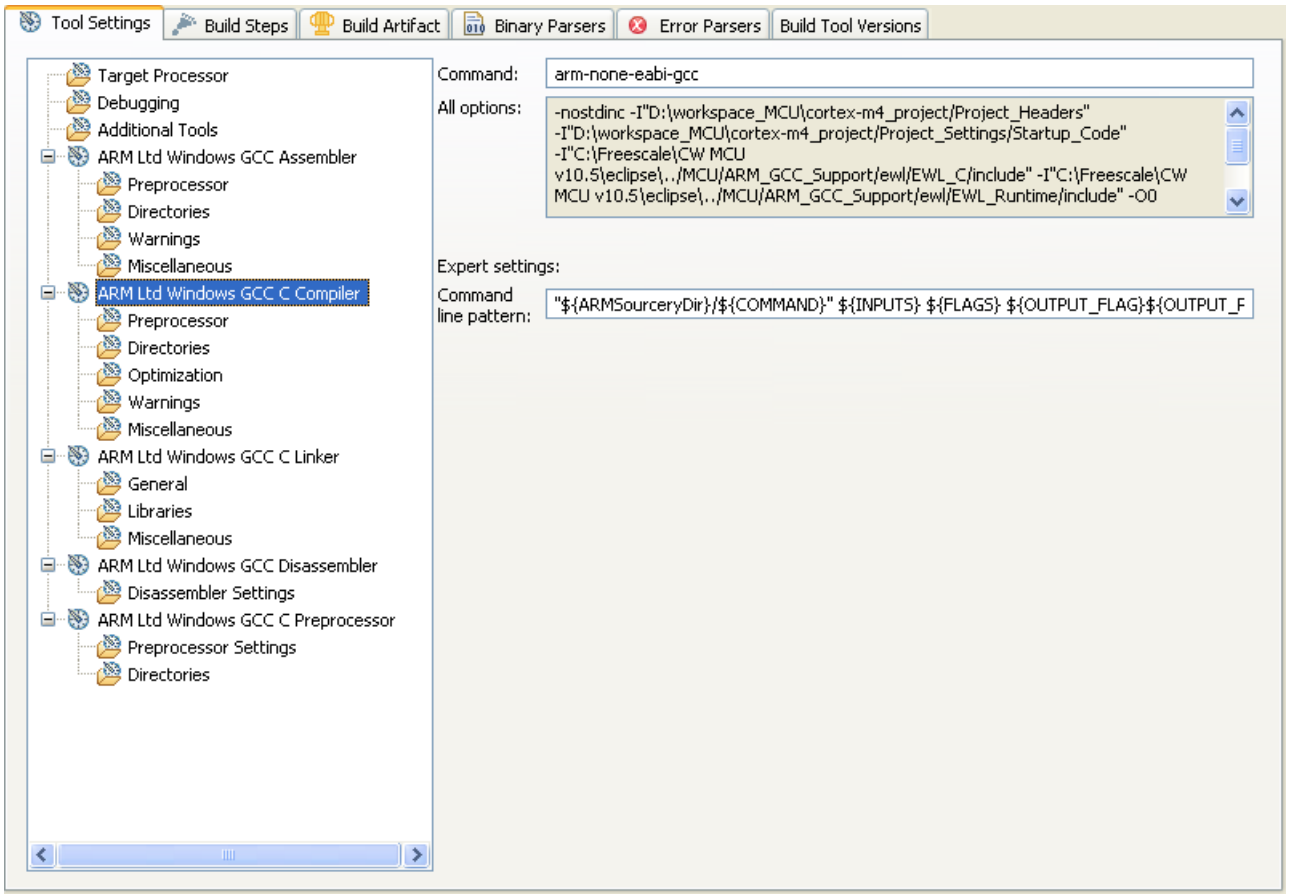

Figure 2-1. Tool settings - ARM Ltd Windows GCC C Compiler

The table below lists and describes the various options available on the **ARM Ltd Windows GCC C Compiler** panel.

Table 2-2. Tool Settings - ARM Ltd Windows GCC C Compiler

Option	Description
Command	Shows the location of the compiler executable file. Default: `arm-none-eabi-gcc`
All options	Shows the actual command line the compiler is called with.
Expert settings	Shows the expert settings command line parameters. Default: `"${ARMSourceryDir}/${COMMAND}" ${INPUTS} ${FLAGS} ${OUTPUT_FLAG}${OUTPUT_PREFIX}${OUTPUT}"`
Command line pattern	

2.2 ARM Ltd Windows GCC C Compiler > Preprocessor

Use the **Preprocessor** panel to specify preprocessor behavior. You can specify whether to search system directories or preprocess only based on the options available in this panel.

The following figure shows the **Preprocessor** panel.

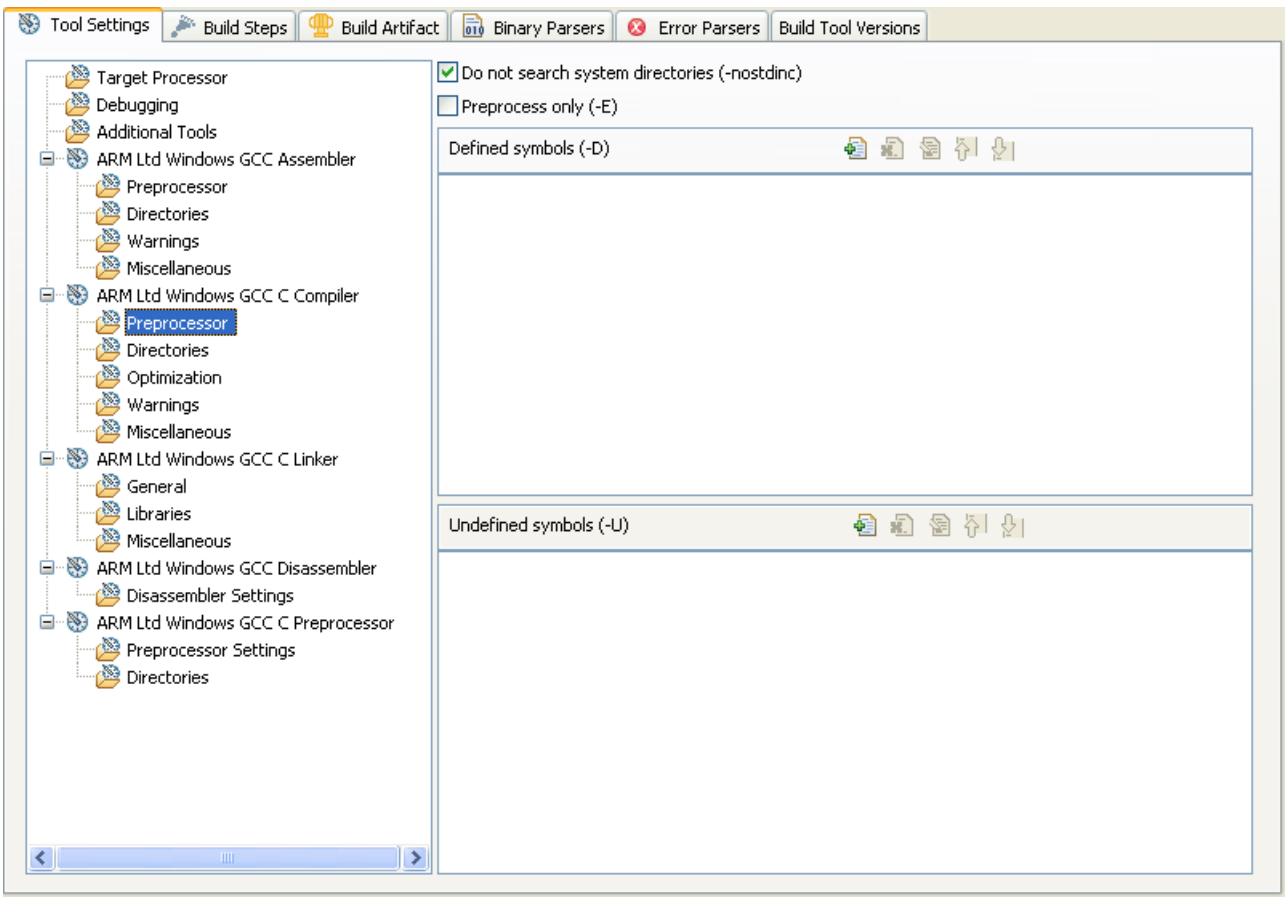

Figure 2-2. Tool settings - ARM Ltd Windows GCC C Compiler > Preprocessor

The table below lists and describes the various options available on the **Preprocessor** panel.

Table 2-3. ARM Ltd Windows GCC C Compiler > Preprocessor

Option	Description
Do not search system directories (-nostdinc)	Check this option to specify the `-nostdinc` command to the compiler. The compiler does not search the system directories. By *default* this checkbox is clear. The compiler performs a full search that includes the system directories
Preprocess only (-E)	Check this option to specify the `-E` command to the compiler. The compiler tells the command-line tool to preprocess source files. By *default* this checkbox is clear. The compiler does not preprocess source files.

Table continues on the next page...

Table 2-3. ARM Ltd Windows GCC C Compiler > Preprocessor (continued)

Option	Description
Defined symbols (`-D`)	Use this option to specify the substitution strings that the assembler applies to all the assembly-language modules in the build target. Enter just the string portion of a substitution string. The IDE prepends the `-D` token to each string that you enter. For example, entering `opt1 x` produces this result on the command line: `-Dopt1 x`**Note:** This option is similar to the `DEFINE` directive, but applies to all assembly-language modules in a build target.
Undefined symbols (`-U`)	Undefines the substitution strings you specify in this panel.

2.3 ARM Ltd Windows GCC C Compiler > Directories

Use the **Directories** panel to specify the directories paths. The following figure shows the **Directories** panel.

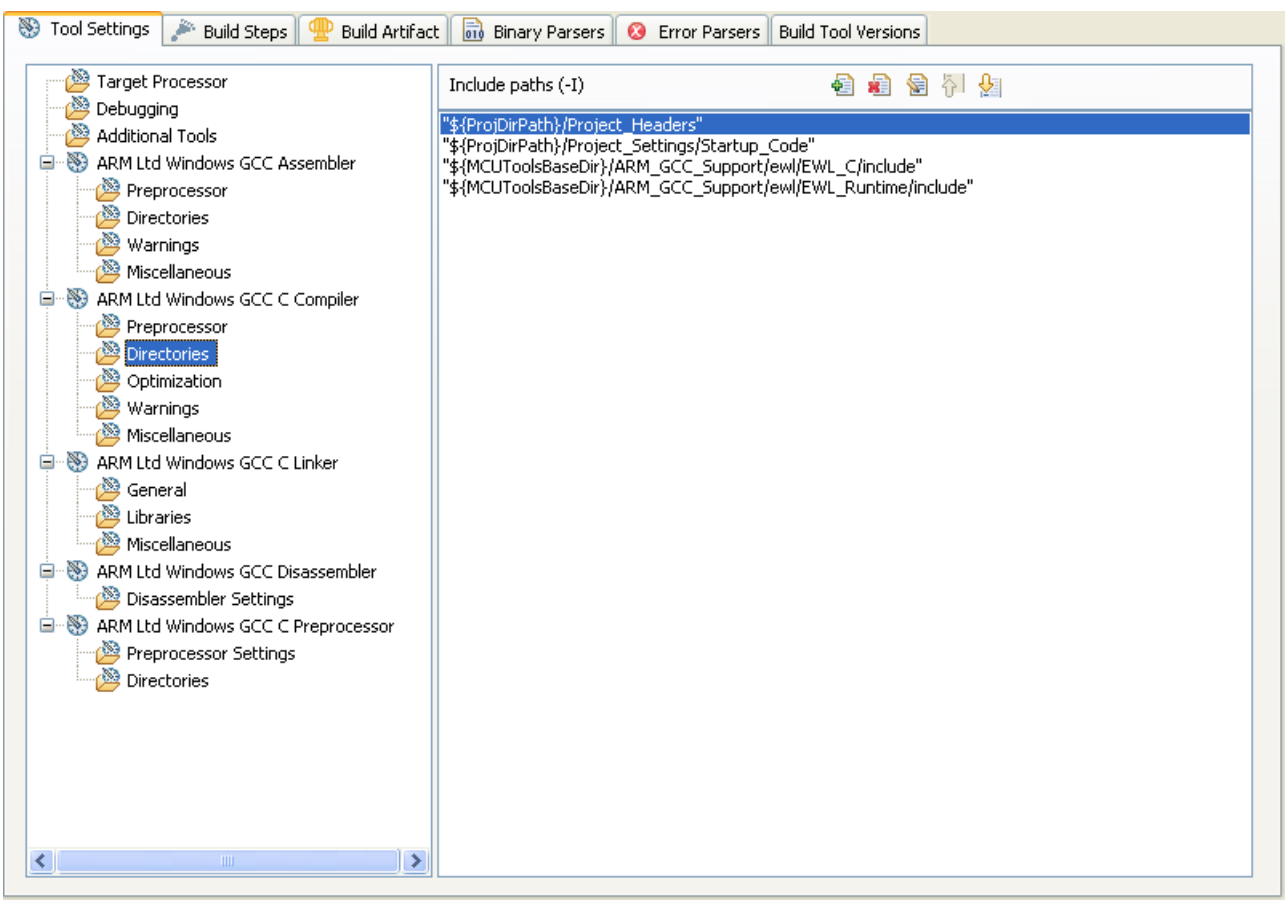

Figure 2-3. Tool settings - ARM Ltd Windows GCC C Compiler > Directories

The table below lists and describes the various options available on the **Directories** panel.

Table 2-4. ARM Ltd Windows GCC C Compiler > Directories

Option	Description
Include paths (`-I`)	This option changes the build target's search order of access paths to start with the system paths list. The compiler can search `#include` files in several different ways. You can also set the search order as follows: For include statements of the form `#include"xyz"`, the compiler first searches user paths, then the system paths For include statements of the form `#include<xyz>`, the compiler searches only system paths This option is global.

2.4 ARM Ltd Windows GCC C Compiler > Optimization

Use the **Optimization** panel to control compiler optimizations. Compiler optimization can be applied in either global or non-global optimization mode. You can apply global optimization at the end of the development cycle, after compiling and optimizing all source files individually or in groups.

The following figure shows the **Optimization** panel.

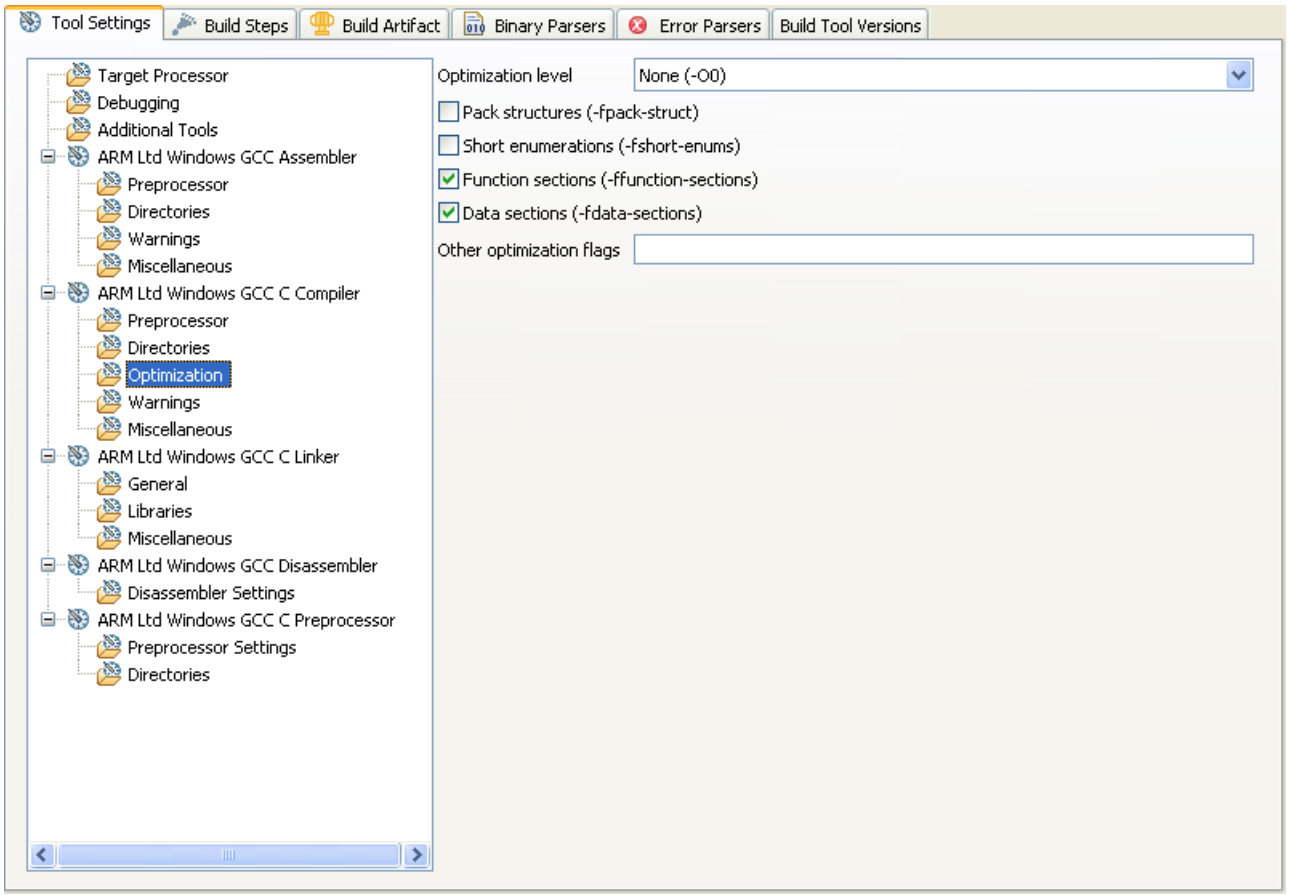

Figure 2-4. Tool settings - ARM Ltd Windows GCC C Compiler > Optimization

The table below lists and describes the various options available on the **Optimization** panel.

Table 2-5. ARM Ltd Windows GCC C Compiler > Optimization

Option	Description
Optimization Level	Use this option to specify the optimizations that you want the compiler to apply to the generated object code. The options available are: • **None (-O0)** - Disable optimizations. This setting is equivalent to specifying the -O0 command-line option. The compiler generates unoptimized, linear assembly-language code. • **Optimize (-O1)** - The compiler performs all target-independent (that is, non-parallelized) optimizations, such as function inlining. This setting is equivalent to specifying the -O1 command-line option. The compiler omits all target-specific optimizations and generates linear assembly-language code. • **Optimize more (-O2)** - The compiler performs all optimizations (both target-independent and target-specific). This setting is equivalent to specifying the -O2 command-line option. The compiler outputs optimized, non-linear, parallelized assembly-language code.

Table continues on the next page...

Table 2-5. ARM Ltd Windows GCC C Compiler > Optimization (continued)

Option	Description
	• **Optimize most (-O3)** - The compiler performs all the level 2 optimizations, then the low-level optimizer performs global-algorithm register allocation. This setting is equivalent to specifying the -O3 command-line option. At this optimization level, the compiler generates code that is usually faster than the code generated from level 2 optimizations. • **Optimize for size (-Os)** - Optimize for size. `-Os` enables all `-O2` optimizations that do not typically increase code size. It also performs further optimizations designed to reduce code size. This setting is equivalent to specifying the - `Os` command-line option.
Pack structures (-fpack-struct)	Packed data structures are supported in the compiler with the keyword `__packed` or `__attribute__((packed))`. There is no code generation support for accessing un-aligned, packed data members.Users should exercise caution when accessing packed data structures because data might not be aligned.
Short enumerations (-fshort-enums)	Check to use short enumerated constants and is equivalent to `-fshort-enums`.
Function sections (-ffunction-sections)	Check to use function sections and is equivalent to `-ffunction-sections`.
Data sections (-fdata-sections)	Check to use short data sections and is equivalent to `-ffunction-sections`.
Other optimization flags	Specifies individual optimization flag that can be turned ON/ OFF based on the user requirements.

2.5 ARM Ltd Windows GCC C Compiler > Warnings

Use the Warnings panel to control how the compiler reports the error and warning messages. The following figure shows the **Warnings** panel.

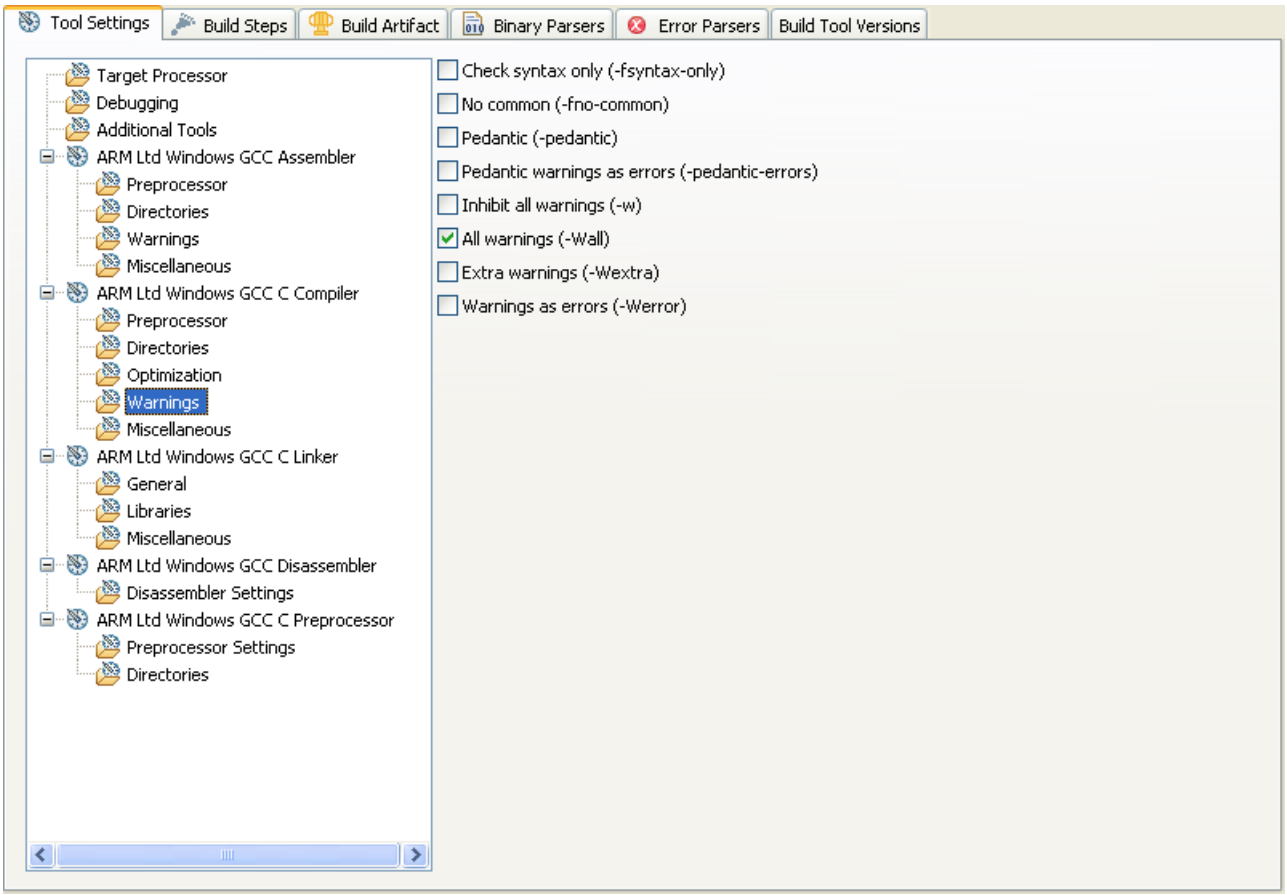

Figure 2-5. Tool settings - ARM Ltd Windows GCC C Compiler > Warnings

The following table lists and describes the various options available on the **Warnings** panel.

Table 2-6. ARM Ltd Windows GCC C Compiler > Warnings

Option	Description
Check syntax only (-fsyntax-only)	Check this option if if you want to check the syntax of commands and throw a syntax error.
No common (-fno-common)	Check this option if if you want to issue all the warnings demanded by strict ISO C and ISO C++; reject all programs that use forbidden extensions, and some other programs that do not follow ISO C and ISO C++. For ISO C, follows the version of the ISO C standard specified by any `-std' option used.
Pedantic (-pedantic)	Check if you want warnings like `-pedantic`, except that errors are produced rather than warnings.
Pedantic warnings as errors (-pedantic-errors)	Check this option if if you want to inhibit the display of warning messages.
Inhibit all warnings (-w)	Check this option if if you want to enable all the warnings about constructions that some users consider questionable, and that are easy to avoid (or modify to prevent the warning), even in conjunction with macros.

Table continues on the next page...

Table 2-6. ARM Ltd Windows GCC C Compiler > Warnings (continued)

Option	Description
All warnings (-Wall)	Check this option if if you want to enable all the warnings about constructions that some users consider questionable, and that are easy to avoid (or modify to prevent the warning), even in conjunction with macros.
Extra warnings (-Wextra)	Check this option to enable any extra warnings.
Warnings as errors (-Werror)	Check this option if if you want to make all warnings into hard errors. Source code which triggers warnings will be rejected.

2.6 ARM Ltd Windows GCC C Compiler > Miscellaneous

Use the **Miscellaneous** panel to specify compiler options. The following figure shows the **Miscellaneous** panel.

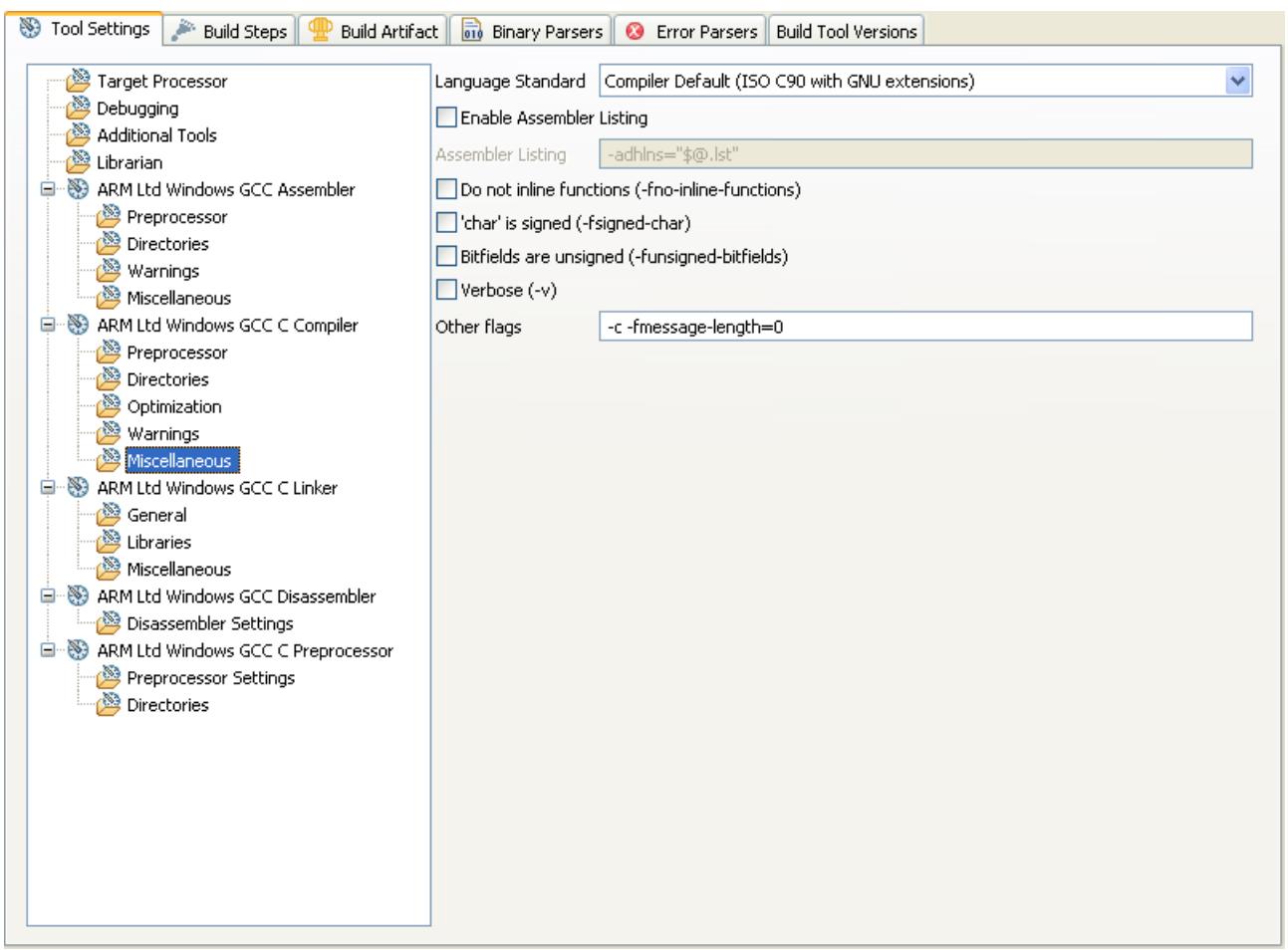

Figure 2-6. Tool Settings - ARM Ltd Windows GCC C Compiler > Miscellaneous

The following table lists and describes the various options available on the **Miscellaneous** panel.

Table 2-7. ARM Ltd Windows GCC C Compiler > Miscellaneous

Option	Description
Language Standard	Select the programming language or standard to which the compiler should conform. • ISO C90 (-ansi) - Select this option to compile code written in ANSI standard C. The compiler does not enforce strict standards. For example, your code can contain some minor extensions, such as C++ style comments (//), and $ characters in identifiers. • ISO C99 (-std=c99) - Select this option to instruct the compiler to enforce stricter adherence to the ANSI/ISO standard. • Compiler Default (ISO C90 with GNU extensions) - Select this option to enforce adherence to ISO C90 with GNU extensions. • ISO C99 with GNU Extensions (-std=gnu99)
Enable Assembler Listing	Check this option to enable the **Assembler Listing** option to create a listing.
Assembler Listing	Enables the assembler to create a listing file as it compiles assembly language into object code. Default: `-adhlns="$@.lst"`
Do not inline functions (-fno-inline-functions)	Check this option if you do not wnat to inline function.
char is signed (-fsigned-char)	Check this option if you want to ensure that the char is signed.
Bitfield are unsigned (-funsigned-bitfields)	Check this option to ensure bitfields are unsigned.
Verbose (-v)	Check this option if if you want the IDE to show each command-line that it passes to the shell, along with all progress, error, warning, and informational messages that the tools emit. This setting is equivalent to specifying the -v command-line option. By default this checkbox is clear. The IDE displays just error messages that the compiler emits. The IDE suppresses warning and informational messages.
Other flags	Specifies the compiler flags. The default value is: -c -fmessage-length=0

Chapter 3
EWL for Kinetis Development

Codewarrior GCC ARM Build Tools use EWL (Embedded Warrior Library) as a default library. EWL library set aims at reducing the memory footprint taken by IO operations and introduces a simpler memory allocator. To use newlib available with GCC instead of EWL, refer to the topic: Convert Kinetis Project to use newlib instead of EWL.

EWL Librarian I/O model is divided into 6 modes for C language:

- ewl- I/O operations through UART port.
- ewl_hosted- I/O operations through debugger console.
- ewl_noio- no I/O support, printf or scanf .
- c9x- same as ewl with additional wide char support.
- c9x_hosted- same as ewl_hosted with additional wide char support.
- c9x_noio- same as ewl_noio with additional wide char support.

Similarly, for C++, 6 sets of Librarian modes are supported.

- ewl_c++
- ewl_c++_hosted
- ewl_c++_noio
- c9x_c++
- c9x_c++_hosted
- c9x_c++_noio

The I/O modes can be selected depending on requirement while creating the project as illustrated below.

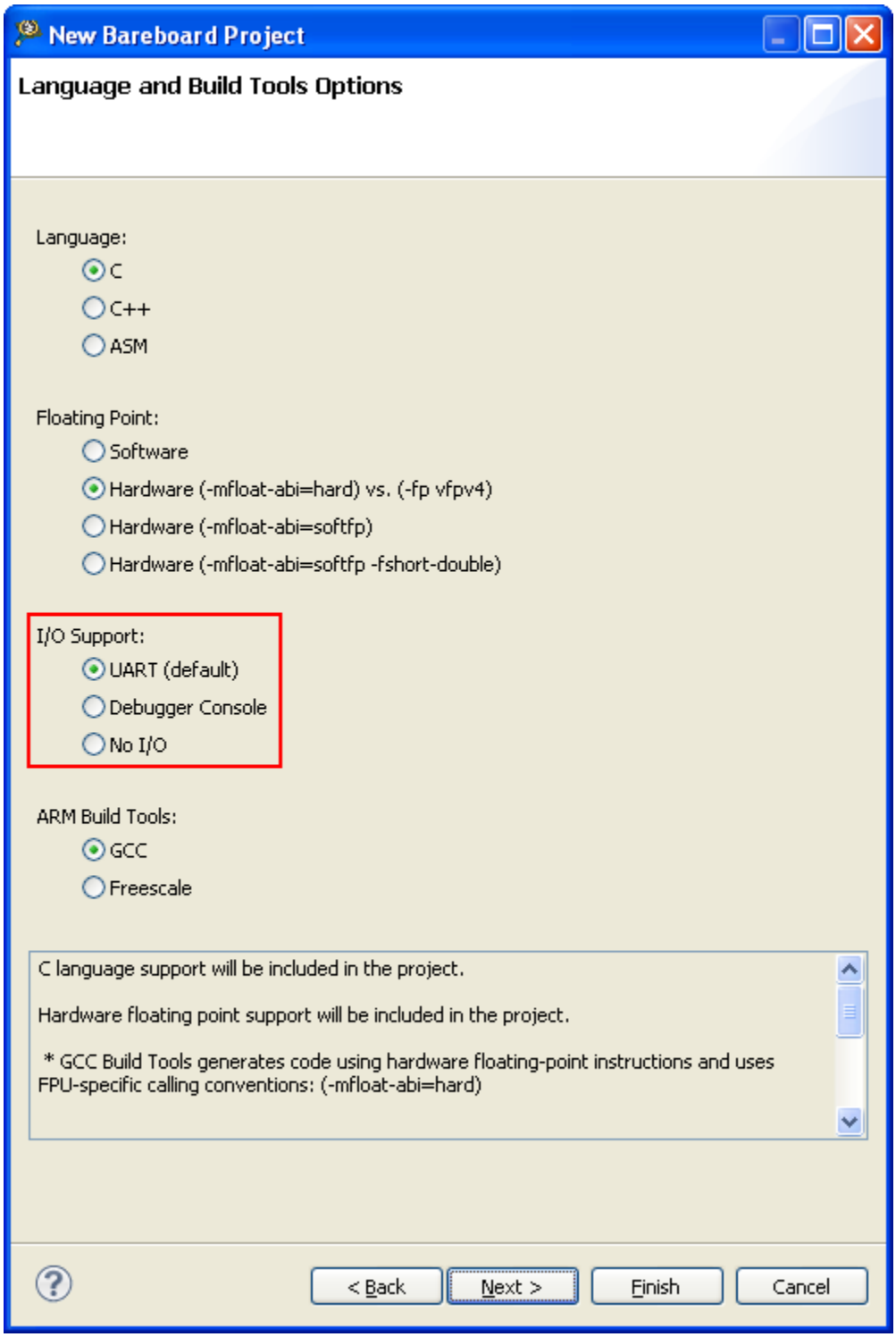

Figure 3-1. Language and Build Tools Options

Following library configurations are selected depending on the I/O mode:

- UART (default) - Librarian Model as ‘ewl’.
- Debugger Console - Librarian Model as ‘ewl_hosted’ .
- No I/O- Librarian Model as ‘ewl_noio’.

Similarly when C++ language is selected, c++ versions are picked.

NOTE

For UART I/O mode, additional configuration is required. For details, refer Set up UART Connections.

Following image shows compiler settings showing Librarian Model and its modes.

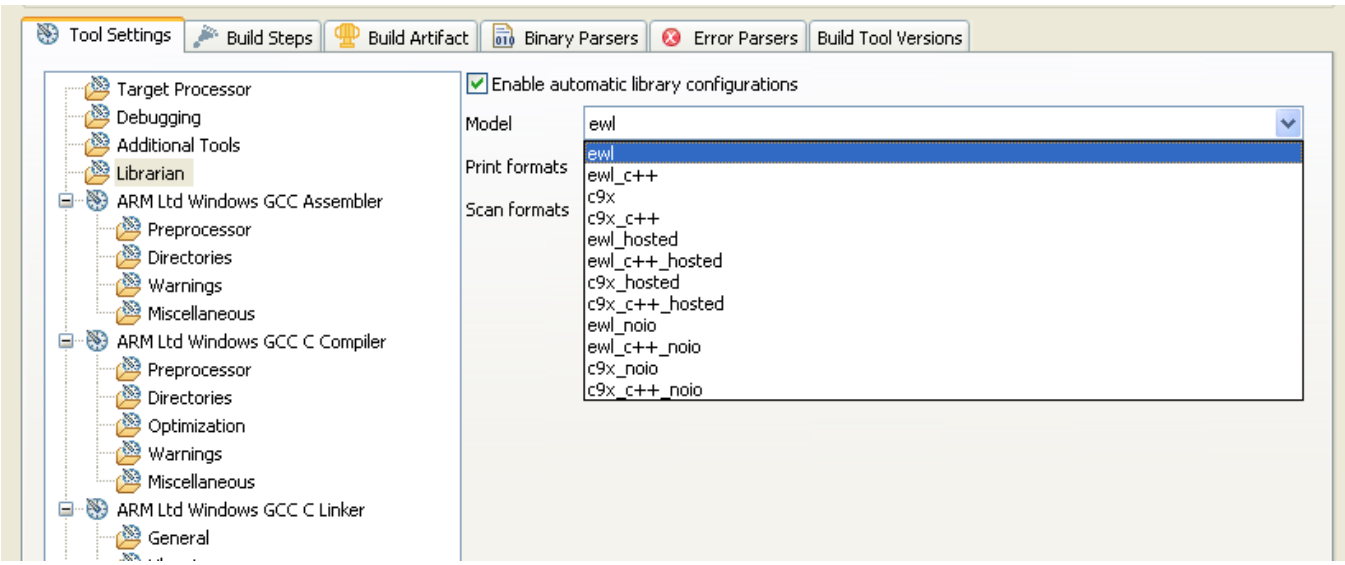

Figure 3-2. Compiler Settings - Librarian Model

The printing and scanning formatters for EWL are grouped in an effort to provide only the support required for the application:

- int - integer and string processing
- int_FP - integer, string and floating point
- int_LL - integer (including long long) and string
- int_FP_LL - all but wide chars

The above options can be selected at Print Formats and Scan Formats drop down box provided under Librarian panel. This is shown below.

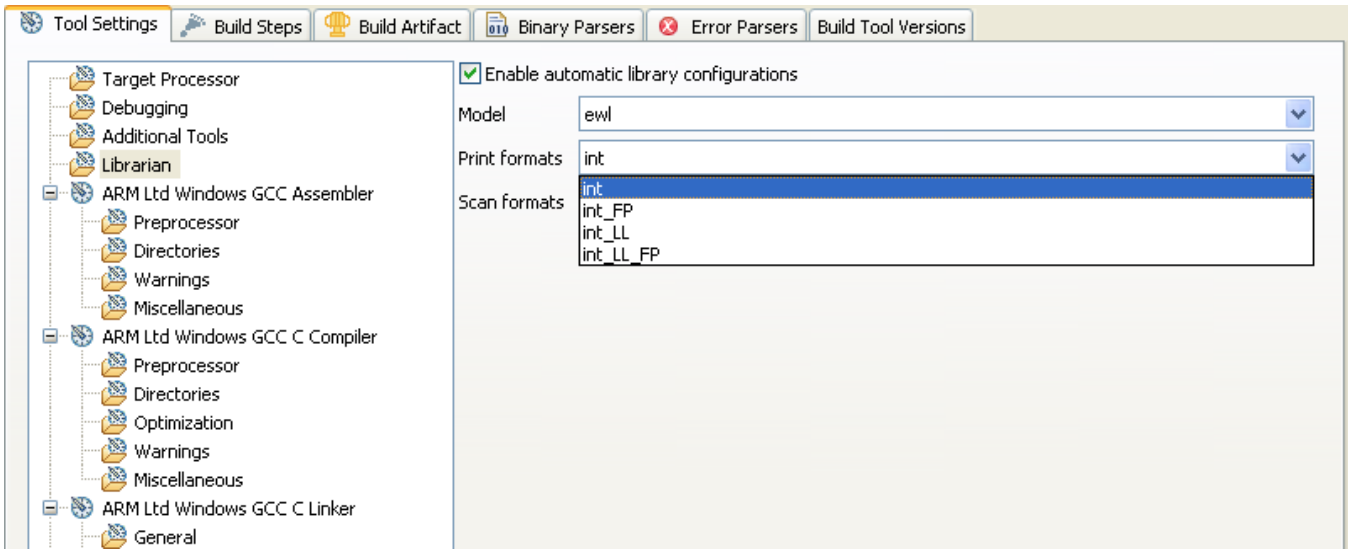

Figure 3-3. Compiler Settings - Print Formats

EWL libraries are prebuilt for several configurations. They are placed at **[CW Install Dir]/MCU/ARM_GCC_Support/ewl/lib**. Library is built for cortex-m0 and cortex-m4 ARM targets.

1. armv6-m for cortex-m0 and
2. armv7e-m for cortex-m4
    a. armv7e-m- Software floating-point
    b. armv7e-m/fpu- Hardware floating-point with hardware FP ABI
    c. armv7e-m/softfp- Hardware floating-point with soft FP ABI
    d. armv7e-m/spfp- Same as softfp but built with –fshort-double enabled, where double has same size as float.

Each target lib has following components:

- libc.a - c library
- libc99.a - c library with c99 support
- libc++.a- c++ library
- libstdc++.a- c++ with c99 support
- libhosted.a- library for console i/o(hosted) mode
- libuart.a- library for UART mode
- libm.a- math library
- librt.a- ewl runtime library containing ARM specific runtime support.

EWL sources are present at [CW Install Dir]/MCU/ARM_GCC_Support/ewl. User can rebuild library using existing makefiles. This can be done in two ways.

1. Rebuilding EWL Libraries from IDE
2. Rebuilding EWL Libraries from Command Prompt

3.1 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.

3.2 Rebuilding EWL Libraries from Command Prompt

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.

3.3 Convert Kinetis Project to use newlib instead of EWL

You can convert the ewl hello world project created for Kinetis devices to use newlib by making several changes to the build settings and the startup code. The purpose of these changes is to help the build process search the right directories for the right library files, and provide the right instructions to the linker.

Perform these steps to convert Kinetis project to use newlibe instead of EWL.

1. Start the IDE.
2. In the **CodeWarrior Projects** view, select the project for which you want to modify the build properties.
3. Select **Project > Properties** .

   The **Properties** window appears. The left side of this window has a properties list. This list shows the build properties that apply to the current project.

4. Expand the **C/C++ Build** property.

5. Select **Settings** .

   The **Properties** window shows the corresponding build properties.

6. Select **Librarian** and clear the **Enable automatic library configurations** checkbox.
7. Select **ARM Ltd Windows GCC C Compiler > Directories** .
8. Select and remove the EWL C/C++/Runtime include directories paths.

   For example remove:

   ```
   "${MCUToolsBaseDir}/ARM_GCC_Support/ewl/EWL_C/include" and

   "${MCUToolsBaseDir}/ARM_GCC_Support/ewl/EWL_Runtime/ include"
   ```

   This ensures that the compiler does not search in these EWL directories.

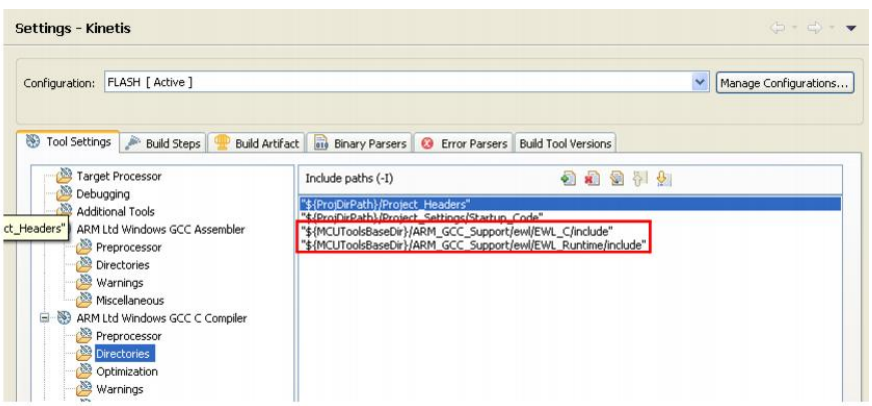

Figure 3-4. EWL C/C++/Runtime Include Directories Paths

9. Select **ARM Ltd Windows GCC C Linker > Libraries** .
10. Clear the **EWL library search path (-L)** .

    For example, remove:

    ```
    "${MCUToolsBaseDir}/ARM_GCC_Support/ewl/lib/armv6-m"
    ```

    The linker will not be searching for the ewl library.

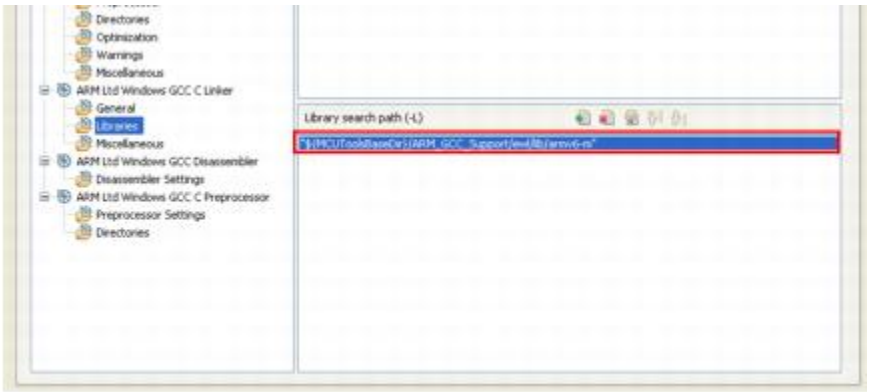

Figure 3-5. ARM Ltd Windows GCC C Linker Libraries

11. Select **ARM Ltd Windows GCC C Linker > Miscellaneous** .
12. Under **Linker flags (-Xlinker [option])** add:

    `-lc -lm -lgcc` -lrdimon

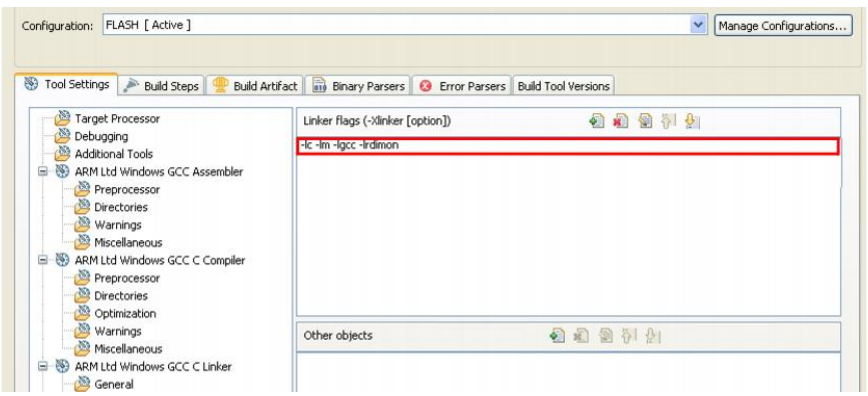

Figure 3-6. ARM Ltd Windows GCC C Linker Miscellaneous Settings

13. Scroll to the bottom of the settings panel and in the **Other flags** text box add `-specs=rdimon.specs`.

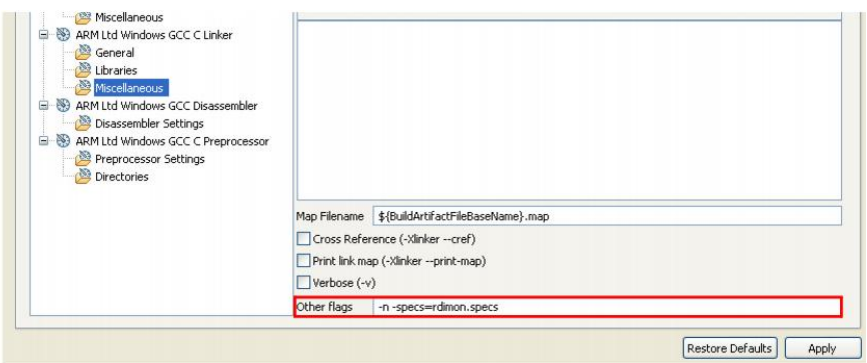

Figure 3-7. ARM Ltd Windows GCC C Linker Miscellaneous Settings - Other flags

14. Remove '`__arm_start.c`', '`__arm_end.c`' and '`runtime_configuration.h`' from Project_Settings/Startup_Code.

Figure 3-8. Remove __arm_start.c, __arm_end.c and runtime_configuration.h - Before

Figure 3-9. Remove __arm_start.c, __arm_end.c and runtime_configuration.h - After

15. Click **OK** to apply the modified build settings for the project.

    The above changes to the build settings ensure that the build picks newlib instead of ewl. However to make the build complete, you need to provide a startup routine.

Newlib provides a startup routine called *_start*. However, you must write a wrapper startup function that will do the following tasks before calling newlib '_start'.

1. Initialize the `sp` register.
2. Call the `__init_hardware` function which is defined in `kinetis_sysinit.c`.
3. Copy ROM section to RAM sections. This function call is not required if you are using RAM programming.
4. Call the `_start` function.

The following code demonstrates how what the wrapper startup file looks like. This file includes a `__rom_to_ram_copy` function along with wrapper startup function `__thumb_startup`.

To write a startup file and add it to the project, perform these steps.

1. Right click on the **Sources** folder of the project and select **New/Source File** .
2. Provide a name to the source file. For example `startup.S`.

   Alternatively, you can also create a `*.c` file and port the following code to inline c code.

3. Click **Finish** .
4. Open the recently created .S file and copy the code in the following listing.
5. The startup function `__thumb_startup` is included as an entry point in the existing linker command file. Therefore, the `__thumb_startup` function will be called first as execution starts.

Listing: Inline C Code

```
 .syntax unified
    .arch armv6-m

 .text
    .thumb
    .thumb_func

.align
 2

.globl
   __rom_to_ram_copy

 .type
   __rom_to_ram_copy, %function

__rom_to_ram_copy:

ldr r1,=___ROM_AT
ldr r2,=_sdata
ldr r3,=_edata

    subs    r3, r2
    ble    rom_to_ram_loop_end

movs r4,0

rom_to_ram_loop_begin:

   ldr    r0, [r1,r4]
   str    r0, [r2,r4]

    adds   r4, 4
    cmp    r4, r3
    blt    rom_to_ram_loop_begin

rom_to_ram_loop_end:

mov pc,lr

    .pool

 .size
 __rom_to_ram_copy, . - __rom_to_ram_copy
    .thumb_func

 .align
 2

 .globl
   __thumb_startup

 .type
    __thumb_startup, %function

__thumb_startup:
    ldr    r0, =_estack
    mov    sp,r0
```

```
    ldr     r0, =__init_hardware
    blx     r0
    ldr     r0, =__rom_to_ram_copy
    blx     r0
    ldr     r0, =_start
    bx      r0
    .pool

.size
__thumb_startup, . - __thumb_startup
    .end
```

3.4 Set up UART Connections

You can use puts/prints in the EWL hello world project created for Kinetis devices in the UART environment. You will need to make the following changes to build settings using EWL with the Kinetis gcc compiler and build tools. To perform the steps below, first create a project using the new project wizard (**File > New > Bareboard Project > Kinetis L Series > KL2x Family > KL25Z**) and select the derivative MKL25Z128. Next, select UART as I/O Support.

The steps below are for the ConsoleIO support for TWR-KL25Z128 + TRW-SER boards but the same steps can be applied for the other boards supported.

3.4.1 ConsoleIO support for TWR-KL25Z128 + TWR-SER boards

To make changes to the project build settings, perform these steps:

1. Add `ConsoleIO.c, ConsoleIO.h, UART0_PDD .h, PDD_Types.h` files to the project in the respective folders as shown in the image below. `ConsoleIO.c` contains the UART Console routines `__read_console, __write_console, __close_console, InitClock and ConsoleIO_Init`.

   Location of the files: `<CWInstallDir>\MCU\ARM_GCC_Support\UART\TWR-KL25Z128` (or your respective board).

   NOTE

   For TWR-K60N512 + TRW-SER board (Derivative - PK60N512) the location of the files is `<CWInstallDir>\MCU \ARM_GCC_Support\UART\TWR-K60N512. (ConsoleIO.c, ConsoleIO.h, UART_PDD .h, PDD_Types.h)`.

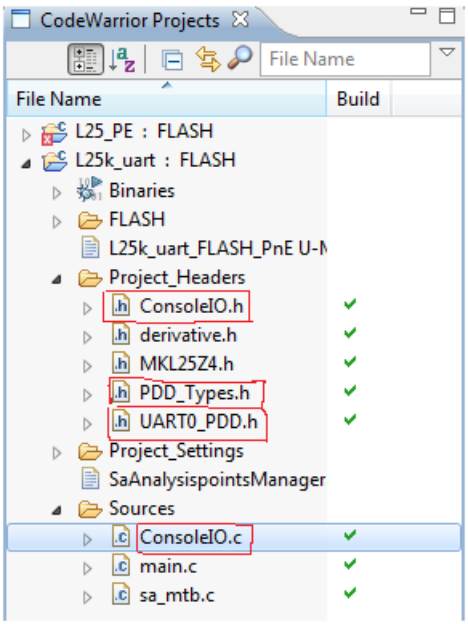

Figure 3-10. CodeWarrior Projects View for KL25Z128

2. Modify your code in main.c:
3. Include ConsoleIO.h
4. Call ConsoleIO_Init() to initialize the UART.

Listing: Simple Program

```
#include "derivative.h" /* include peripheral declarations */
#include "ConsoleIO.h"
int main(void)
{
int counter = 0;
ConsoleIO_Init();
puts("Hello World in 'C'!");
for(;;) {
counter++;
}
return 0;
}
```

5. Clean and rebuild the project.
6. Download your project to the board
7. Connecting a terminal to your board and 38400 baud, you should see the message printed to the terminal.

Chapter 4
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.

4.1 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

4.1.1 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.

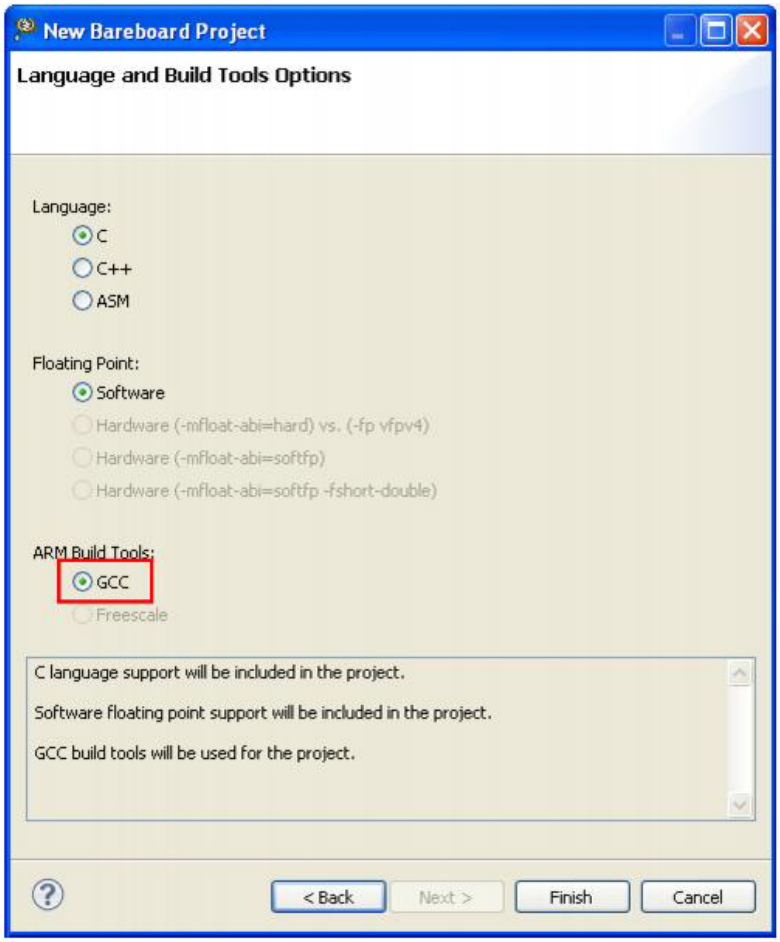

Figure 4-1. Language and Build Tools Options

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.

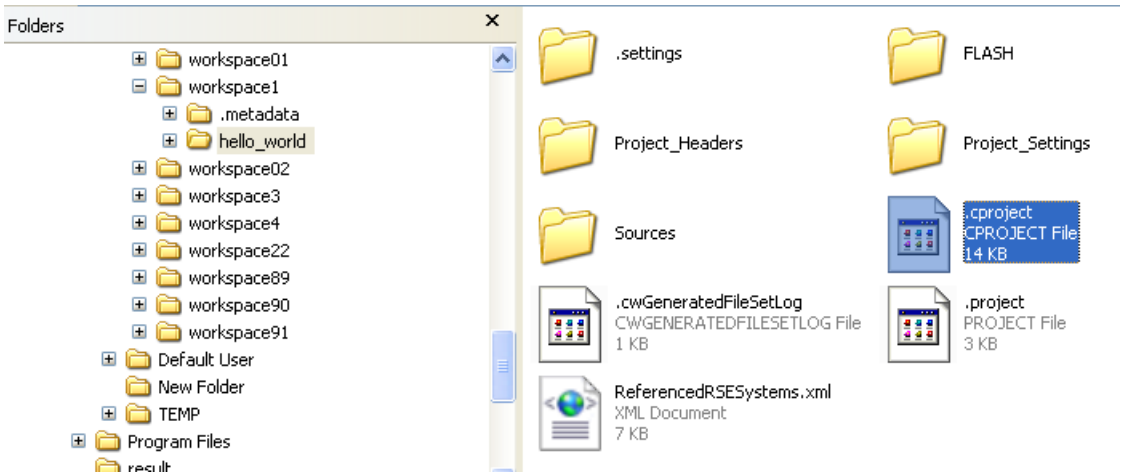

Figure 4-2. Replace the *.cproject file

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*.

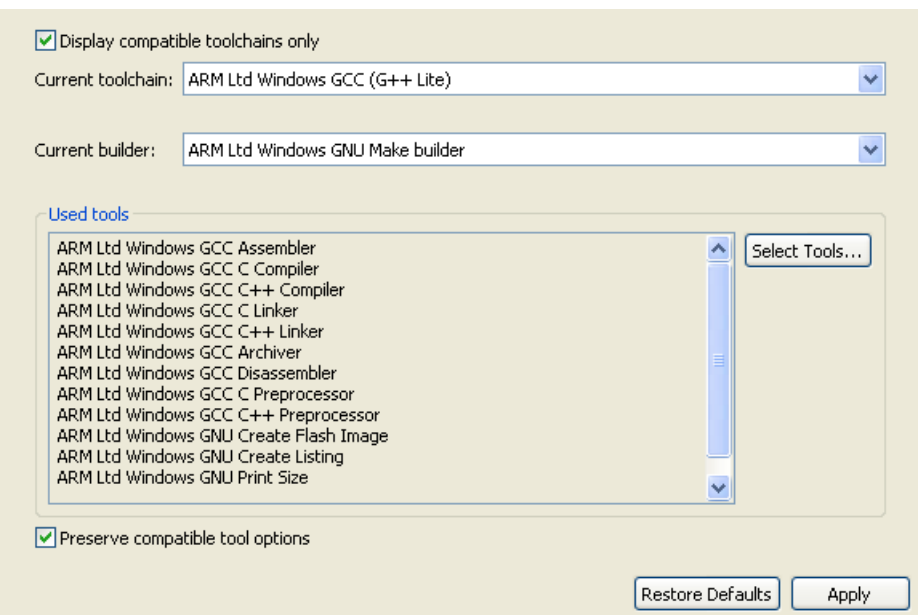

Figure 4-3. Tool Chain Editor

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.

4.1.2 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.

4.1.3 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.

4.1.4 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`.

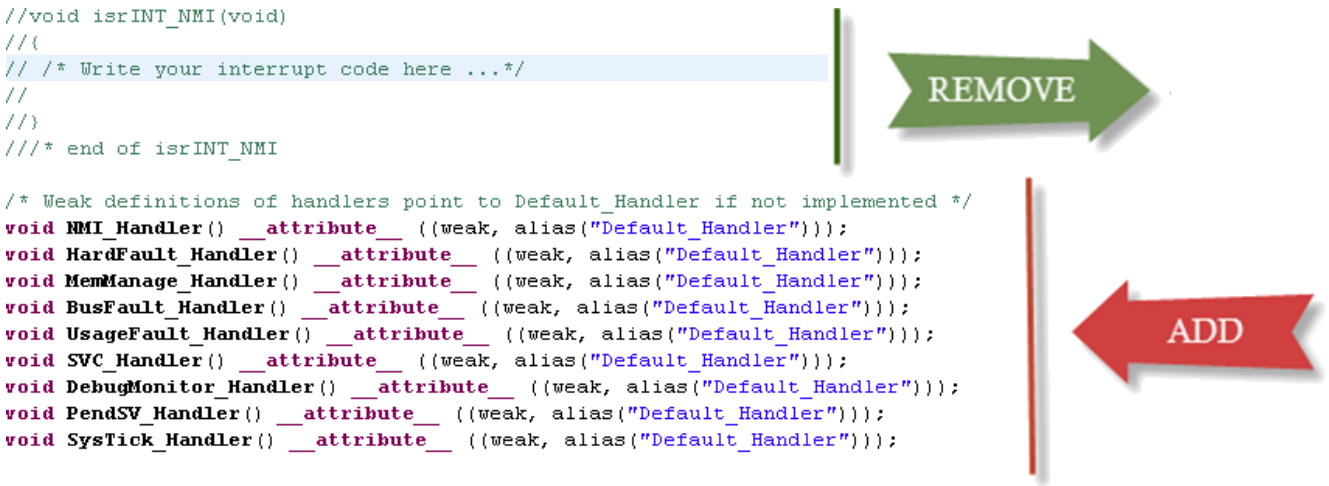

Figure 4-4. Add Weak Definition

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 4-5. 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: -) */
  //}
};
```

ADD

Figure 4-6. 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: -) */
  //}
};
```

ADD

Figure 4-7. 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();
```

4.1.5 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.

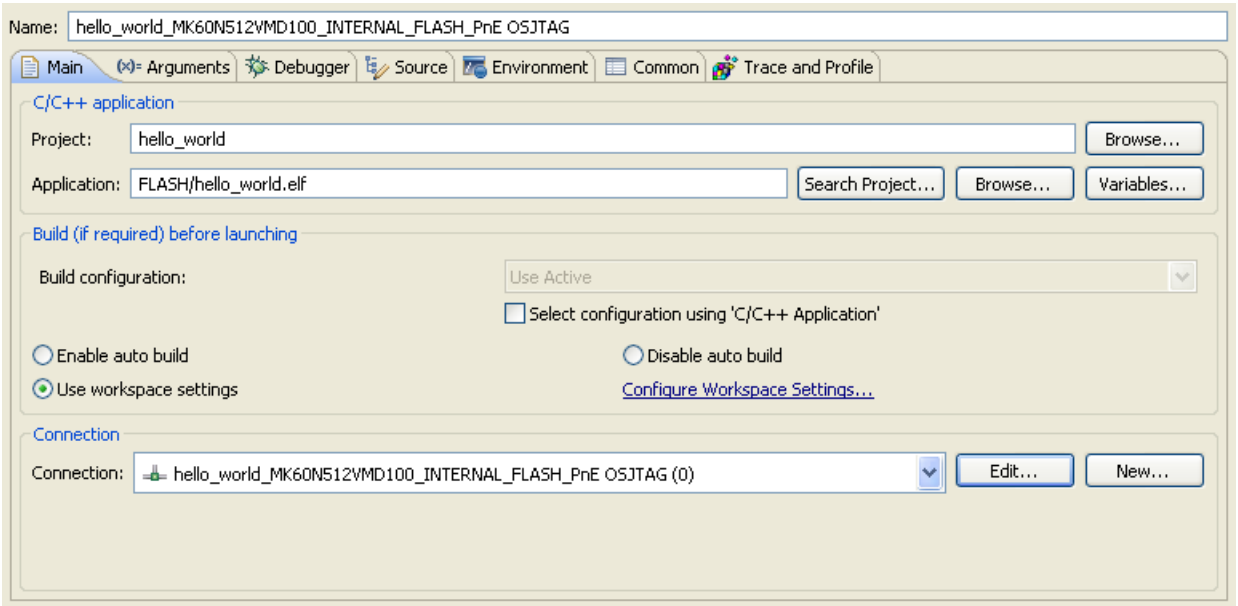

Figure 4-8. Debug Configuration - Main

5. Click **Debug** button.

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

4.2 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 5
Porting 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.

5.1 Built-in Macros

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

Table 5-1. Built-in Macros for CodeWarrior and GCC

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

5.2 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

5.2.1 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 5-2. 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)`	`#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`
`__option(__thumb)`	`#define __thumb __thumb__`

Table continues on the next page...

Table 5-2. Treatment of x in #define __option(x) x (continued)

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

5.2.2 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`.

5.2.3 Mapping __has_feature directive of CodeWarrior Driver to GCC

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

5.2.4 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.
```

5.3 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 5-3. 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`

5.4 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()))`

5.5 Macro

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

Table 5-4. 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`

5.6 Command-line Options

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

Table 5-5. 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 5-6. 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}

5.7 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`.

5.8 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)**

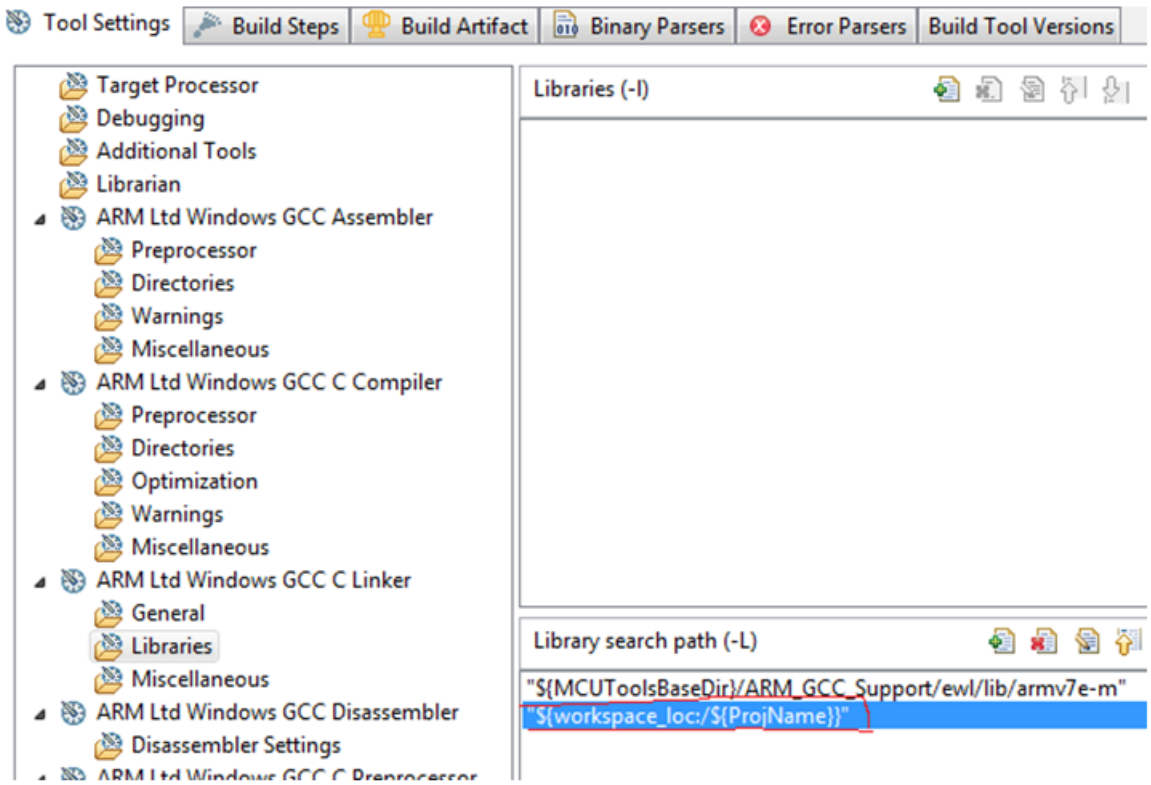

Figure 5-1. ARM Ltd. GCC C Linker->Libraries->Library search path (-L)

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.

5.9 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.

5.10 Target Specific Notes

The following are GCC predefined macros for ARM.

Table 5-7. 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)

5.11 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.

5.12 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

How to Reach Us:

Home Page:
freescale.com

Web Support:
freescale.com/support

Information in this document is provided solely to enable system and software implementers to use Freescale products. There are no express or implied copyright licenses granted hereunder to design or fabricate any integrated circuits based on the information in this document.

Freescale reserves the right to make changes without further notice to any products herein. Freescale makes no warranty, representation, or guarantee regarding the suitability of its products for any particular purpose, nor does Freescale 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 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 does not convey any license under its patent rights nor the rights of others. Freescale sells products pursuant to standard terms and conditions of sale, which can be found at the following address: freescale.com/SalesTermsandConditions.

Document Number CWMCUGCCCMPREF
Revision 10.6, 02/2014