Freescale Semiconductor, Inc.
User's Guide

Document Number: MQXKSDKPUG
Rev. 0, 03/2015

Porting an MQX RTOS Application to MQX RTOS for Kinetis SDK

1 Introduction

Freescale MQX™ Real Time Operating System (RTOS) for Kinetis SDK is the latest evolution of Freescale MQX Software Solutions for Kinetis MCUs. It is built on top of the Kinetis Software Development Kit (KSDK) for Kinetis MCUs, leveraging the software framework provided by the KSDK.

Freescale MQX RTOS for Kinetis SDK provides extensions to the SDK including the MQX RTOS, Real-Time TCP/IP Communication Suite (RTCS), and MQX File System (MFS).

Starting with Kinetis SDK v1.1, Freescale MQX RTOS is released with the Kinetis SDK.

For more information and downloads for the Kinetis SDK, visit www.freescale.com/ksdk.

This document describes how to port applications built for MQX RTOS v4.1.1 to MQX RTOS for Kinetis SDK v1.1.

Table 1

Contents

This information is intended to help embedded system developers port applications based on the legacy MQX RTOS architecture (known as classic MQX RTOS) to run on MQX RTOS for Kinetis SDK. However, the specific details do assume the existing application is based on classic MQX RTOS release, v4.1.1. Applications based on earlier MQX RTOS releases may require additional changes.

A specific demo application, known as the web_hvac demo, was ported and is used as the example for this document. This demo application is included in classic MQX RTOS. The application was ported and tested using the FRDM-K64F Freescale evaluation board. The tool-chain screenshots and examples come from Kinetis Design Studio (KDS) v2.0 Integrated Development Environment (IDE) unless stated otherwise.

2 Why port?

For new Kinetis MCU-based products in development, Freescale recommends upgrading to the Kinetis SDK as the software platform. This has several advantages:

- **Simpler structure** – This new platform is more flexible and extendable, with a simplified structure that is efficient for Kinetis MCU software development.
- **Easier migration of applications between Kinetis MCUs** – Porting and maintaining software for Kinetis MCUs is easier with a standard software platform designed to support all Kinetis MCU families, including the Kinetis K-series, L-series, and more.
- **Easier migration of applications to custom hardware** – The Kinetis SDK offers an efficient layered architecture that allows easier customization to meet the application requirements.
- **More drivers, improved drivers** – The new Kinetis SDK platform is more comprehensive with optimized drivers for key target applications

Beginning in 2015, support for new Kinetis MCUs will no longer be added to classic MQX RTOS (v4.x) releases. These new Kinetis MCUs will be supported instead by MQX RTOS for Kinetis SDK releases. Also, be aware that not all legacy Kinetis MCUs may be supported by the Kinetis SDK, so Freescale encourages to check the Kinetis SDK availability.

Porting MQX RTOS-based applications to MQX RTOS for Kinetis SDK keeps up with the continuing advancements in Kinetis MCUs and software. However, if the existing application is not ported, support and maintenance will continue to be available for classic Freescale MQX RTOS through Freescale's support packages. More details at www.freescale.com/mqx/support.

3 What is different in the new Kinetis SDK-based architecture?

This new architecture has a few key differences from the classic MQX RTOS architecture. This includes:

- A new directory structure
  - MQX RTOS, stacks, and middleware are integrated into the Kinetis SDK directory structure.

    - The default Kinetis SDK v1.1 installation path for Windows® Operating System, referenced as ${KSDK_PATH} in this document, is at *C:/Freescale/KSDK_1.1.0/.*
- New library organization
    - As of v1.1, there is no longer a board support package library (BSP). Much of the equivalent functionality of the BSP library is now provided by the Kinetis SDK platform library.
    - A fully re-entrant standard C library is provided for MQX RTOS applications (called mqx_stdlib).
    - A new POSIX-compliant Shell command line interface library (called nShell) is provided. The API is slightly changed.
- Changes to the I/O subsystem
    - A new POSIX-compliant IO subsystem is provided (called nIO) for I/O file descriptor handling. The API is slightly changed.
- Change to the peripheral driver model
    - In all but a very few number of cases, MQX RTOS no longer includes its own peripheral drivers. Instead, the Kinetis SDK peripheral drivers are used to access peripherals.
    - The use of most drivers requires that they be installed by the application. For MQX RTOS for Kinetis SDK v1.1, only the UART and Ethernet drivers (when supported) are installed prior to the application starting up.
- Change in startup code and interrupt handling
    - MQX RTOS for KSDK is using KSDK linker command files and startup files instead of MQX RTOS files. Therefore MQX RTOS for KSDK does not contain the vectors.c file, and the macro MQXCFG_VECTOR_ROM is no longer used to decide between RAM and ROM vector table. The linker symbol __ram_vector_table__ is used instead. Startup code is now located in KSDK platform folder.
- USB Stack now part of KSDK
    - The USB stacks available with Kinetis SDK are different than those found with classic MQX RTOS, however the APIs used by the application are very similar if not identical.

4 What is the same in the new Kinetis SDK-based architecture?

- MQX RTOS API is unchanged
  - The same full-featured and lightweight services are available and have equivalent functionality, such as sem, lwsem, event, lwevent, etc. Also, the way tasks are created and scheduled is the same.
- MQX RTOS RTCS API is unchanged
  - Although the library location in the source tree is different, the TCP/IP stack API is the same.
- MQX RTOS MFS API is unchanged
  - Although the library location in the source tree is different, the file system API is the same.

5 Overview of porting an application

5.1 MQX RTOS Core

The MQX RTOS API is the same, however the naming convention has changed with the kernel.

Classic MQX RTOS	***MQX RTOS for Kinetis SDK***
PSP library is used for the kernel components	The kernel library is simply called ***mqx_<board>***, no longer **PSP**
BSP library is used for drivers and hardware configuration	**BSP** library has been removed and replaced by the MQX RTOS for Kinetis SDK library - ***ksdk_mqx_lib***

5.2 Project Settings

Aside from the peripheral driver changes, the majority of the porting effort is changing the project settings for the application in the toolchain. With MQX RTOS for Kinetis SDK, the directory locations of the MQX RTOS libraries and header files have changed and some libraries and paths have been renamed. The following sections provide the details for the changes required in the application to port to MQX RTOS for Kinetis SDK.

5.3 Drivers

As MQX RTOS now uses the Kinetis SDK peripheral drivers and hardware abstraction layer, the application code needs to be updated for the Kinetis SDK driver APIs. The Kinetis SDK drivers offer similar functionality, and the porting effort involves changing the initialization of the drivers in the application, and changing the APIs used in the tasks.

Refer to the Kinetis SDK v.1.1 API Reference Manual.pdf and Kinetis SDK v.1.1 Demo Applications User's Guide.pdf as a reference for the new drivers. These documents are located in KSDK installation path: C:/Freescale/KSDK_1.1.0/doc

5.4 Additional Libraries and Stacks

The MFS file system, Ethernet RTCS, and Shell libraries are very similar. If any application changes are required for these libraries, they are superficial.

The USB stacks used with MQX RTOS for Kinetis SDK are different than classic MQX RTOS. The Kinetis SDK uses new unified USB stacks that run baremetal (no RTOS), with MQX RTOS, or with other RTOSes. While the stacks are different, the APIs used by the application are very similar if not identical. The Kinetis SDK USB stacks were modeled after the classic MQX RTOS stacks, and porting is not difficult. Refer to the Kinetis SDK USB stack documentation and examples for the specific API details. All USB stack documentation can be found in KSDK installation path also: C:/Freescale/KSDK_1.1.0/usb/doc

6 Porting the example

A specific demo application known as the web_hvac demo is used as the example for this document. This demo application is included in classic MQX RTOS installation path: C:/Freescale/Freescale_MQX_4_1/demo/web_hvac

It represents the residential HVAC controller system requirements and demonstrates the following features of MQX RTOS, stacks, middleware, and drivers.

- Multiple tasks
- Light Weight Events
- Message Passing
- Kernel Logging
- USB Stack
- File System
- Shell
- RTCS (TCP/IP stack)
- Webserver with CGI, interactive web pages with AJAX
- FTP Server
- Telnet Server
- Light Weight GPIO driver

The application was ported and tested using the **FRDM-K64F** Freescale evaluation board, the tool-chain screenshots and examples come from **Kinetis Design Studio (KDS) 2.0** Integrated Development Environment (IDE) unless stated otherwise.

The next figure shows in detail all the parts that interact with this demo application.

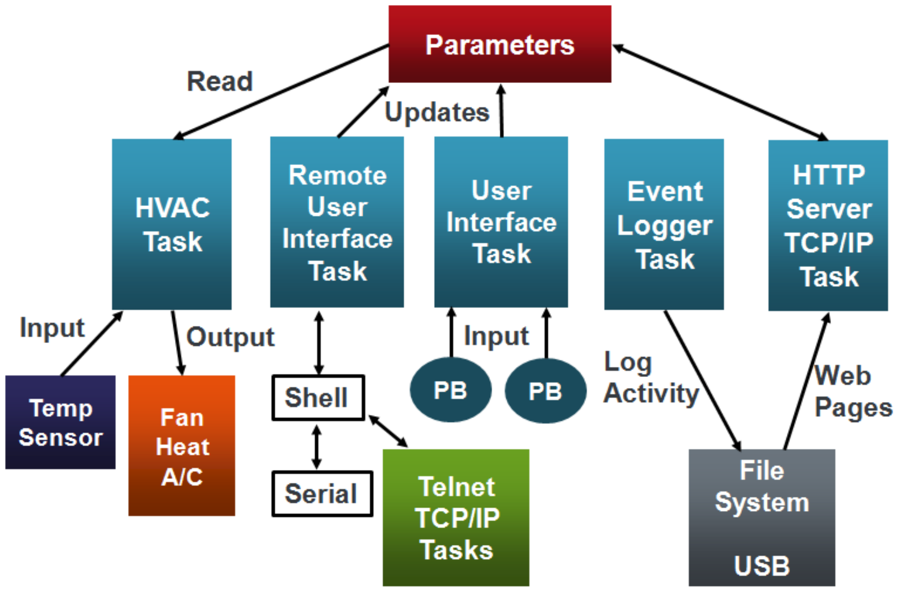


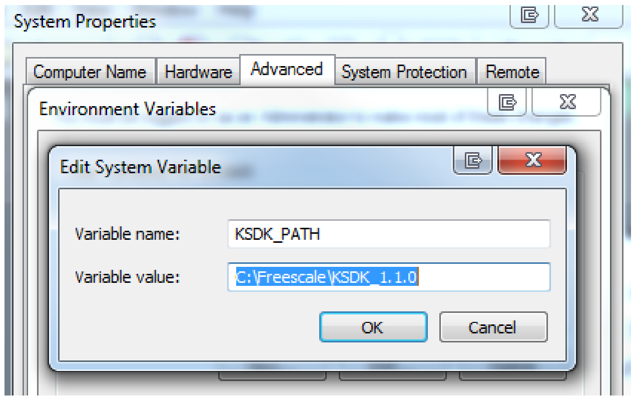
Figure 1 Web_HVAC Application Block Diagram

For features not covered by this example, additional porting may be required that is not documented here.

6.1 Installation requirements

When using KDS with KSDK projects, the user must install the Eclipse update. The update instructions are located in chapter **5.2 Install Eclipse update** of **'Getting Started with Kinetis SDK (KSDK)'** document, which is located in KSDK installation path: ***C:/Freescale/KSDK_1.1.0/doc/Getting Started with Kinetis SDK (KSDK).pdf***

It is also important to verify that **KSDK_PATH** environment variable points to the KSDK directory ***C:/Freescale/KSDK_1.1.0***. To verify, go to *Windows Control Panel > System and Security > System > Advanced System Settings* and click Environment variables.



Variable name: KSDK_PATH

Variable value: C:\Freescale\KSDK_1.1.0

Figure 2 Windows Operating System environment variable for KSDK_PATH

6.2 Create a Base Project

A copy of the classic MQX RTOS web_hvac project is used as a porting base.

1. Go to menu File > Import > Existing Project into Workspace and search for the classic MQX RTOS web_hvac project located in this path: C:/Freescale/Freescale_MQX_4_1/demo/web_hvac/build/kds/web_hvac_frdmk64f.
2. Right click on the project name and select 'Copy' in the context menu. Then, right click on the KDS workbench and select 'Paste'.

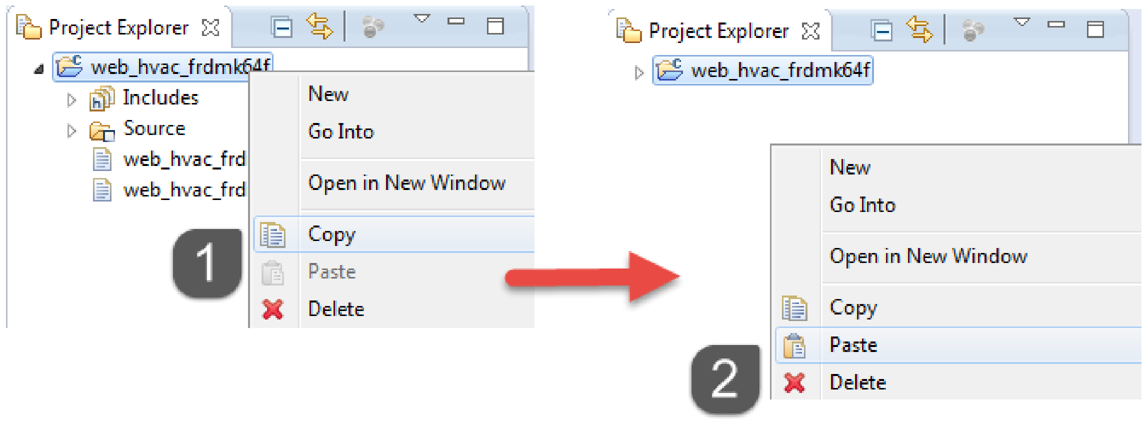
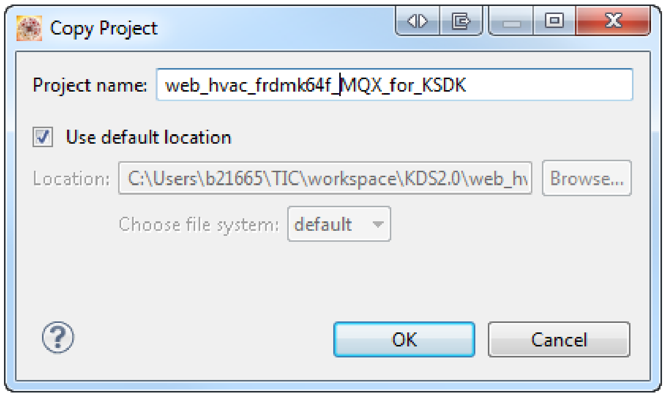


Figure 3 KDS copy imported project

3. Write a name for the project and click 'OK'. The new project should be created.


Copy Project

Project name: web_hvac_frdmk64f_MQX_for_KSDK

[✓] Use default location

Location: C:\Users\b21665\TIC\workspace\KDS2.0\web_hv Browse...

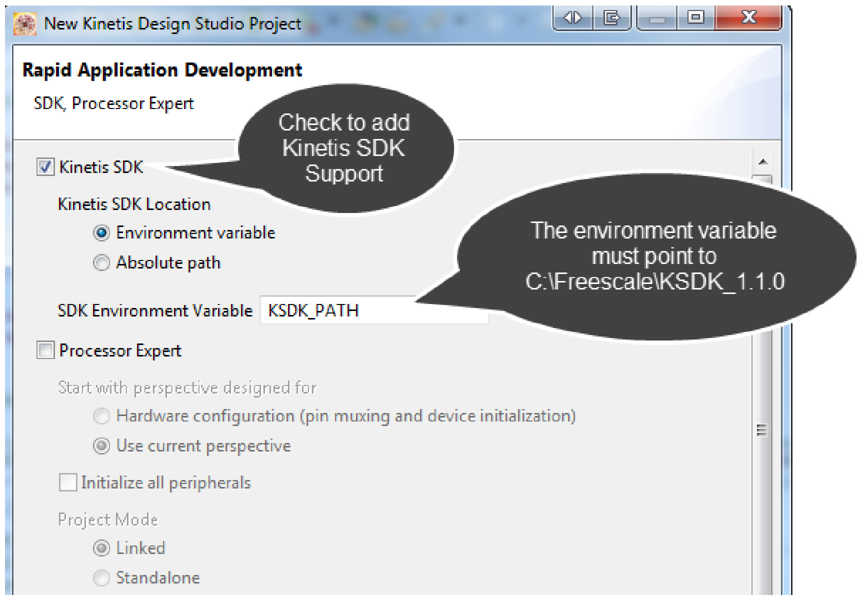
Choose file system: default

OK Cancel

Figure 4 Rename the project

Note

If a New Kinetis Design Studio Project is required to be used as a base project, the user must add the Kinetis SDK support by checking the Kinetis SDK option in the Rapid Application Development Window of the New Kinetis Design Studio Project Wizard as shown in the figure above. KDS IDE version 2.0 and KSDK version 1.1.0 are used in this document.



New Kinetis Design Studio Project

Rapid Application Development

SDK, Processor Expert

[✓] Kinetis SDK

Kinetis SDK Location

● Environment variable

○ Absolute path

SDK Environment Variable: KSDK_PATH

[ ] Processor Expert

Start with perspective designed for

○ Hardware configuration (pin muxing and device initialization)

● Use current perspective

[ ] Initialize all peripherals

Project Mode

● Linked

○ Standalone

Check to add Kinetis SDK Support
The environment variable must point to C:\Freescale\KSDK_1.1.0

Figure 5 KDS New Project Wizard settings for new KSDK project

6.3 Compiler Settings Changes

Update the project compiler settings (include paths for the header files and some processor definitions) need to ensure that the application works with the Kinetis SDK.

Before making any changes, build all necessary libraries to create the corresponding library files (*.a). The paths containing the libraries source code and the output (*.a) files are listed here.

MQX RTOS for KSDK Platform Lib

${KSDK_PATH}/lib/ksdk_mqx_lib/kds/K64F12

${KSDK_PATH}/lib/ksdk_mqx_lib/kds/K64F12/Debug/libksdk_platform_mqx.a

MQX RTOS Lib

${KSDK_PATH}/rtos/mqx/mqx/build/kds/mqx_frdmk64f

${KSDK_PATH}/rtos/mqx/lib/frdmk64f.kds/debug/mqx/lib_mqx.a

MQX RTOS stdlib

${KSDK_PATH}/rtos/mqx/mqx_stdlib/build/kds/mqx_stdlib_frdmk64f

${KSDK_PATH}/rtos/mqx/lib/frdmk64f.kds/debug/mqx_stdlib/lib_mqx_stdlib.a

MQX RTOS nShell Lib

${KSDK_PATH}/rtos/mqx/nshell/build/kds/nshell_frdmk64f

${KSDK_PATH}/rtos/mqx/lib/frdmk64f.kds/debug/nshell/lib_nshell.a

MFS File System

${KSDK_PATH}/filesystem/mfs/build/kds/mfs_frdmk64f

${KSDK_PATH}/filesystem/mfs/lib/frdmk64f.kds/debug/mfs/lib_mfs.a

RTCS

${KSDK_PATH}/tcpip/rtcs/build/kds/rtcs_frdmk64f

${KSDK_PATH}/tcpip/rtcs/lib/frdmk64f.kds/debug/rtcs/lib_rtcs.a

USB Host Lib

${KSDK_PATH}/usb/usb_core/host/build/kds/usbh_sdk_frdmk64f_mqx

${KSDK_PATH}/usb/usb_core/host/build/kds/usbh_sdk_frdmk64f_mqx/Debug/libusbh_mqx.a

The subsequent chapters provide information about these libraries and their equivalents in MQX RTOS Classic.

6.3.1 Assembler Include Paths

All assembler include paths can be deleted to avoid warnings. Go to menu Project > Properties C/C++ Build > Settings > Cross ARM® GNU Assembler > Includes > Include paths (-I) and delete all the classic MQX RTOS paths.

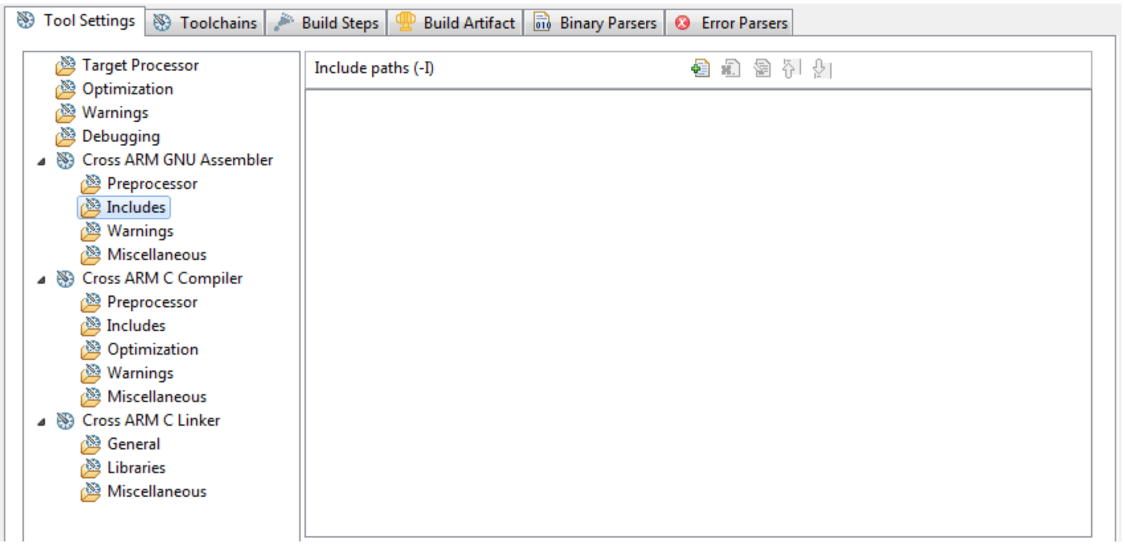


Figure 6 Remove Assembler include paths in KDS

6.3.2 Compiler Include Paths

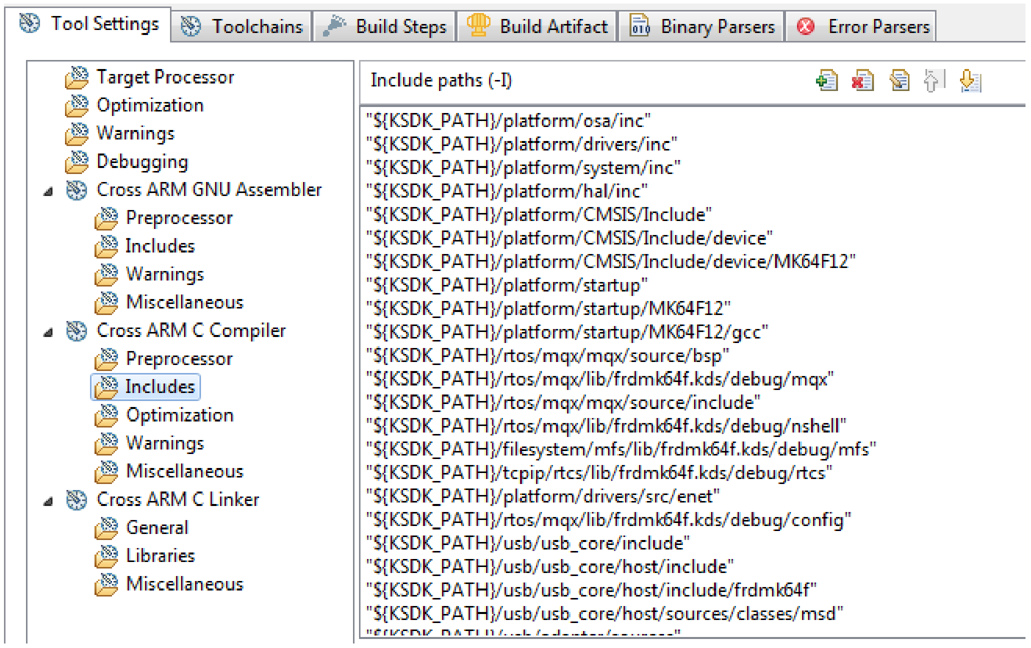
The existing MQX RTOS header file directories have moved and the library paths have changed to accommodate the new KDSK folder structure.

Go to **menu Project > Properties C/C++ Build > Settings > Cross ARM C Compiler > Includes > Include paths (-I)** and change the compiler include paths of the classic MQX RTOS directories and add the Include paths of the libraries with the Kinetis SDK directory structure.

Library	**Classic MQX RTOS**	***MQX RTOS for Kinetis SDK***
BSP	*/lib/<board>.<tool>/debug/bsp* */lib/<board>.<tool>/debug/bsp/Generated_Code* */lib/<board>.<tool>/debug/bsp/Sources*	"${KSDK_PATH}/rtos/mqx/mqx/source/bsp"
PSP	*/lib/<board>.<tool>/debug/psp*	"${KSDK_PATH}/rtos/mqx/lib/<board>.<tool>/debug/mqx" "${KSDK_PATH}/rtos/mqx/mqx/source/include"
Shell	*/lib/<board>.<tool>/debug/shell*	"${KSDK_PATH}/rtos/mqx/lib/<board>.<tool>/debug/nshell"
MFS	*/lib/<board>.<tool>/debug/mfs*	"${KSDK_PATH}/filesystem/mfs/lib/<board>.<tool>/debug/mfs"
RTCS	*/lib/<board>.<tool>/debug/rtcs*	"${KSDK_PATH}/tcpip/rtcs/lib/<board>.<tool>/debug/rtcs" "${KSDK_PATH}/platform/drivers/src/enet"

USB Host	*/lib/<board>.<tool>/debug/usb/*	"${KSDK_PATH}/usb/usb_core/include" "${KSDK_PATH}/usb/usb_core/host/include" "${KSDK_PATH}/usb/usb_core/host/include/<board>" "${KSDK_PATH}/usb/usb_core/host/sources/classes/<class>" "${KSDK_PATH}/usb/adapter/sources" "${KSDK_PATH}/usb/adapter/sources/sdk" "${KSDK_PATH}/usb/usb_core/host/sources/classes/hub"
Config	*/config/<board>*	"${KSDK_PATH}/rtos/mqx/lib/<board>.<tool>/debug/config"
Other	*/lib/<board>.<tool>/debug*	*"${KSDK_PATH}/platform/osa/inc"* *"${KSDK_PATH}/platform/drivers/inc"* *"${KSDK_PATH}/platform/system/inc"* *"${KSDK_PATH}/platform/hal/inc"* *"${KSDK_PATH}/platform/CMSIS/Include"* [1] *"${KSDK_PATH}/platform/CMSIS/Include/device"* [1] *"${KSDK_PATH}/platform/CMSIS/Include/device/MK64F12"* [1] *"${KSDK_PATH}/platform/startup"* [1] *"${KSDK_PATH}/platform/startup/MK64F12"* [1] *"${KSDK_PATH}/platform/startup/MK64F12/gcc"* [1]

This is an example of the final compiler include paths after making the changes:



Include paths (-I)

```
"${KSDK_PATH}/platform/osa/inc"
"${KSDK_PATH}/platform/drivers/inc"
"${KSDK_PATH}/platform/system/inc"
"${KSDK_PATH}/platform/hal/inc"
"${KSDK_PATH}/platform/CMSIS/Include"
"${KSDK_PATH}/platform/CMSIS/Include/device"
"${KSDK_PATH}/platform/CMSIS/Include/device/MK64F12"
"${KSDK_PATH}/platform/startup"
"${KSDK_PATH}/platform/startup/MK64F12"
"${KSDK_PATH}/platform/startup/MK64F12/gcc"
"${KSDK_PATH}/rtos/mqx/mqx/source/bsp"
"${KSDK_PATH}/rtos/mqx/lib/frdmk64f.kds/debug/mqx"
"${KSDK_PATH}/rtos/mqx/mqx/source/include"
"${KSDK_PATH}/rtos/mqx/lib/frdmk64f.kds/debug/nshell"
"${KSDK_PATH}/filesystem/mfs/lib/frdmk64f.kds/debug/mfs"
"${KSDK_PATH}/tcpip/rtcs/lib/frdmk64f.kds/debug/rtcs"
"${KSDK_PATH}/platform/drivers/src/enet"
"${KSDK_PATH}/rtos/mqx/lib/frdmk64f.kds/debug/config"
"${KSDK_PATH}/usb/usb_core/include"
"${KSDK_PATH}/usb/usb_core/host/include"
"${KSDK_PATH}/usb/usb_core/host/include/frdmk64f"
"${KSDK_PATH}/usb/usb_core/host/sources/classes/msd"
```

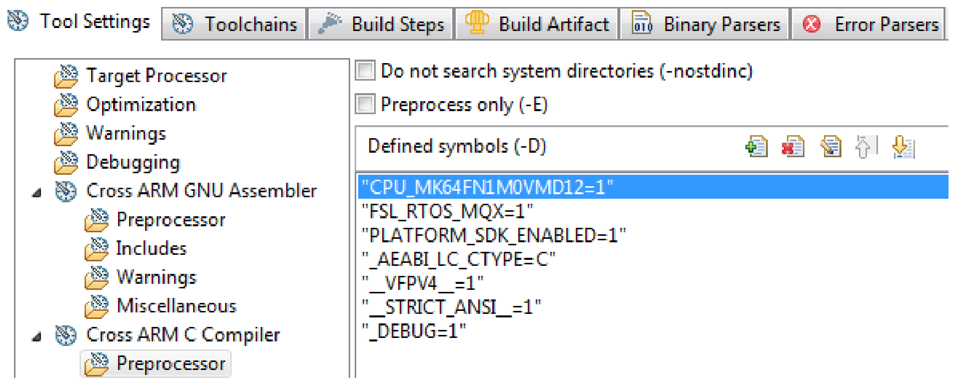
Figure 7 Compiler include paths for FRDM-K64F example

6.3.3 Compiler preprocessor settings

The Kinetis SDK source files use macros defined in the compiler preprocessor settings. These definitions need to be added to the ported MQX RTOS application project. Below is a list of definitions to add.

The user should review the definitions used in the Kinetis SDK example projects. To review the KSDK definitions, open a MQX RTOS for Kinetis SDK example project for the preferred toolchain and the development board and review the preprocessor definitions used in that project.

- "CPU_MK64FN1M0VMD12=1"
- "FSL_RTOS_MQX=1"
- "PLATFORM_SDK_ENABLED=1"
- "_AEABI_LC_CTYPE=C"
- "__VFPV4__=1"
- "__STRICT_ANSI__=1"
- "_DEBUG=1"



```
"CPU_MK64FN1M0VMD12=1"
"FSL_RTOS_MQX=1"
"PLATFORM_SDK_ENABLED=1"
"_AEABI_LC_CTYPE=C"
"__VFPV4__=1"
"__STRICT_ANSI__=1"
"_DEBUG=1"
```

Figure 8 Compiler preprocessor settings from FRDM-K64F example

6.4 Linker setting changes

The user should update the application project linker settings, such as the settings for the MQX RTOS and Kinetis SDK libraries, and the linker command file.

6.4.1 Linker library settings

MQX RTOS for Kinetis SDK has new libraries that need to be added to the ported application.

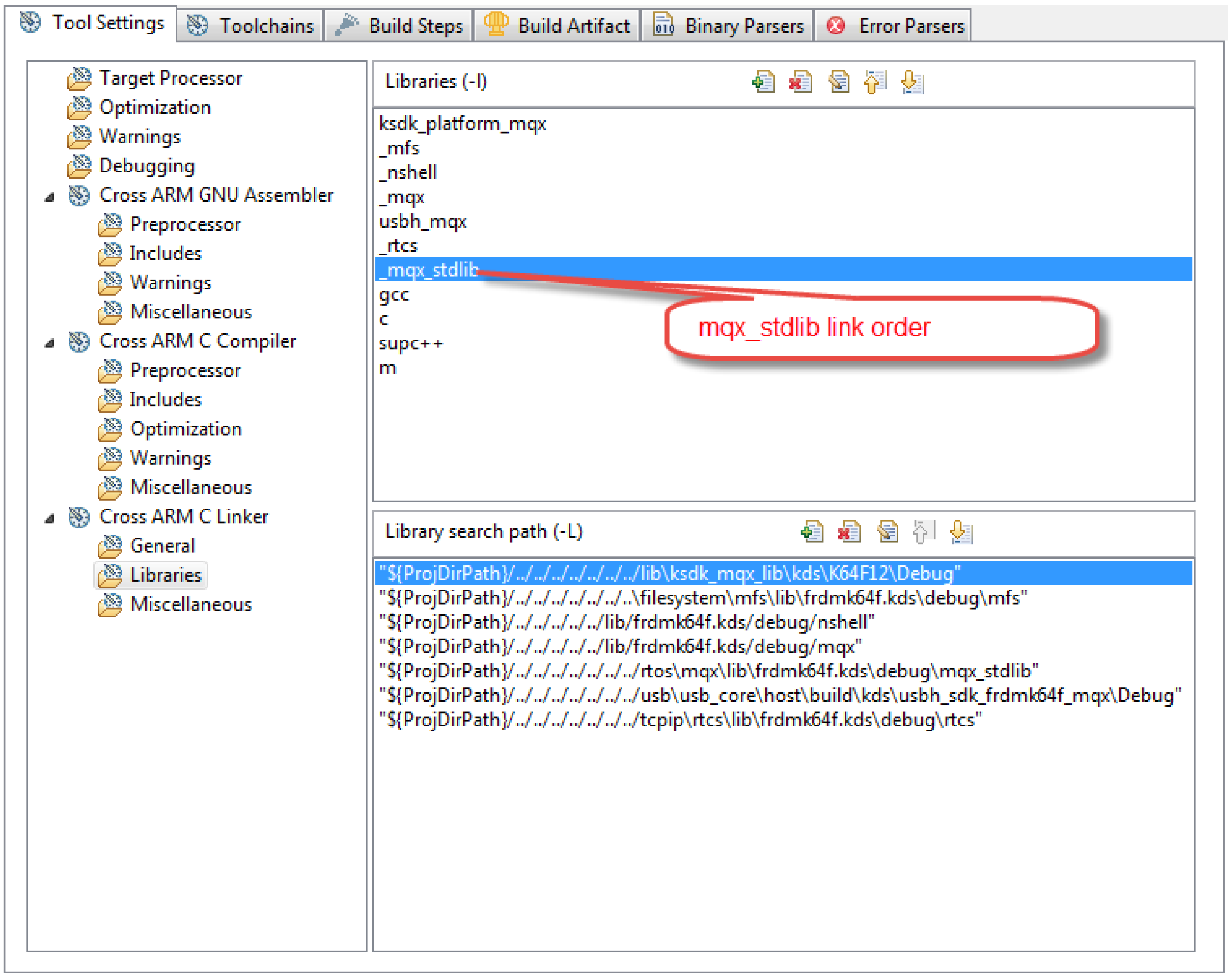
NOTE

It is critical to link the the new mqx_stdlib in the correct order. If this library is linked in the incorrect order, the linker will link some of the stdio functions used in the application to the wrong library. The user should link the mqx_library after all other MQX RTOS and Kinetis SDK libraries, but before the toolchain runtime libraries. To see the proper order for the desired toolchain, open an MQX RTOS for Kinetis SDK application example project and review the linking order for that project.

The existing MQX RTOS libraries have moved to new directories and have been renamed. In this table are the new locations and their equivalents in MQX RTOS classic. Go to menu **Project > Properties C/C++ Build > Settings > Cross ARM C++ Linker > Libraries** and add the library names (without lib prefix and without .a extension) in Libraries (-l). In Miscellaneous, add the library names with the entire path. See Figures 6 and 7.

Library	Classic MQX RTOS	*MQX RTOS for Kinetis SDK*
Platform (BSP)	*/lib/<board>.<tool>/debug/bsp/bsp.a*	*"${KSDK_PATH}/lib/ksdk_mqx_lib/kds/<derivative>/Debug/libksdk_platform_mqx.a"*
STDIO	*N/A*	*"${KSDK_PATH}/rtos/mqx/lib/<board>.<tool>/debug/mqx_stdlib/lib_mqx_stdlib.a"*
PSP	*/lib/<board>.<tool>/debug/psp/psp.a*	*"${KSDK_PATH}/rtos/mqx/lib/<board>.<tool>/debug/mqx/lib_mqx.a"*
Shell	*/lib/<board>.<tool>/debug/shell/shell.a*	*"${KSDK_PATH}/rtos/mqx/lib/<board>.<tool>/debug/nshell/lib_nshell.a"*
MFS	*/lib/<board>.<tool>/debug/mfs/mfs.a*	*"${KSDK_PATH}/filesystem/mfs/lib/<board>.<tool>/debug/mfs/lib_mfs.a"*
RTCS	*/lib/<board>.<tool>/debug/rtcs/rtcs.a*	*"${KSDK_PATH}/tcpip/rtcs/lib/<board>.<tool>/debug/rtcs/lib_rtcs.a"*
USB Host	*/lib/<board>.<tool>/debug/usb/usbh.a*	*"${KSDK_PATH}/usb/usb_core/host/build/kds/usbh_sdk_<board>_mqx/Debug/libusbh_mqx.a"*

This is an example of the linker library settings after the changes.


Tool Settings | Toolchains | Build Steps | Build Artifact | Binary Parsers | Error Parsers

- Target Processor
- Optimization
- Warnings
- Debugging
- Cross ARM GNU Assembler
  - Preprocessor
  - Includes
  - Warnings
  - Miscellaneous
- Cross ARM C Compiler
  - Preprocessor
  - Includes
  - Optimization
  - Warnings
  - Miscellaneous
- Cross ARM C Linker
  - General
  - Libraries
  - Miscellaneous

Libraries (-l)

```
ksdk_platform_mqx
_mfs
_nshell
_mqx
usbh_mqx
_rtcs
_mqx_stdlib
gcc
c
supc++
m
```

Library search path (-L)

```
"${ProjDirPath}/../../../../../../lib\ksdk_mqx_lib\kds\K64F12\Debug"
"${ProjDirPath}/../../../../../../..\filesystem\mfs\lib\frdmk64f.kds\debug\mfs"
"${ProjDirPath}/../../../../../lib/frdmk64f.kds/debug/nshell"
"${ProjDirPath}/../../../../../lib/frdmk64f.kds/debug/mqx"
"${ProjDirPath}/../../../../../../rtos\mqx\lib\frdmk64f.kds\debug\mqx_stdlib"
"${ProjDirPath}/../../../../../../usb\usb_core\host\build\kds\usbh_sdk_frdmk64f_mqx\Debug"
"${ProjDirPath}/../../../../../../tcpip\rtcs\lib\frdmk64f.kds\debug\rtcs"
```

Figure 9 Linker library settings from FRDM-K64F example

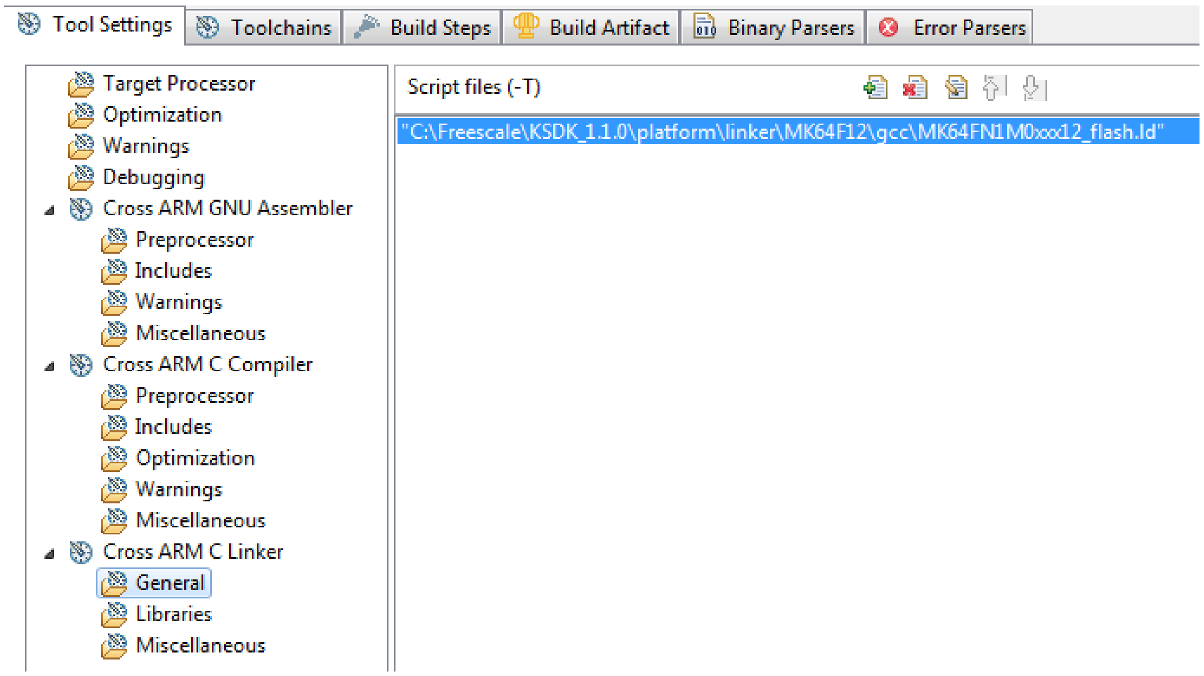
6.4.2 Linker command file

The command file, which the linker uses to add the code and the symbols to the memory, has also been moved and renamed. Update the linker settings with the new linker file:

- *"${KSDK_PATH}/platform/linker/<derivative>/<tool>*

Note

New KDS IDE Projects created with KSDK support can use the default linker file.



Script files (-T)

"C:\Freescale\KSDK_1.1.0\platform\linker\MK64F12\gcc\MK64FN1M0xxx12_flash.ld"

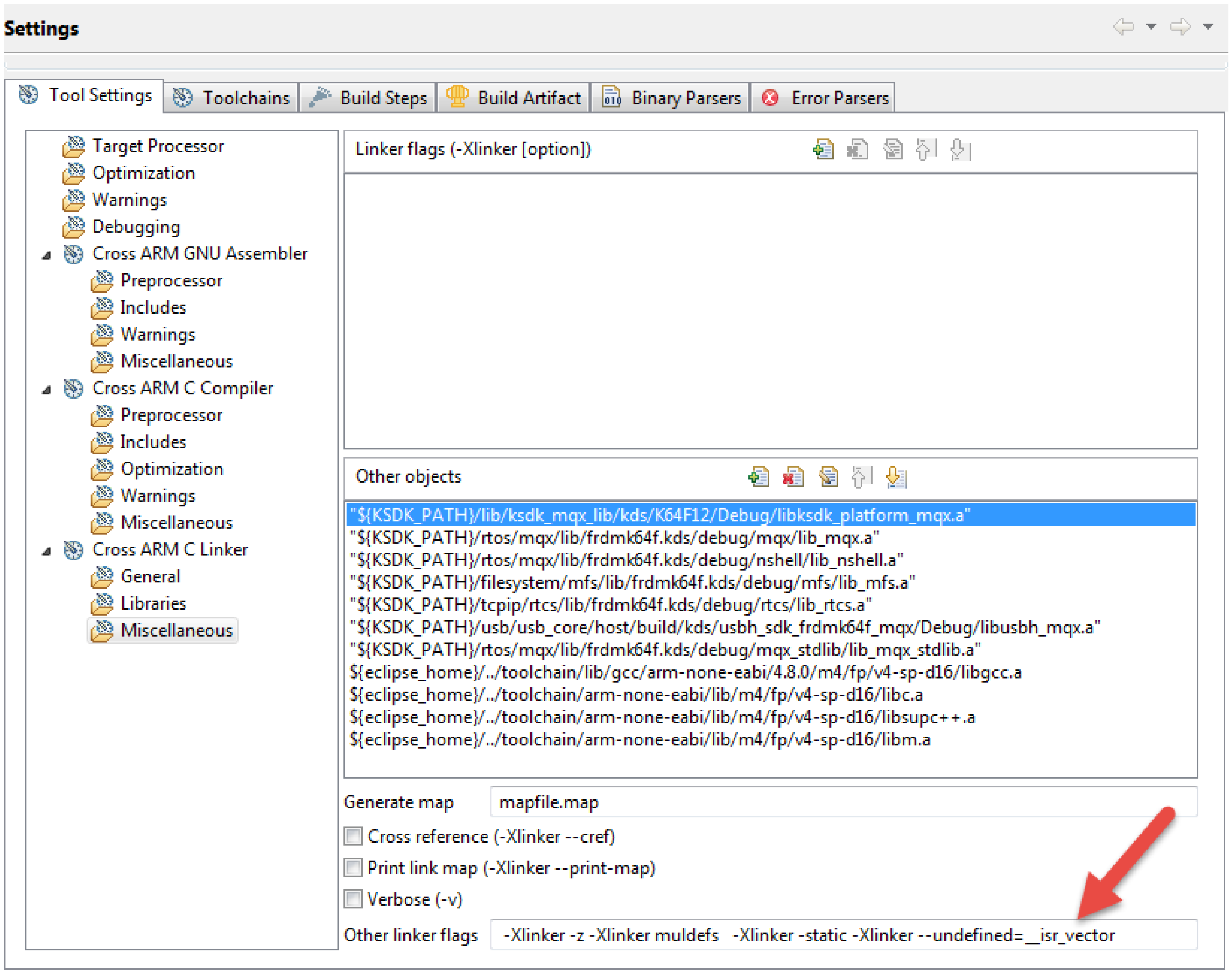
Figure 10 Linker command file settings from FRDM-K64F example

6.5 Toolchain-specific changes

In addition to the common toolchain changes in the previous sections, some toolchains supporting classic MQX RTOS require additional changes in the application project to build correctly with MQX RTOS for Kinetis SDK.

6.5.1 Kinetis Design Studio (KDS)

The GCC linker used in KDS IDE strips out the vector table in the Kinetis SDK library unless a flag is added to the linker settings. To prevent this issue, add the flag "-Xlinker --undefined=__isr_vector" to the other linker flags field in the KDS IDE linker settings.



```
"${KSDK_PATH}/lib/ksdk_mqx_lib/kds/K64F12/Debug/libksdk_platform_mqx.a"
"${KSDK_PATH}/rtos/mqx/lib/frdmk64f.kds/debug/mqx/lib_mqx.a"
"${KSDK_PATH}/rtos/mqx/lib/frdmk64f.kds/debug/nshell/lib_nshell.a"
"${KSDK_PATH}/filesystem/mfs/lib/frdmk64f.kds/debug/mfs/lib_mfs.a"
"${KSDK_PATH}/tcpip/rtcs/lib/frdmk64f.kds/debug/rtcs/lib_rtcs.a"
"${KSDK_PATH}/usb/usb_core/host/build/kds/usbh_sdk_frdmk64f_mqx/Debug/libusbh_mqx.a"
"${KSDK_PATH}/rtos/mqx/lib/frdmk64f.kds/debug/mqx_stdlib/lib_mqx_stdlib.a"
${eclipse_home}/../toolchain/lib/gcc/arm-none-eabi/4.8.0/m4/fp/v4-sp-d16/libgcc.a
${eclipse_home}/../toolchain/arm-none-eabi/lib/m4/fp/v4-sp-d16/libc.a
${eclipse_home}/../toolchain/arm-none-eabi/lib/m4/fp/v4-sp-d16/libsupc++.a
${eclipse_home}/../toolchain/arm-none-eabi/lib/m4/fp/v4-sp-d16/libm.a
```

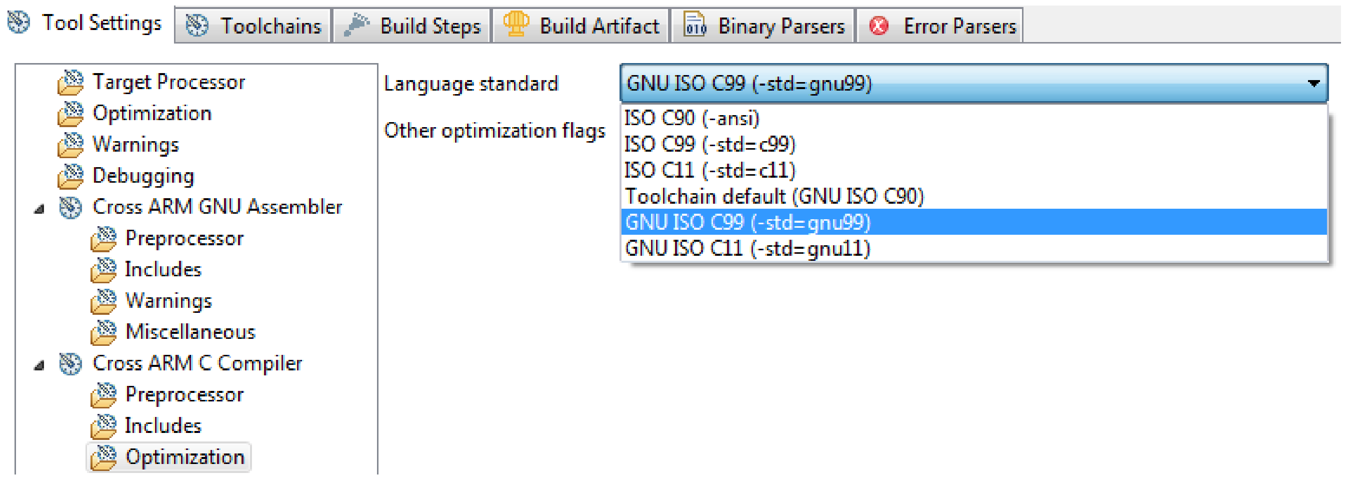
Generate map: mapfile.map

- [ ] Cross reference (-Xlinker --cref)
- [ ] Print link map (-Xlinker --print-map)
- [ ] Verbose (-v)

Other linker flags: -Xlinker -z -Xlinker muldefs -Xlinker -static -Xlinker --undefined=__isr_vector

Figure 11 Linker flag required in KDS application project settings

Ensure that the language standard is set to the “*GNU ISO* C99 (-std=gnu99)” in the menu **Project > Properties C/C++ Build > Settings > Cross ARM C Compiler > Optimization**.

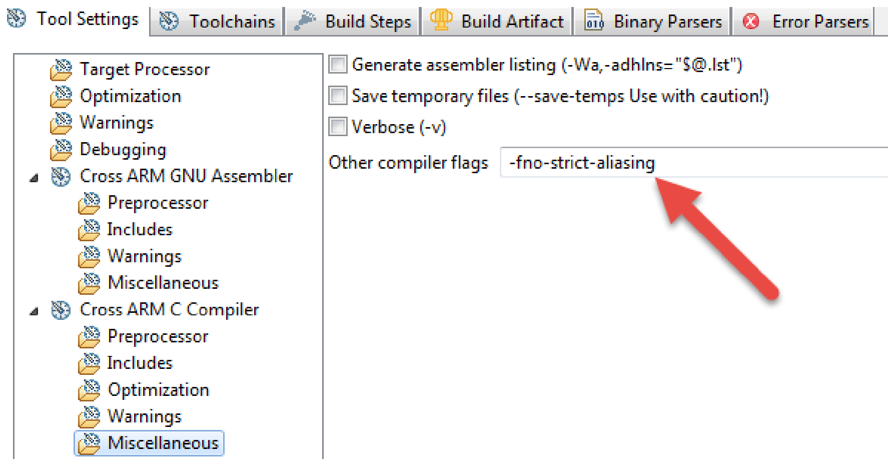


Language standard: GNU ISO C99 (-std=gnu99)

Other optimization flags

Figure 12 Language standard settings in KDS IDE

Add the *“-fno-strict-aliasing*” to other compiler flag box in the menu **Project** > **Properties C/C++ Build** > **Settings** > **Cross ARM C Compiler** > **Miscellaneous**.



Generate assembler listing (-Wa,-adhlns="$@.lst")

Save temporary files (--save-temps Use with caution!)

Verbose (-v)

Other compiler flags: -fno-strict-aliasing

Figure 13 Other compiler flags in KDS IDE

6.6 Application project changes

Additional changes in the application source files enable the project to build with the Kinetis SDK. To continue porting, add these folders with the indicated source files to the project. The folders can be copied into the project or linked from their original location which is also indicated below. See Figure 12.

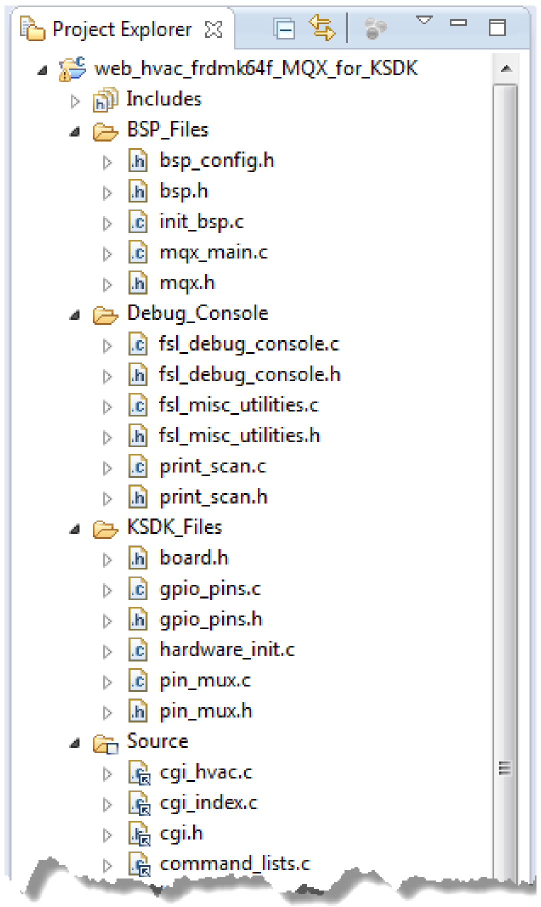
- BSP_Files

- ${KSDK_PATH}/rtos/mqx/mqx/source/bsp
  - init_bsp.c
  - init_hardware.c
  - mqx_init.c
  - mqx_main.c
- ${KSDK_PATH}/rtos/mqx/mqx/source/include
  - mqx.h

- KSDK_ Files
  - ${KSDK_PATH}/boards/<board>
    - board.h
    - gpio_pins.c
    - gpio_pins.c
    - hardware_init.c
    - pin_mux.c
    - pin_mux.c
- Debug_Console
  - ${KSDK_PATH}/platform/utilities/inc
    - fsl_debug_console.h
    - fsl_misc_utilities.h
  - ${KSDK_PATH}/platform/utilities/src
    - fsl_debug_console.c
    - fsl_misc_utilities.c
    - print_scan.c
    - print_scan.h

Note

Classic MQX RTOS projects which use the USB Host stack include the **usb_classes.c** source file. Because MQX RTOS for Kinetis SDK does not use this file, the file should be removed from the application project.

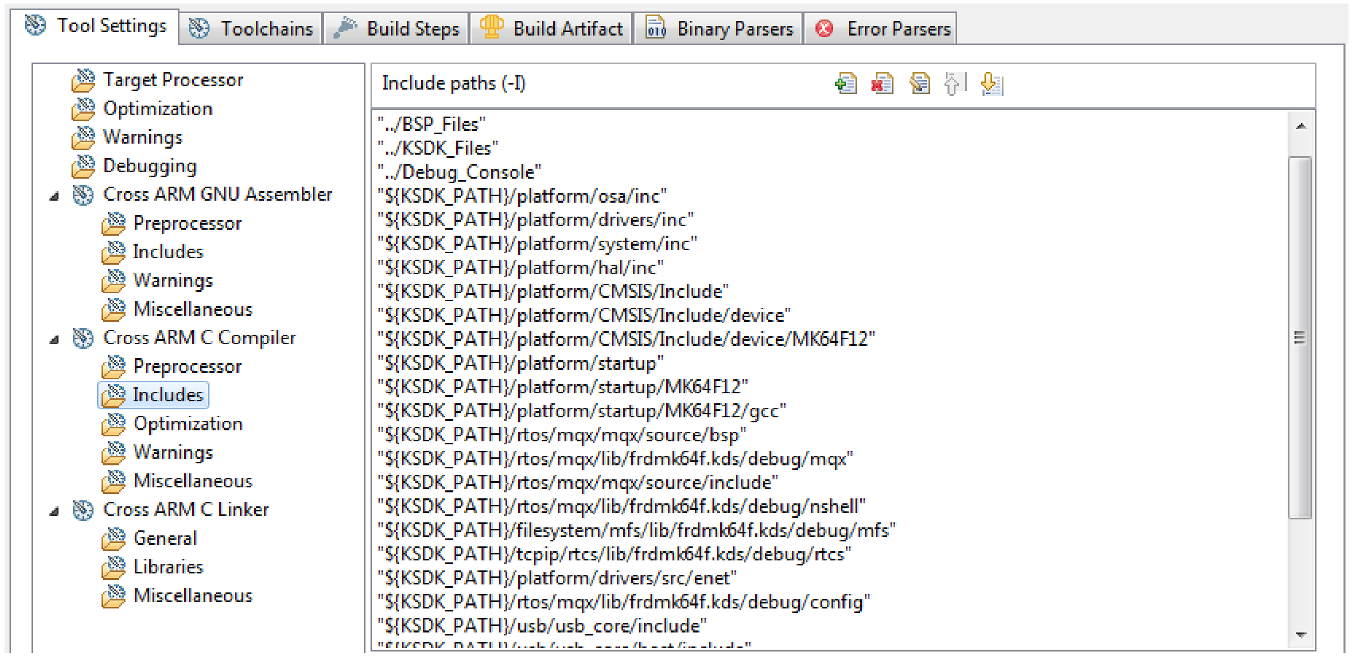

Project Explorer

web_hvac_frdmk64f_MQX_for_KSDK
- Includes
- BSP_Files
  - bsp_config.h
  - bsp.h
  - init_bsp.c
  - mqx_main.c
  - mqx.h
- Debug_Console
  - fsl_debug_console.c
  - fsl_debug_console.h
  - fsl_misc_utilities.c
  - fsl_misc_utilities.h
  - print_scan.c
  - print_scan.h
- KSDK_Files
  - board.h
  - gpio_pins.c
  - gpio_pins.h
  - hardware_init.c
  - pin_mux.c
  - pin_mux.h
- Source
  - cgi_hvac.c
  - cgi_index.c
  - cgi.h
  - command_lists.c

Figure 14 Additional source files added to application project

Add these folders paths in menu **Project > Properties C/C++ Build > Settings > Cross ARM C Compiler > Includes > Include paths (-I)**.

Note that if creating a virtual folder, add the path where the original files are located.



Include paths (-I)

```
"../BSP_Files"
"../KSDK_Files"
"../Debug_Console"
"${KSDK_PATH}/platform/osa/inc"
"${KSDK_PATH}/platform/drivers/inc"
"${KSDK_PATH}/platform/system/inc"
"${KSDK_PATH}/platform/hal/inc"
"${KSDK_PATH}/platform/CMSIS/Include"
"${KSDK_PATH}/platform/CMSIS/Include/device"
"${KSDK_PATH}/platform/CMSIS/Include/device/MK64F12"
"${KSDK_PATH}/platform/startup"
"${KSDK_PATH}/platform/startup/MK64F12"
"${KSDK_PATH}/platform/startup/MK64F12/gcc"
"${KSDK_PATH}/rtos/mqx/mqx/source/bsp"
"${KSDK_PATH}/rtos/mqx/lib/frdmk64f.kds/debug/mqx"
"${KSDK_PATH}/rtos/mqx/mqx/source/include"
"${KSDK_PATH}/rtos/mqx/lib/frdmk64f.kds/debug/nshell"
"${KSDK_PATH}/filesystem/mfs/lib/frdmk64f.kds/debug/mfs"
"${KSDK_PATH}/tcpip/rtcs/lib/frdmk64f.kds/debug/rtcs"
"${KSDK_PATH}/platform/drivers/src/enet"
"${KSDK_PATH}/rtos/mqx/lib/frdmk64f.kds/debug/config"
"${KSDK_PATH}/usb/usb_core/include"
"${KSDK_PATH}/usb/usb_core/host/include"
```

Figure 15 Additional include directories added to application project

6.7 Application source code changes

After the project settings have been updated, modify the application source files to build with MQX RTOS for Kinetis SDK.

6.7.1 Port application to Kinetis SDK drivers

Because the peripheral drivers in Kinetis SDK are different than the classic MQX RTOS drivers, these changes are likely the most significant for the porting effort. Kinetis SDK includes documentation and examples for all the new drivers and the HAL APIs. Use these references to find similar driver functionality in Kinetis SDK and update the application.

This is an example from porting the web_hvac demo to MQX RTOS for Kinetis SDK. This demo uses the LWGPIO driver from classic MQX RTOS to control the LEDs and read the input switches. The demo was ported to the Kinetis SDK GPIO driver with these changes in hvac_io.c file:

- Header files included in source code
  - Remove include of ***lwgpio.h*** form *hvac_io.c*
- Initialize driver
  - Classic MQX RTOS used ***lwgpio_init()*** for the LED and button pins initialization. E.g.

```
lwgpio_init(&led1, LED_1, LWGPIO_DIR_OUTPUT, LWGPIO_VALUE_NOCHANGE);
lwgpio_init(&button1, TEMP_PLUS, LWGPIO_DIR_INPUT, LWGPIO_VALUE_NOCHANGE);
```

  - MQX RTOS for KSDK uses ***GPIO_DRV_Init()***. In hvac_io.c remove all the classic MQX RTOS pin initializations and call the KSDK driver init function.

```
bool HVAC_InitializeIO(void)
{
    /* Init Gpio for Leds and switches */
        GPIO_DRV_Init(switchPins, ledPins);

    return (input_port!=0) && (output_port!=0);
}
```

- Set the output pin to the LED
  - Classic MQX RTOS uses ***lwgpio_set_value()***. E.g.

```
lwgpio_set_value(&led1, LWGPIO_VALUE_HIGH);
```

  - MQX RTOS for Kinetis SDK uses ***GPIO_DRV_WritePinOutput()***. In hvac_io.c replace all the classic MQX RTOS calls for the KSDK driver function.

```
void HVAC_SetOutput(HVAC_Output_t signal,bool state)
{
   if (HVAC_OutputState[signal] != state) {
      HVAC_OutputState[signal] = state;
      if (output_port) {
          switch (signal) {
              case HVAC_FAN_OUTPUT:
                  (state) ? GPIO_DRV_WritePinOutput(kGpioLED1, 1):GPIO_DRV_WritePinOutput(kGpioLED1, 0);
                   break;
              case HVAC_HEAT_OUTPUT:
                  (state) ? GPIO_DRV_WritePinOutput(kGpioLED2, 1):GPIO_DRV_WritePinOutput(kGpioLED2, 0);
                   break;
```

```
            case HVAC_COOL_OUTPUT:
                (state) ? GPIO_DRV_WritePinOutput(kGpioLED3, 1):GPIO_DRV_WritePinOutput(kGpioLED3, 0);
                  break;
          }
       }
    }
}
```

- Read input pin from push button
    - Classic MQX RTOS uses ***lwgpio_get_value()***. E.g.

```
value = lwgpio_get_value(&button1);
```

-
    - MQX RTOS for Kinetis SDK uses ***GPIO_DRV_ReadPinInput ()***. In hvac_io.c replace all the classic MQX RTOS calls for the KSDK driver function.

```
bool HVAC_GetInput(HVAC_Input_t signal)
{
   bool  value=FALSE;
    if (input_port){
        switch (signal) {
            case HVAC_TEMP_UP_INPUT:
                value = !GPIO_DRV_ReadPinInput(kGpioSW1);
                break;
            case HVAC_TEMP_DOWN_INPUT:
                value = !GPIO_DRV_ReadPinInput(kGpioSW2);
                break;
#if defined(FAN_ON_OFF)
         case HVAC_FAN_ON_INPUT:
            value = !GPIO_DRV_ReadPinInput(kGpioSW3);
            break;
#endif
   }
  }
   return value;
}
```

6.7.2 Adjust task priorities

The ported application may need the application task priorities adjusted to use the Kinetis SDK features.

6.7.2.1 Operating System Abstraction (OSA) priority scheme used by the Kinetis SDK

To support multiple Operating Systems (OS), the Kinetis SDK provides an Operating System Abstraction (OSA) layer that is used as interface between the KSDK stacks and operating systems.

Because each supported Operating System (OS) uses a different priority scheme, any tasks created by the OSA API, require their OSA priority to be mapped to the corresponding OS priority. OSA supports application tasks priorities from 0 to unlimited, where the priority 0 is the highest priority and higher numbers equal lower priority.

MQX RTOS supports application tasks priorities from 7 to unlimited, where the priority 7 is the highest priority and higher numbers equal lower priority. MQX RTOS task priorities 0 to 6 are used directly by operating system.

Any task created using the OSA API function "OS_Task_create()"use OSA priority levels. Any task created using the MQX RTOS API function "_task_create()"use MQX RTOS priority levels.

Therefore, application developers should keep in mind the priority mapping of OSA to MQX RTOS, which is:

MQX RTOS Priority = OSA Priority + 7

6.7.2.2 Adjustment to the application task priorities to accommodate the default USB stack task priorities

One example where adjustment to the task priorities is necessary occurs when using the Kinetis SDK USB Host stack. This library creates two tasks for the USB host controller. Tasks should be set to a higher priority (lower task priority number) than the other application tasks which uses their functionality to ensure that the stack can operate properly.

The USB Host stack defines these macros:

```
  USBCFG_HUB_TASK_PRIORITY    (set to OSA priority 7 by default)
USBCFG_HOST_KHCI_TASK_PRIORITY (set to OSA priority 8 by default)
```

This corresponds to MQX RTOS task priorities 14 and 15.

Therefore, to ensure the USB stack operates properly, all other application tasks and stack task priorities should be set to 16 or higher (higher the number = lower the priority).

The RTCS stack creates tasks with priority set by the macro RTCSCFG_DEFAULT_RTCSTASK_PRIO. By default, this is set to MQX RTOS priority 6. Since web server in web_hvac example reads from USB, RTCSCFG_DEFAULT_RTCSTASK_PRIO in this case should be adjusted to MQX RTOS priority level 16.

Note that the RTCS creates tasks at RTCSCFG_DEFAULT_RTCSTASK_PRIO and RTCSCFG_DEFAULT_RTCSTASK_PRIO+1. Therefore, the application tasks should start from MQX RTOS priority level 18.

6.7.2.3 Example web_hvac task priorities

Example of task priorities:

Task	MQX RTOS Priority Level
USBCFG_HUB_TASK	14 (Highest Priority) [Set by USB stack using OSA priority 7]
USBCFG_HOST_KHCI_TASK	15 [Set by USB stack using OSA priority 8]
RTCSCFG_DEFAULT_RTCSTASK	16
USB_TASK	18
HVAC_TASK	19
SWITCH_TASK	20
ALIVE_TASK	21
LOGGING_TASK	22

SHELL_TASK	23 (Lowest Priority)

Note

USB_TASK is an application task in web_hvac for handling USB data in the application. This is not to be confused with the USB stacks tasks. It should be a lower priority than the USB stack tasks and RTCS stack tasks.

Note

When MQX_LITE_CONFIG is chosen with static allocations, maximum priority level is limited by the MQXCFG_LOWEST_TASK_PRIORITY macro. It has to be changed to at least level 23 in this case.

6.7.2.4 Alternative example web_hvac task priorities

The USBCFG_HUB_TASK_PRIORITY can be decreased to (e.g. OSA priority 0) and USBCFG_HOST_KHCI_TASK_PRIORITY (e.g. OSA priority 1). The priorities of the application and RTCS stack task can be appropriately adjusted. The operation of the system is the same as above.

Task	MQX RTOS Priority Level
USBCFG_HUB_TASK	7 (Highest Priority) [Set by USB stack using OSA priority 0]
USBCFG_HOST_KHCI_TASK	8 [Set by USB stack using OSA priority 1]
RTCSCFG_DEFAULT_RTCSTASK	9
USB_TASK	11
HVAC_TASK	12
SWITCH_TASK	13
ALIVE_TASK	14
LOGGING_TASK	15
SHELL_TASK	16 (Lowest Priority)

Note

The macros PRIORITY_OSA_TO_RTOS(osa_prio) and PRIORITY_RTOS_TO_OSA(rtos_prio) help to recalculate right priority level.

6.8 Other changes

Additional minor changes are necessary to ensure that the application can build with the Kinetis SDK. Use the Kinetis SDK documentation and code examples as references for these changes.

These are some examples encountered while porting the web_hvac demo:

1. With the changes to the USB Host stack and MFS file system, the web_hvac demo was updated to enable mounting the file system on a USB flash drive. The header file usbmfs.h from classic

MQX RTOS is no longer used in Kinetis SDK, and the inclusions to this file were removed from the application. Also, the application USB task was updated using the example project at /filesystem/mfs/examples/usbdisk. These files were taken from this example and used in the application:

- usb_file.h
- usb_file.c
- main.c (usbdisk) -> usb_task.c (web_hvac)

2. Inclusions need to be added for stdio.h in any source files using those features. After the STDIO features have been pulled out of the PSP library and are included in mqxstd_lib, they are no longer included in the application through mqx.h.
3. In MQX RTOS for Kinetis SDK, the TFS driver has been renamed to nio_tfs. The functionality is the same, but web_hvac needed these changes to continue using that driver:

- Change header file includes to nio_tfs.h
- Change structures and driver APIs to nio_tfs

4. Some types and declarations have changed. For example, the type MQX_FILE_PTR used in web_hvac is no longer defined in MQX RTOS for Kinetis SDK. The demo was updated to use type “int” instead.

7 Conclusion

Porting an application from classic MQX RTOS to MQX RTOS for Kinetis SDK requires changes which are manageable with the documentation and examples provided. After the transition is made, embedded system developers will benefit from the Kinetis SDK architectural advantages, while still leveraging the benefits of the popular MQX RTOS kernel, stacks, and middleware.

Visit www.freescale.com/mqx for updates and information about all Freescale MQX RTOS software.

Additionally, visit the freescale.com/community/mqx to access the moderated online community and get answers to questions, tips and tricks, and other helpful resources related to MQX RTOS.

How to Reach Us:

Home Page:
freescale.com

Web Support:
freescale.com/support

Document Number: MQXKSDKPUG
Rev. #0
03/2015