Freescale Semiconductor
User's Guide

Document Number: MQXKSDKGSUG
Rev. 1, 04/2015

Getting Started with MQX™ RTOS for Kinetis SDK

1 Introduction

This document describes the steps required to configure supported development tools used to build, run, and debug applications with the Freescale MQX™ RTOS operating system targeted for Kinetis Software Development Kit (KSDK).

2 MQX RTOS role in the KSDK

The Kinetis Software Development Kit (KSDK) is a software framework for developing applications on Kinetis MCUs. The software components in the framework include hardware abstraction layers (HAL), peripheral drivers, libraries, middleware, and more. The Kinetis SDK also includes an operating system abstraction (OSA) layer for application developers to extend the capabilities of the KSDK with the addition of a supported real time operating system (RTOS).

This version of Freescale MQX RTOS adopts the Kinetis SDK architecture for the platform features that the KSDK provides, and adds additional components that have traditionally been available within standard MQX RTOS release packages.

Contents

Application developers can now use MQX RTOS, communication stacks, file system, and other components with KSDK libraries and peripheral drivers.

3 Building the MQX RTOS libraries

Basic MQX RTOS based user application is created from application code itself and from set of libraries. Text below will provide a guide describing build process and basic configuration:

- MQX RTOS Kernel and core components mqx.a (mandatory)
- KSDK HAL and driver library built in the MQX RTOS configuration libksdk_platform_mqx.a (mandatory)
- MQX STDLIB - Subset of Standard C-Library - mqx_stdlib.a (optional)
- MFS - FAT12/16/32 filesystem mfs.a (optional)
- RTCS TCPIPv4/v6 stack rtcs.a (optional)
- USB Host and Device stack (optional)

This document assumes the Kinetis SDK and MQX RTOS are installed on your computer. It also assumes:

- <KSDK_DIR> is the directory where the Kinetis SDK package is installed on your hardware.
- <MQX_DIR> is the directory where MQX RTOS is located within KSDK, specifically <KSDK_DIR>/rtos/mqx.
- <board> replaces the name of the board (e.g., twrk64f120m).
- <mcu> replaces the name of the processor (e.g., mk64f120m).
- <tool> replaces the name of the toolchain (e.g., IAR).
- <target> replaces the name of the project target (e.g., Debug).
- <library> replaces the name of the library (e.g., mqx_stdlib).

3.1 Build targets in libraries

Each build project in Freescale MQX RTOS contains multiple compiler/linker configurations called build “targets”.

Two different types of build targets exist for different compiler optimization settings:

- Debug – the compiler optimizations are turned off or set to low. The compiled code is easy to debug but may be less effective and much larger than the release build.
- Release – the compiler optimizations are set to maximum. The compiled code is very hard to debug and should be used for final applications only.

3.2 Compile-time configuration

The configuration of the board for the MQX RTOS application is specified in the configuration header files. Major compile-time configuration options for specific supported boards are grouped in the user configuration file located in:

<MQX_DIR>/config/board/<board>/user_config.h

Some of the board configuration settings are taken from the KSDK's board.h settings located in <KSDK_DIR>/boards/<board>/board.h

The processor and architecture specific configuration for MQX RTOS and KSDK MQX RTOS library is in the user configuration file located in:

<MQX_DIR>/config/mcu/<mcu>/mqx_sdk_config.h

The user configuration files are included internally from the libraries. The configuration for the board or processor can be retrieved from the files.

3.3 Library build process

After either the compile-time user configuration file or MQX kernel source files are changed, rebuild the MQX RTOS libraries. The minimalistic application use the MQX Kernel and KSDK Platform libraries, but most of the applications require additional components such as MQX RTOS stdlib library, MFS FAT32 Filesystem or RTCS TCPIP stack. Example application workspaces located next to the application projects contains all libraries required by given application.

After successful compilation of each MQX library all header files describing library API are copied to one directory:

- <MQX_DIR>/lib/<board>.<tool>/<board>/debug/<library>.

The necessary header files required for the application consist of:

- user_config.h
- mqx_sdk_config.h
- device driver public headers
- kernel public headers

KSDK Platform library consisting of device drivers and HAL is located in following directory:

- <KSDK_DIR>/lib/ksdk_mqx_lib/<tool>/<mcu>

4 MQX RTOS Example Application

MQX RTOS software provides a broad set of example applications. The examples are written to demonstrate the most frequently used features of the MQX RTOS and other components. Examples for RTOS kernel features are available in <MQX_DIR>/mqx/examples directory, RTCS TCPIP stack examples in <KSDK_DIR>/middleware/tcpip/rtcs/examples directory and MFS examples in <KSDK_DIR>/middleware/filesystem/mfs/examples directory.

4.1 Build targets in applications

Similary as Library projects, Example projects in Freescale MQX RTOS contains multiple compiler/linker configurations called build “targets”.

Two different types of build targets exist for different compiler optimization settings:

- Debug – the compiler optimizations are turned off or set to low. The compiled code is easy to debug but may be less effective and much larger than the release build. Application code is linked with libraries build with Debug setting:
    - <MQX_DIR>/lib/<board>.<tool>/<board>/debug/<library>/*.*
    - <KSDK_DIR>/lib/ksdk_mqx_lib/<tool>/<mcu>/debug/*.*
- Release – the compiler optimizations are set to maximum. The compiled code is very hard to debug and should be used for final applications only. Application code is linked with libraries build with Release setting:
    - <MQX_DIR>/lib/<board>.<tool>/<board>/release/<library>/*.*
    - <KSDK_DIR>/lib/ksdk_mqx_lib/<tool>/<mcu>/release/*.*

4.2 Building your first MQX RTOS application

To build one specific example application, use the workspace which contains all project files (library and application) for that specific example.

1. Open the workspace file for a specific example located in the <MQX_DIR>/mqx/examples/<example>/build/<tool>/<example>_<board>/<example>_<board>.*
2. Build the required targets (e.g., Debug) in all library projects contained in the workspace.
3. Build the required targets (e.g., Debug) in example application projects contained in the workspace.
4. Run the application.

4.3 Building your first MQX application with IAR EWARM IDE

For detailed information about the MQX RTOS support and building process in IAR tool, see *Getting Started with Freescale MQX™ RTOS and IAR Embedded Workbench* (document MQXGSIAR). The document is located in <KSDK_dir>/doc/rtos/mqx/MQX RTOS IDE Guides/.

4.4 Building your first MQX RTOS application with Freescale KDS IDE

For detailed information about the MQX RTOS support and building process in KDS IDE tool, see *Getting Started with Kinetis Design Studio IDE and Freescale MQX™ RTOS* (document MQXKDSUG). The document is located in <mqx_install_dir>/doc/rtos/mqx/MQX RTOS IDE Guides/.

4.5 Building your first MQX RTOS application with Keil µVision®4

For detailed information about the MQX RTOS support and building process in Keil µVision4 tool, see *Getting Started with Freescale MQX™ RTOS and MDK-ARM Keil™ µVision4®* (document MQXGSKEIL). The document is located in <mqx_install_dir>/doc/rtos/MQX RTOS IDE Guides/MQX-KSDK-uVision4-Getting-Started.pdf document.

4.6 Building your first MQX RTOS application with Atollic® TrueSTUDIO®

For detailed information about the MQX RTOS support and building process in Atollic TrueSTUDIO tool, see *Getting Started with Freescale MQX™ RTOS for Kinetis SDK and Atollic TrueSTUDIO® for ARM® (document KSDKGSATLUG)* <mqx_install_dir>/doc/rtos/mqx/MQX-KSDK-Atollic-Getting-Started.pdf document.

4.7 Building your first MQX RTOS application with cmake and GCC compiler

For detailed information about the MQX RTOS support and building process in cmake and GCC tool see <mqx_install_dir>/doc/rtos/mqx/MQX RTOS IDE Guides/MQX-KSDK-ARMGCC-Getting-Started.pdf document.

5 Building MFS or RTCS library

This section describes the build process for the MFS and RTCS libraries.

The MFS library implements Microsoft™ FAT file systems, specifically FAT12, FAT16, and FAT32.

The RTCS library is a library for the POSIX-like TCP/IP stack.

In the subsequent sections, these placeholders are used:

- <MFS_DIR> is the location of MFS directory, specifically <KSDK_DIR>/filesystem/middleware/mfs
- <RTCS_DIR> is the location of RTCS directory, specifically <KSDK_DIR>/middleware/tcpip/rtcs

5.1 MFS/RTCS library build process

After the compile-time user configuration file or MFS/ RTCS configuration files are changed, rebuild the MFS/ RTCS library. The workspace with library can be found at this location:

- MFS library: <MFS_DIR>/build/<tool>/mfs_<board>
- RTCS library: <RTCS_DIR>/build/<tool>/rtcs_<board>

5.2 Building your first MFS application

This example of a build procedure is based on IAR Embedded Workbench.

1. Build the MQX RTOS libraries as described in section 4.
2. Open the IAR EWARM workspace file. An example is located at: <MFS_DIR>/examples/<example_name>/build/<tool>/<example_name>_<board>

   e.g., <MFS_DIR>/examples/ramdisk/build/iar/ramdisk_twrk64f120m/ramdisk_twrk64f120m.eww.

   The workspace contains the necessary library projects which should be built before the example. In this case, the MFS library – mfs_twrk64f120m.ewp.
3. After building the MFS library, build the Debug target of the application project.
4. Run the application.

6 The I/O subsystem NIO

The file streaming functionality was moved from the MQX RTOS PSP library into MQX RTOS stdlib library. The NIO covers only the I/O file descriptor handling.

The MQX RTOS for KSDK 1.2.0 comes with new I/O subsystem called NIO. The NIO subsystem was designed for application developers and device driver developers to handle variety of devices with POSIX-compliant API.

The application developer should refer to the POSIX standard to use the device drivers registered in the NIO.

7 Differences between MFS and FatFS

In the basic KSDK installation, you can find the open source FatFS filesystem library located in <KSDK_DIR>/middleware/filesystem/fatfs.

The MQX RTOS uses its own specific MFS filesystem library, installed in <KSDK_DIR>/middleware/filesystem/mfs.

It is up to the application developer to choose which package to use.

This table addresses most questions regarding file systems contained in KSDK.

Table 1. File systems in KSDK

Criteria	MFS	FatFS	Measurement conditions, remarks
Reentrancy	YES	NO	The FatFS supports reentrancy with _FS_REENTRANT, but it is set to 0 by default. No adaptation for supported RTOS in KSDK is provided.
API compliancy	POSIX	ANSI-C	The FatFS uses its own API. The POSIX-compliant API provided by MFS simplifies porting applications to/from other POSIX systems.
Multiple partitions	YES	NO	
Multiple instances	YES	NO	
SDCARD driver	YES	YES	
USB MSD driver	YES	YES	
RAMDISK driver	YES	NO	

8 Revision History

This table summarizes revisions to this document.

Table 1 Revision History

Revision number	Date	Substantial changes
1	04/2015	Kinetis SDK 1.2.0 release
0	12/2014	Kinetis SDK 1.1.0 release

Document Number MQXKSDKGSUG
Revision 1, 04/2015