Document Number: MQXKSDK110RN
Rev 0, 12/2014

Freescale MQX™ RTOS for Kinetis SDK 1.1.0 Release Notes

Contents

1 Read Me

This is the release notes for Freescale MQX™ RTOS for Kinetis SDK.

1.1 Development Tools Requirements

Freescale MQX RTOS was compiled and tested with these development tools:

- Kinetis Development Studio version 2.0.0
  - See build projects in `kds` subdirectories
- IAR Embedded Workbench for ARM® Version 7.20.2
  - See build projects in `iar` subdirectories
- ARM-MDK - Keil® µVision® version 5.11.0
  - See build projects in the `uv4` subdirectories
- Atollic® TrueSTUDIO® for ARM version 5.2.0
  - See build projects in the atl subdirectories
- Cmake version 3.0 support for GCC compiler revision 4.8.3 2014q2 from ARM Embedded
  - See Cmake definition files in the armgcc subdirectories

1.2 System Requirements

System requirements are based on the requirements for the development tools. There are no special host system requirements for hosting the Freescale MQX RTOS distribution itself. The code was tested in the Windows 7® operating system and Ubuntu 14.04 host environment.

The minimum PC configuration is determined by the development tools. The recommended PC configuration is 2 GHz processor, 2 GB RAM and 2 GB free disk space.

1.3 Target Requirements

Freescale MQX RTOS supports the evaluation boards and device families mentioned below. There are no special requirements for the target hardware other than what each board requires for its operation (power supply, cabling, jumper settings etc). For more details about the board-specific setup for MQX RTOS applications, see Kinetis Software Development Kit (SDK) board-related documentation.

Table 1. Device family and valuation boards supported

Device Family	Boards
MK24F12	TWR-K64F120M, FRDM-K64F
MK63F12	
MK64F12	
MK22F12810	TWR-K22F120M, FRDM-K22F
MK22F25612	
MK22F51212	
MKV31F12810	TWR-KV31F120M
MKV31F25612	
MKV31F51212	
MK24F25612	TWR-K24F120M
MK60D10	TWR-K60D100M
MKL46Z4	FRDM-KL64Z
MKV10Z	TWR-KV10Z75M
MKV30F12810	TWR-KV31F120MKV30
MK02F12810	FRM-K22FK02, FRDM-K22FK0264, TWR-K22FK02

1.4 Set up installation instructions and technical support

MQX RTOS source code and build projects are distributed as part of the Kinetis SDK package. To install MQX RTOS, select the corresponding option during the Kinetis SDK installation. This package contains the Kinetis SDK driver set, MQX RTOS kernel, RTCS - MQX RTOS TCP/IP stack and MFS - MQX RTOS File System.

It is recommended that you install Kinetis SDK to a path without spaces to avoid build problems with certain tool chains.

For a description of available support including commercial support options, visit the MQX RTOS support site on freescale.com.

2 What is new

This section describes the major changes and new features implemented in this release.

- This is the first official release of MQX RTOS for Kinetis SDK. See Release Content for component version information. Note that most components are based on version 4.1.2 while MQX RTOS kernel was updated by support of MQX Lite configuration and it is released under version 5.0.1
- Peripheral I/O drivers are fully based on the Kinetis SDK driver set. MQX RTOS provides POSIX wrappers for the I/O console and filesystem only.
- The file API was changed to comply with POSIX standard (FILE* is used instead of legacy MQX_FILE)
- MQX RTOS provides POSIX-based I/O drivers for serial console (tty, uart), file system, Ramdisk and other MQX RTOS native I/O drivers (TFS, NULL, PIPE ...)
- MQX RTOS provides its own re-entrant stdlib.h implementation.
- Support of MQX Lite configuration and new example application rtos/mqx/examples/. There are provided options for creating tasks from statically allocated memory and application can define these tasks before MQX RTOS starts (using create_task() API).
- Optional support for RTCS IPv6 (available as separate package).
- Optional support for CyaSSL stack. Demonstrated by HTTPs example application.
- MFS is integrated with USB stack, demonstrated in MFS example application.
- MQX RTOS Boards Support Package library has been removed and board-related configuration is now directly taken from Kinetis SDK framework.

3 Release Content

MQX RTOS for KSDK 1.1.0 consists of components in the following versions:

MQX RTOS Kernel components	version: 5.0.1
RTCS TCP/IP4 stack	version: 4.1.2
RTCS TCP/IP6 stack (optional)	version: 4.1.2
MFS FAT file system	version: 4.1.2
nShell command line interpreter	version: 1.0.1
MQX RTOS Standard Library	version: 1.0.1
CaySSL evaluation (optional)	version: 3.2.0

This table describes the release contents:

Table 2. Release Contents

Deliverable	Location
Configuration Files	`<install_dir>/rtos/mqx/config/...`
Mass-build project for all supported boards	`<MQX_DIR>/build/<tool>/<board>`
MQX RTOS PSP, and Examples	`<install_dir>/rtos/mqx/...`
MQX RTOS PSP source code for Kinetis ARM Cortex®-M core	`.../mqx/source/psp/cortex_m`
MQX RTOS BSP source code	`.../mqx/source/bsp/...`

Table continues on the next page...

Table 2. Release Contents (continued)

RTCS source code and examples	`<install_dir>/tcpip/rtcs/...`
MFS source code and examples	`<install_dir>/filesystem/mfs/...`
NShell Library Source Code	`<install_dir>/rtos/mqx/nshell/...`
Keil Task Aware Debugging plugin (TAD)	`<install_dir>/tools/mqx_plugins/keil_extensions/...`
IAR Task Aware Debugging plugin (TAD)	`<install_dir>/tools/mqx_plugins/iar_extensions/...`
KDS Task Aware Debugging plugin (TAD)	`<install_dir>/tools/mqx_plugins/kds_extensions/...`
Documentation	`<install_dir>/rtos/mqx/doc`

This figure shows the Freescale MQX RTOS directories installed to the user host computer (subdirectories reduced for clarity):

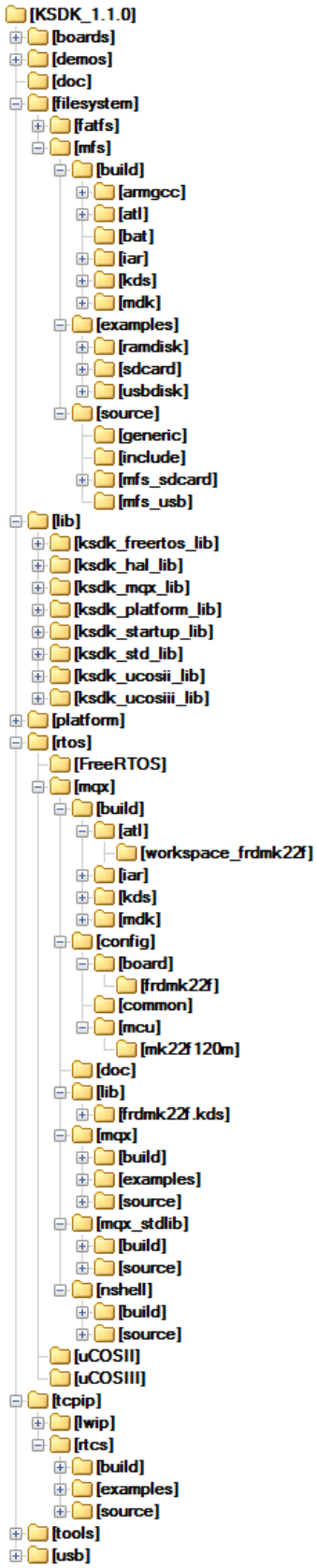

Figure 1. Freescale MQX RTOS Directories

4 MQX RTOS Release Overview

The Freescale MQX RTOS for Kinetis SDK consist of the following components:

- MQX RTOS real-time kernel
- TCP/IP networking stack (RTCS)
- FAT file system (MFS)
- Platform and Board support packages

4.1 MQX RTOS kernel

Freescale MQX RTOS for KSDK release contains ARM Cortex-M Platform Support only. Contact MQX RTOS support on freescale.com for other Freescale platforms.

The platform-specific code from `/mqx/source/psp/<platform>` is built together with the generic MQX RTOS core files. These two parts form a static library generally referred to as "MQX RTOS Library" which enables the target application to access RTOS features.

4.2 MQX RTOS BSPs

The board support library is no longer part of MQX RTOS. The peripheral drivers and board adaptation are provided by the KSDK framework and are part of user applications. MQX RTOS still contains several board-specific files supporting board-related configuration (<install_dir>/rtos/mqx/mqx/source/bsp directory):

- mqx_main.c - MQX initialization structure, for example heap size, interrupt settings, and task template.
- init_bsp.c - init_task code (installation of NIO drivers, custom MQX RTOS drivers, system tick initialization)
- bsp.h; bsp_config.h - default values for the MQX RTOS initialization

4.3 I/O drivers supported

I/O drivers in MQX RTOS for Kinetis SDK are based on peripheral drivers and HAL layers provided as a part of the Kinetis SDK framework. For some of these drivers, MQX RTOS provides POSIX-compliant API wrappers. MQX RTOS also provides a set of platform independent drivers which simplify apps coding.

The driver code is located in `<install_dir>/rtos/mqx/mqx/source/nio/drivers`.

The following list describes POSIX based I/O drivers available in the latest MQX RTOS release. The full set of peripheral I/O drivers (non POSIX based) can be found in the Kinetis SDK documentation

nio_dummy

The dummy driver does very little. This driver internally keeps some information on how many bytes have been read or written per file and per device.

nio_null

The null driver does nothing. This is the simplest possible driver, and is usually used as a template for new drivers.

nio_mem

This driver provides I/O operations on memory block. The memory block is specified by address and size.

nio_pipe

This driver provides a FIFO buffer where I/O operations are used to access the buffer.

nio_serial

This driver is a NIO subsystem wrapper built on top of the Kinetis SDK UART driver. This driver is mainly used by nio_tty driver.

nio_tfs

Trivial Filesystem is used as a simple read-only file repository instead of the fully featured MFS. TFS is not installed in the BSP startup code. Applications must initialize the TFS and pass a pointer to the filesystem data image. The mktfs tool is available (both as executable and Perl script) to generate the image from the existing directory structure. The RTCS HTTP example demonstrates the use of TFS.

nio_tty

Tty driver is installed on top of driver used for standard input/output. This driver provides basic formatting for terminal such as echo and end of line handling. When the MQX RTOS starts, nio_tty driver is installed on the top of nio_serial driver. It is then opened for stdin, stdout, and stderr.

4.4 Default I/O Channel

An I/O communication device installed by MQX RTOS BSP can be used as the standard I/O channel.

4.5 MQX RTOS PSP and BSP Directory Structure

RTOS files are located in the /rtos/mqx subdirectory of the Freescale MQX RTOS installation.

4.6 MQX RTOS MFS

MFS files from the `/filesystem/mfs/source` directory are built into a static library. When linked to the user application, the MFS library enables the application to access FAT12, FAT16, or FAT32-formatted drives.

4.7 MQX RTOS RTCS

RTCS files from the `/tcpip/rtcs/source` directory are built into a static library. When linked to the user application, the RTCS library enables the application to provide and consume network services of the TCP/IP protocol family.

The MQX RTOS 4.1.0 RTCS stack is IPv6 ready with respect to IPv6 Ready Logo certification and has passed all required tests. IPv6 support is available as a separate update package available from Freescale.

4.8 MQX RTOS USB Host and Device

USB Host and Device stack is provided as a part of the Kinetis SDK release. These stacks use an OS abstraction layer (OSA) to adapt to MQX RTOS. See USB documentation for details.

4.9 MQX RTOS nShell

The shell and command-line handling code is implemented as a separate library called nShell.

4.10 Building the MQX RTOS libraries

When using MQX RTOS for the first time and making changes to the compile-time user configuration file or MQX RTOS kernel source files, rebuild MQX RTOS libraries to ensure that the changes are propagated to the user applications.

4.11 Example applications

The examples are written to demonstrate the most frequently used features of the Freescale MQX RTOS. In addition to these demo applications, there are simpler example applications available in MQX RTOS, RTCS, MFS, and USB directories.

The tables summarize all demo and example applications provided in this release.

MQX RTOS Example Applications

Table 3. `<install_dir>/rtos/mqx/mqx/examples`

Name	Description
benchmark	Code size analysis example.
demo	Shows MQX RTOS multitasking and inter-process communication using standard objects like semaphores, events, or messages. See lwdemo for the same example using the lightweight objects.
event	Simple demonstration of MQX RTOS events.
hello	A trivial Hello World application spread across two tasks.
hello_lite	A trivial Hello World application spread across two tasks for lite configuration.
isr	Shows how to install an interrupt service routine and how to chain it with the previous handler.
klog	Shows kernel events being logged and later the log entries dumped on the console.
log	Shows the application-specific logging feature.
lwdemo	Same as the demo application, but implemented using lightweight components only.
lwdemo_lite	Same as the lwdemo application, but for lite configuration.
lwevent	Simple demonstration of MQX RTOS lightweight events.
lwlog	Simple demonstration of MQX RTOS lightweight log feature.
lwmsgq	Simple demonstration of MQX RTOS lightweight inter-process messaging.
lwsem	Simple demonstration of MQX RTOS task synchronization using the lightweight semaphore object.
msg	Simple demonstration of MQX RTOS inter-process message passing.
mutex	Simple demonstration of MQX RTOS task synchronization using the mutex object.
nill	Even simpler than Hello World. A void application which may be used for copy/paste to start custom application.
sem	Simple demonstration of MQX RTOS task synchronization using the semaphore object.
taskat	Shows how task can be created within statically allocated memory buffer (avoid heap allocation for task stack and context).
taskq	Shows custom task queue and how the queue can be suspended and resumed.

Table continues on the next page...

Table 3. `<install_dir>/rtos/mqx/mqx/examples` (continued)

Name	Description
test	Shows the self-testing feature of each MQX RTOS component.
timer	Simple demonstration of MQX RTOS timer component.
watchdog	Simple demonstration of the MQX RTOS task timeout detection using the kernel (not to be confused with watchdog) component.

RTCS Example Applications

Table 4. `<install_dir>/tcpip/rtcs/examples/...`

Name	Description
eth_to_serial	Simple character passing between the UART console and the telnet session. Shows custom "lightweight" telnet.
httpsrv	Simple web server with CGI-like scripts and web pages stored in internal flash.
shell	Shell command line providing commands for network management.
snmp	SNMP protocol example providing microprocessor state information.
benchmark	Throughput benchmark application. Fnet benchmark is required on the PC side.

MFS Example Applications

Table 5. `<install_dir>/rtos/filesystem/mfs/examples/...`

Name	Description
ramdisk	Shows use of MFS accessing the external RAM (or MRAM).

5 Known issues and limitations

Idle Task Required on Kinetis Platforms

The Kinetis kernel, by design, cannot operate without an idle task. The MQX_USE_IDLE_TASK configuration option must be set to 1.

UTF8 support in MFS

The UTF8 (CESU-8) support in MFS is limited to read-only access and for long file names. The UTF8 support for write access will be implemented in a future release.

Interrupt handling and priorities

The KSDK package allows the application developer to use the NVIC_* CMSIS functions. A set of functions handle the NVIC interrupt priorities. The MQX RTOS scheduler, however, limits the usage of interrupt priorities. Follow these criteria when using NVIC_SetPriority with MQX RTOS:

- priority level should be an even number
- priority level should be equal or higher than 2 times the value of MQX_HARDWARE_INTERRUPT_LEVEL_MAX value input in mqx_init structure if you want to use the MQX RTOS services in the interrupt service handler

These limitations give the application developer a maximum of seven levels of interrupt priorities.

ISR name in the MQX RTOS application versus the ISR name defined in the KSDK drivers

The ISR name in the MQX RTOS application should be different than the ISR name defined in the KSDK drivers. KSDK drivers are designed so that the vector table contains weak symbols to the driver ISR and the user is allowed to overload the weak vectors with the vectors specified by the user. This creates an issue because the MQX RTOS setup allows vector overloading with the _default_kernel_isr() function only. The vectors are then dispatched from the common interrupt handler to the user-specific handler with the MQX RTOS API.

To resolve this issue, keep the ISR name associated with MQX RTOS distinct and choose a different ISR name for the weak-defined vectors in the KSDK. See I2C RTOS for an example.

Automatic release of memory resources for an FPU task when full memory allocators are enabled.

Note that this problem is not related to Light Weight Memory configuration, which is used as a default setting for this release.

Automatic releasing of the task memory resources of a parent to a floating point can result in a memory leak under these conditions:

1. MQX RTOS is built with a full memory allocator option (MQX_USE_MEM set to 1 and MQX_ALLOCATOR_GARBAGE_COLLECTING set to 1)
2. Task A creates a floating point-enabled task B
3. Task A finishes or _task_destroy() is called referring to task A

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 MQXKSDK110RN
Revision 0, 12/2014