Document Number: MQXKSDK120RN
Rev. 0, 04/2015

MQX™ RTOS for Kinetis SDK 1.2.0 Release Notes

Contents

1 Read Me

These are the release notes for the Freescale MQX™ RTOS for Kinetis SDK.

1.1 Development Tools Requirements

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

- Kinetis Design Studio version 3.0.0
  - See build projects in `kds`subdirectories
- IAR Embedded Workbench for ARM® version 7.42
  - See build projects in `iar` subdirectories
- MDK ARM - Keil® µVision® version 5.14.0
  - MDK ARM Legacy Pack add-in - extending MDK with MQX RTOS Task Aware Debugger plugin
    - See build projects in the `uv4` subdirectories
- Atollic® TrueSTUDIO® for ARM version 5.3.0
  - See build projects in the atl subdirectories
- Cmake version 3.0 support for GCC compiler revision 4.8-2014-q3-update 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 64-bit 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 (KSDK) board-related documentation.

Table 1. Device family and evaluation boards supported

Boards	Device Family
TWR-K22F120M, FRDM-K22F	MK22F51212, MK22F25612, MK22F12812
TWR-KV31F120M	MKV31F51212, MKV31F25612, MKV31F12810
TWR-K24F120M	MK24F25612
TWR-K64F120M, FRDM-K64F	MK64F12, MK63F12, MK24F12
TWR-KV10Z32	MKV10Z7
TWR-K60D100M	MK10D10, MK20D10, MK30D10, MK40D10, MK50D10, MK51D10, MK52D10, MK53D10, MK60D10
FRDM-KL46Z	MKL34Z4, , MKL36Z4, MKL46Z4
TWR-K21D50M	MK11DA5, MK21DA5
TWR-K21F120M	MK21FA12
TWR-KW24D512, USB-KW24D512, FRDM-KW24	MKW21D5, MKW22D5, MKW24D5
TWR-KV46F150M	MKV40F15, MKV43F15, MKV44F15, MKV45F15, MKV46F15
FRDM-KL43Z, TWR-KL43Z48M	MKL17Z4, MKL27Z4, MKL33Z4, MKL43Z4
FRDM-KL27Z	MKL17Z644, MKL27Z644
TWR-K65F180M	MK66F18, MK26F18, MK65F18
FRDM-KL25Z	MKL14Z4, MKL15Z4, MKL24Z4, MKL25Z4
MRB-KW019032NA, MRB-KW019032JA, MRB-KW019032EU	MKW01Z4
FRDM-KL26Z	MKL16Z4, MKL26Z4

NOTE

These processor families are included in the KSDK but not supported by the MQX RTOS release because these processors do not meet the minimal RAM requirements: MK02F12810, MKL02Z4, MKL03Z4 and MKV30F12810

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.

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 premium support options, visit the MQX RTOS support site on www.freescale.com/mqx/support.

2 What is new

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

- MQX RTOS Kernel Components - version 5.0.2
  - Bare metal (boot) stack area is newly re-used as an RTOS Interrupt stack frame during MQX RTOS startup. This saves RAM space which is used only during application startup.
  - Init task was disabled for MQX Lite configuration, all initialization is done as a part of a user application. This feature reduces RAM footprint of MQX Lite applications.
  - The new TLSF best-fit memory allocators were added.
- RTCS TCP/IPv4 and TCP/IPv6 (optional) stack - version 4.2.0
  - LLMNR - RTCS newly support Link-Local Multicast Name Resolution (LLMNR) Server. This protocol allows resolving simple label names on local subnet without the necessity of having a DNS server.
  - Added DHCP Client IPv6 client application protocol support (Available in MQX RTOS IPv6 Add-on for purchase).
  - Added Telnet Client IPv6 protocol support. The API of this component was changed (Available in MQX RTOS IPv6 Add-on for purchase).
  - Added TFTP Client/Server IPv6 protocol support. The API of this component was changed.
  - Added support of WebSocket server as part of Http server code. The implementation was fully tested by AutoBahn test suite.
  - HTTP server was extended by SSL support (Available in WolfSSL add-on for evaluation).
  - Socket code has been updated by various BSD compatible options and flags.
  - ARP cache handling was modified to protect RTCS against DoS Attack.
  - For more details see complete change log in the <KSDK>/middleware/rtcs/rtcs_changelog.txt file.
- MFS FAT file system - version 4.2.0
  - Directory read in MFS was reworked to utilize sector cache and FAT chain abstraction.
  - Find first/next API was updated to allow extraction of long filenames using single directory chain traversal (performance improvement).
  - Path parsing was reworked to avoid allocation of path buffers (RAM footprint decreased).
  - Operations which do not create or rename directory records now support Unicode characters the filenames (UTF-8 encoding). In particular, this includes directory search and opening of existing files.
- CyaSSL evaluation package (Available in optional WolfSSL add-on for evaluation) - version 3.3.0
  - The CyaSSL library support was updated to version 3.3.0 .
  - Resolved problem with memory leak in RTCS SSL wrapper. The memory resources were incorrectly deallocated if the underlying sockets report error conditions.
- Other
  - Added examples demonstrating all MQX RTOS synchronization objects in the MQX Lite configuration - see <KSDK>/rtos/mqx/mqx/examples/

3 Release Content

MQX RTOS for KSDK 1.2.0 consists of components in the following versions:

MQX RTOS Kernel components	version: 5.0.2
RTCS TCP/IP4 stack	version: 4.2.0
RTCS TCP/IP6 stack (optional)	version: 4.2.0
MFS FAT file system	version: 4.2.0
nShell command line interpreter	version: 1.0.2
MQX RTOS Standard Library	version: 1.0.2
CyaSSL evaluation (optional)	version: 3.3.0

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>/workspace_<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/...`
RTCS source code and examples	`<install_dir>/middleware/tcpip/rtcs/...`
MFS source code and examples	`<install_dir>/middleware/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>/doc/rtos/mqx`

4 MQX RTOS Release Overview

The Freescale MQX RTOS for Kinetis SDK consists of these 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

The Freescale MQX RTOS for KSDK release contains ARM Cortex-M Platform Support only. Contact MQX RTOS support on www.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 MQX RTOS Directory Structure

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

4.5 MFS for MQX RTOS

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.

4.6 RTCS for MQX RTOS (with optional IPv6 add-in)

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 for KSDK 1.2.0 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.

Visit www.freescale.com/mqx/ipv6 for information on how to evaluate and purchase.

4.7 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.

4.8 MQX RTOS nShell

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

4.9 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.

4.10 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
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.
demo_lite	Same as the demo application, but for lite configuration.
dspi	Simple demonstration of the SPI driver for MQX RTOS.
dspi_lite	Same as the dspi application, but for lite configuration.
event	Simple demonstration of MQX RTOS events.
event_lite	Same as the event application, but for lite configuration.
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.
isr_lite	Same as the isr application, but for lite configuration.
klog	Shows kernel events being logged and later the log entries dumped on the console.
klog_lite	Same as the klog application, but for lite configuration.
log	Shows the application-specific logging feature.
log_lite	Same as the log application, but for lite configuration.
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.
lwevent_lite	Same as the lwevent application, but for lite configuration.
lwlog	Simple demonstration of MQX RTOS lightweight log feature.
lwlog_lite	Same as the lwlog application, but for lite configuration.
lwmsgq	Simple demonstration of MQX RTOS lightweight inter-process messaging.
lwmsgq_lite	Same as the lwmsgq application, but for lite configuration.
lwsem	Simple demonstration of MQX RTOS task synchronization using the lightweight semaphore object.
lwsem_lite	Same as the lwsem application, but for lite configuration.
msg	Simple demonstration of MQX RTOS inter-process message passing.
msg_lite	Same as the msg application, but for lite configuration.
mutex	Simple demonstration of MQX RTOS task synchronization using the mutex object.
mutex_lite	Same as the mutex application, but for lite configuration.
nill	Even simpler than Hello World. A void application which may be used for copy/paste to start custom application.
nill_lite	Same as the nill application, but for lite configuration.
sem	Simple demonstration of MQX RTOS task synchronization using the semaphore object.
sem_lite	Same as the sem application, but for lite configuration.
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.
taskq_lite	Same as the taskq application, but for lite configuration.
test	Shows the self-testing feature of each MQX RTOS component.

Table continues on the next page...

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

Name	Description
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.
watchdog_lite	Same as the watchdog application, but for lite configuration.

RTCS Example Applications

Table 4. <install_dir>/middleware/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.

MFS Example Applications

Table 5. <install_dir>/middleware/filesystem/mfs/examples/...

Name	Description
ramdisk	Shows use of MFS accessing the external RAM (or MRAM).
sdcard	Shows use of MFS accessing an SDCARD.
sdbench	Performance test of MFS using running on SDCARD.
usbdisk	Shows use of MFS with USB.

5 Known issues and limitations

Missing legacy pack in MDK 5.14

Missing legacy pack in MDK 5.14 block MQX library build. This could be fixed by installing of proper legacy pack from keil.com/mdk5/legacy. Installer name is MDKCM514.EXE.

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.

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 greater than or equal to 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 name of the ISR routine, installed as a native MQX RTOS interrupt, must be different than the name used in the KSDK drivers. KSDK drivers are designed to ensure that the vector table contains weak symbols of the driver ISR and the user is allowed to overload the weak vectors. This creates an issue because the MQX RTOS setup requires 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 C:/Freescale/KSDK_1.2.0_RC3/KSDK_1.2.0/examples/<board>/demo_apps/i2c_rtos example for an example implementation.

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 enabled task 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

6 Revision History

This table summarizes revisions to this document.

Revision History		
Revision number	**Date**	**Substantial changes**
0	04/2015	Kinetis SDK 1.2.0 release

How to Reach Us:

Home Page:
freescale.com

Web Support:
freescale.com/support

Document Number MQXKSDK120RN
Revision 0, 04/2015