Document Number: MQXRN
Rev. 5.0, 12/2016

NXP MQX™ RTOS 5.0 Release Notes

(Revised Dec 2016)

1 Introduction

This document is the Release Notes for the MQX™ RTOS version 5.0. The software is built based on the MQX RTOS version version 4.2. It includes the full set of RTOS services and a standard set of peripheral drivers.

NXP MQX™ RTOS is released for specific NXP Kinetis and i.MX processors. Support for other NXP processors such as Vybrid, ColdFire, and Power Architecture processor families is available upon request.

Contents

1.1 Development Tools Requirements

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

- CodeWarrior Development Studio for Microcontrollers version 10.6.4
  - Support available for Kinetis devices
  - See build projects in the cw10gcc subdirectory
  - Makefile build option (Kinetis GCC only): TOOL=cw10gcc

Embedded Access Inc

- IAR Embedded Workbench for ARM® Version 7.8
  - Support available for Kinetis devices
  - See build projects in iar subdirectories
  - Makefile build option: TOOL=iar

- DS-5 Development Studio Version 5.25
  - Support available for i.MX devices
  - See build projects in the ds5 subdirectory
  - Makefile build option: TOOL=ds5

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 NXP MQX RTOS distribution itself.

Minimum PC configuration:

- As required by Development and Build Tools

Recommended PC configuration:

- 2 GHz processor – 2 GB RAM - 2 GB free disk space

Software requirements:

- OS: Windows® 7 or later

1.3 Target Requirements

This version of NXP MQX RTOS supports the following evaluation boards. There are no special requirements for the target hardware other than what each board requires for its operation (power supply, cabling, jumper settings, etc.).

Evaluation boards supported:

- **Kinetis**
  - TWR-K60D100M Development Kit
  - TWR-K60F120M Development Kit
  - TWR-K64F120M Development Kit
  - TWR-K70F120M Development Kit

- **i.MX**
  - i.MX6 Sabre Board
  - i.MX7 Sabre Board

1.4 Set up installation instructions and technical support

Unzip the provided package to your hard drive. There is no prescribed folder to install that package to but all project files have been set for an installation to the `C:\Freescale\Freescale_MQX_5_0` directory. It is recommended to install MQX RTOS to a path without spaces to avoid build problems with certain tools.

NOTE

Since version 4.0, the pre-built libraries are not distributed in the MQX RTOS release package, which makes it necessary to compile MQX RTOS libraries for a particular board before the first use. For detailed build instructions, see the Building the MQX RTOS Libraries section in *Getting Started with Freescale MQX™ RTOS*.

For a description of available support including commercial support options, click here.

For building procedures, see *Getting Started with Freescale MQX™ RTOS* (document MQXGSRTOS).

2 What Is New

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

New Features:

2.2 Added in version 5.0

New Features:

- RTCS TCP/IP Stack now includes IPv6 protocols including:
  - DHCP Client v6 client
  - Telnet Client IPv6
  - TFTP Client/Server IPv6
- MQTT protocol has been added
- REST support has been added to HTTP Web Server
- A lightweight JSON Parser has been added
- cJSON Parser / Framer has been ported and added
- A lightweight XML Parser / Framer has been added
- zlib Compression / Decompression library has been added.

MISRA Coding Rules:
The MQX kernel code was updated to reflect the following Motor Industry Software Reliability Association (MISRA-C:2012) rules:

- Misra Rule 2.4: A project should not contain unused tag declarations
- Misra Rule 2.7: There should be no unused parameters in functions
- Misra Rule 3.1: The character sequences /*and // shall not be used within a comment
- Misra Rule 4.9: Removed use of obfuscating macros
- Misra Rule 5.2: Identifiers declared in the same scope and name space shall be distinct
- Misra Rule 5.3: An identifier declared in an inner scope shall not hide an identifier declared in an outer scope
- Misra Rule 7.2: A “u” or “U” suffix shall be applied to all integer constants that are represented in an unsigned type
- Misra Rule 7.3: The lowercase character “l” shall not be used in a literal suffix
- Misra Rule 7.4: Functions that take const pointers have now been declared as such
- Misra Rule 8.2: Function types shall be in prototype form with named parameters
- Misra Rule 8.4: A compatible declaration shall be visible when an object or function with external linkage is defined
- Misra Rule 8.7: Functions and objects should not be defined with external linkage if they are referenced in only one translation unit
- Misra Rule 8.8: The static storage class specifier shall be used in all declarations of objects and functions that have internal linkage

- Misra Rule 12.1: The precedence of operators within expressions should be made explicit
- Misra Rule 12.3: The comma operator should not be used
- Misra Rule 14.3: Controlling expressions shall not be invariant
- Misra Rule 15.4: There should be no more than one break or goto statement used to terminate any iteration statement
- Misra Rule 15.5: Many functions have been refactored to have a single point of exit at the end
- Misra Rule 15.7: All if … else if constructs shall be terminated with an else statement
- Misra Rule 17.3: A function shall not be declared implicitly
- Misra Rule 17.7: The value returned by a function having non-void return type shall be used
- Misra Rule 20.1: Modified order of includes where possible per
- Misra Rule 20.7: Literals defined in macros are now enclosed in parentheses

General Clean up and Enhancements:

- Added option to run Ethernet driver as a task.
- Added "named" lwgpio pins. Allows lwgpio pins to be read/written from the shell.32 bit constants have been suffixed with 'L'.
- Removed legacy comments from code.
- Applied consistent formatting to code.
- Code has been refactored to change (var == const) to (const == var).
- Refactored gpio_init to consistently return either MQX_OK or IO_ERROR.
- Added generic character queue component (_charq_init, _charq_enqueue, _charq_enqueue_head, _charq_dequeue, _charq_dequeue_tail, _charq_get_size, _charq_is_empty, _charq_is_not_empty, _charq_is_full, _charq_is_not_full). Serial driver now uses character queue component.
- Added *_lwevent_is_valid*() function.
- Added *_mutex_is_valid*() function.
- Added functions *_lwmsgq_size*(), *_lwmsgq_is_empty*(), *_lwmsgq_is_full*().
- Added function *_time_set_hw_reference*().
- Added *_tad_time*(), *_tad_task_summary*() and *_tad_task*() functions.
- Added function *_int_is_vector_valid*() to check if interrupt vector can be handled by MQX.
- Added functions *_mqx_get_kernel_component_handle*(), *_mqx_set_kernel_component_handle*().
- Refactored MEM driver.
- Refactored LWGPIO driver for Kinetis.
- Refactored event, interrupt, ipc, klog, log, lwmsgq, lwevent, lwsem, lwtimer components.
- Refactored PIPE driver.
- Refactored RTC driver
- Refactored TFS driver.
- 64 bit constants have been suffixed with 'LL'.
- Main now returns the result of calling *_mqx*().
- The options MQX_CHECK_FOR_ERRORS, MQX_CHECK_MEMORY_ALLOCATION_ERRORS, and MQX_CHECK_MEMORY_ALLOCATION_ERRORS have been deprecated. MQX now always checks for errors, etc.
- The option MQX_RUN_TIME_ERR_CHECK_ENABLE has been deprecated.
- The options MQX_USE_INLINE_MACROS and MQX_FORCE_USE_INLINE_MACROS have been removed.
- The option to remove the MQX counter (MQX_KD_HAS_COUNTER) has been deprecated. MQX always has a counter.
- The option to remove a task parent (MQX_TD_HAS_PARENT) has been deprecated. Tasks always have a parent.
- The option MQX_USE_32BIT_TYPES has been deprecated, since only 32-bit processors are supported.
- The option MQX_GUERRILLA_INTERRUPTS_EXIST has been removed, as it was not used.
- The option to generate a crippled evaluation has been removed.
- File system parameters are now configurable
- IOCTL codes have been unified.
- MQX_USE_IPC and MQX_IS_MULTIPROCESSOR have been combined to MQX_IS_MULTIPROCESSOR.
- Removed support for MQX Lite.
- Removed support for MQX_USE_32BIT_MESSAGE_QIDS.
- Removed mqx_assert functionality.

- Removed MQX_MONITOR_TYPEs and associated variable.
- Replaced TRUE and FALSE with true and false.
- Simplified TICK_STRUCT, tick struct always has 64 bits.
- Removed unnecessary PACKED struct directives from core mutex context structures.
- Removed all register keywords.

Bug Fixes

- *_io_get_handle*() function will now always return processor io handles when called from an ISR.
- *_io_set_handle*() function will not allow task io handles to be set when called from an ISR.
- MQX did not check result of *_lwsem*_wait() on internal semaphores. It does now.
- Resolved issues between include files for legacy and fsl can driver.
- Allow fsl can driver to support multiple devices.
- Init_task() now checks for errors and will terminate MQX if an error occurs.
- IOdebug driver did not release resources on failed install
- IOdebug driver held semaphore on failed open.
- IOdebug driver did not check return value of *_lwsem_wait*().
- Many ENET functions did not check input parameters for validity.
- ENET driver did not check return value of *_lwsem_wait*().
- MEM driver waits until no other tasks are accessing the mem driver before it is removed.
- Fixed bug in kinetis interrupt driver.
- Many drivers did not release resources if driver failed to install

3 Release Content

Table 1 lists the contents of this release:

Table 1. Release Contents

Deliverable	Location
Configuration Files and Mass-Build Projects	`<install_dir>/config/...`
Configuration and mass-build project for all supported boards	`.../config/<board>`
MQX PSP, BSP Source Code, and Examples	`<install_dir>/mqx/...`
MQX PSP source code for Kinetis/Vybrid ARM Cortex-M core	`.../mqx/source/psp/cortex_m`
MQX PSP source code for Vybrid ARM Cortex-A core	`.../mqx/source/psp/cortex_a`
MQX PSP build projects	`.../mqx/build/<compiler>/psp_<board>`
MQX BSP source code	`.../mqx/source/bsp/<board>`
MQX BSP build projects	`.../mqx/build/<compiler>/bsp_<board>`
RTCS source code and examples	`<install_dir>/rtcs/...`
RTCS source code	`.../rtcs/source`
RTCS build projects	`.../rtcs/build/<compiler>/rtcs_<board>`
RTCS example applications	`.../rtcs/examples`
MFS source code and examples	`<install_dir>/mfs/...`
MFS source code	`.../mfs/source`
MFS build projects	`.../mfs/build/<compiler>/mfs_<board>`
MFS example applications	`.../mfs/examples`

Table continues on the next page…

Table 1. Release Contents (continued)

USB Host driver source code and examples	<install_dir>/usb/host/...
USB Host source code and class drivers	.../usb/host/source
USB Host build projects	.../usb/host/build/<compiler>/
USB Host example applications (HID, MSD, HUB)	.../usb/host/examples
USB Device drivers source code and examples	<install_dir>/usb/device/...
USB Device source code	.../usb/device/source
USB Device build projects	.../usb/device/build/<compiler>/
USB Device example applications (HID, MSD, CDC, PHDC)	.../usb_v2/device/examples
USBv2 Host and Device driver code and examples	<install_dir>/usb_v2/...
USBv2 Source code	.../usb_v2/usb core
USBv2 example applications	.../usb_v2/example
Shell Library Source Code	<install_dir>/shell/...
Shell source code	.../shell/source
Shell build projects	.../shell/build/<compiler>/ shell_<board>
Build tools plug-ins	<CodeWarrior_dir>/...
MFS on FFS example	ffs/examples/mfs_nandflash
IAR Task Aware Debugging plugin (TAD)	.../tools/iar_extensions/
PC Host tools	<install_dir>/tools
BSP cloning wizard	…/tools/BSPCloningWizard/ BSPCloningWizard.exe
TFS Make Utility	…/tools/mktfs.exe
Check for Latest Version tool	…/tools/webchk.exe
AWK interpreter (GNU General Public License)	…/tools/gawk.exe
SNMP code generation scripts	…/tools/snmp/*.awk
Timing HTML report tool (for mqx/examples/benchmrk/timing)	…/tools/timing.exe
Code size HTML report tool (for mqx/examples/benchmrk/codesize)	…/tools/codesize.exe
TAD string and configuration files	…/tools/tad
Demo Applications	<install_dir>/demo
Various demo applications demonstrating complex MQX RTOS functionalities.	…/demo/...
Documentation	<install_dir>/doc
User Guides and Reference Manuals for MQX RTOS, RTCS, MFS, I/O Drivers, USB etc.	…/doc

<compiler> can be IAR, CodeWarrior GCC or Kinetis Design Studio GCC

This figure shows the NXP MQX RTOS directories installed to the user's host computer (subdirectories not shown for clarity):

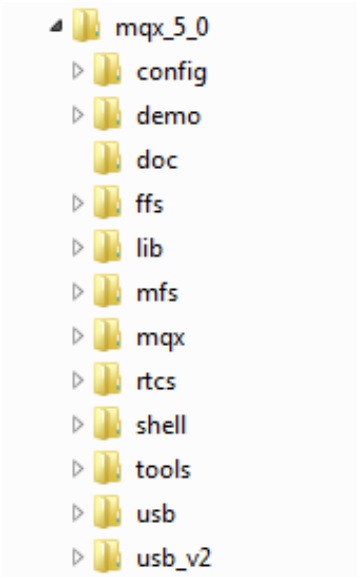

Figure 1. NXP MQX RTOS Directories

4 MQX RTOS Release Overview

The NXP MQX RTOS is intended for specific Kinetis and i.MX processors. The release consists of the following libraries:

- MQX RTOS real time kernel and system components
- TCP/IP networking stack (RTCS)
- FAT file system (MFS)
- Shell
- NAND flash file system (FFS)
- USB Host and Device stacks
- Platform and Board support packages
- I/O drivers

This release contains the following components and I/O drivers, however drivers will only be supported for processors that have the corresponding circuitry.

- UART Serial driver (polled and interrupt driven version)
- I2C driver (polled and interrupt driven version)
- SPI driver
- LWGPIO – light weight GPIO
- Audio driver I2S or SAI
- QuadSPI Driver
- FlashX Flash diver
- NAND Flash diver
- ESDHC driver
- Compact Flash Card driver
- SD Card driver (SPI or SDHC based)
- RTC / IRTC Real Time Clock driver
- TSS Touch Sensing driver

- DCU driver
- FlexCAN, msCAN
- Ethernet Driver

4.1 MQX RTOS PSP

This release of NXP MQX RTOS contains support for specific ARM Cortex-M Platform Support Packages. Contact mqxsales@nxp.com for ports to other NXP platforms from the Kinetis, i.MX, ColdFire, Power Architecture, Qorivva, and QorIQ processor families.

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

4.2 MQX RTOS BSPs

NXP MQX RTOS release includes Board Support Packages (BSPs) for the boards mentioned in Target Requirements.

The board-specific code from `/mqx/source/bsp/<board>` is built together with I/O driver files from `/mqx/source/io`. These two parts form a static library generally referred as a BSP. The functions included in this library enable the board and operating system to boot up and use the I/O driver functions.

The following section describes drivers supported by the MQX BSPs.

4.3 I/O drivers supported

The following list describes I/O drivers available in the latest MQX RTOS release. The drivers are an optional part of the MQX RTOS and their installation can be enabled or disabled in the BSP startup code. To provide the optimal code and RAM application size, most of the drivers are disabled by default in the `/config/<board>/user_config.h` file. The drivers required by demonstration applications (in the /demo folder) are enabled by default.

See *MQX™ RTOS I/O Drivers User's Guide* (document MQXIOUG) for details.

NOTE

When BSPCFG_ driver-enabling macros are missing in the `/config/<board>/user_config.h` file, the default setting is taken from the BSP-specific header file located in the `/mqx/source/bsp/<board>/<board>.h`. The user decides whether to enable the automatic installation of the driver in the BSP startup code (by enabling the appropriate BSPCFG_ENABLE_XXX macro in the `user_config.h`), or manually in the application code.

TFS – Trivial Filesystem

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.

I2C I/O Driver

This driver supports the I2C interface in both master and slave mode. If enabled in user configuration, the I2C driver is installed during the BSP startup code as the "i2cx" in polled mode and as the "ii2cx" in interrupt mode where "x" stands for a specified I2C channel number. Example applications are provided in the MQX RTOS source tree for both master and slave mode.

I2S and SAI I/O Driver

This driver supports an I2S interface in a master mode. If enabled in user configuration, the I2S device driver is installed during the BSP startup code as “i2s0:”. An example application is provided in the MQX RTOS source tree.

SPI I/O Driver

This driver supports the operation master mode. If enabled in user configuration, the SPI device drivers are installed during the BSP startup code as “spi0:” (or “spiX:” where X is index of the SPI module used). The SPI driver was significantly rewritten in MQX RTOS 4.0, so that there is no distinct interrupt or polled driver type. See *MQX™ RTOS I/O Drivers User's Guide* (document MQXIOUG) for details. On Kinetis platforms, the driver uses DMA to function.

QuadSPI I/O Driver

This driver provides a C language API to the QuadSPI peripheral module. If enabled in user configuration, the QuadSPI device drivers are installed during BSP startup code as "qspi0:" (or "qspiX:" where X is index of QSPI module used). See *MQX™ RTOS I/O Drivers User's Guide* (document MQXIOUG) for details.

FlexCAN Driver

This driver provides a C language API to the FlexCAN peripheral module. An example application is provided in the MQX RTOS source tree.

msCAN Driver

This driver provides a C language API to the msCAN peripheral module. An example application is provided in the MQX RTOS source tree.

RTC Driver

This driver provides a C language API to the Real Time Clock peripheral module and functions, and synchronizes the clock time between RTC and MQX RTOS systems. If enabled in user configuration, the RTC module is initialized and MQX RTOS time is renewed automatically during BSP startup.

Serial I/O Driver

The standard SCI (UART) driver supports both polled and interrupt-driven modes. If enabled in user configuration, the serial devices are installed as “ttya:”, “ttyb:” and “ttyc:” (polled mode) and “ittya:”, “ittyb:” and “ittyc:” (interrupt mode) automatically during BSP startup.

LWGPIO I/O Driver

This the light weight GPIO driver which provides a C language API to all GPIO ports available on a particular device.

ADC Driver (obsolete)

This I/O driver provides a uniform interface to ADC channels. This driver has been replaced by LWADC I/O driver.

LWADC I/O Driver

This driver provides a C language API to ensure a uniform access to ADC peripheral basic features.

Flash I/O Driver

This I/O driver provides a standard interface to either internal or external Flash memory. If enabled in user configuration, the Flash driver (called FlashX) is installed as “flashx:” device automatically by the BSP startup code. Note that “flash0”, “flash1” etc. device names are used for FlashX devices installed for external Flash memory. For devices with internal Flash memory, the FlashX driver depends on several parameters passed in a form of global symbols from an application or from a Linker Command File. For more information, see driver installation code in the BSP and an example application provided in the MQX RTOS source tree.

ENET Driver

The low-level Ethernet driver is used by the RTCS TCP/IP software stack. The driver is initialized directly by the application before RTCS is used for the first time. The RTCS Shell and HTTP examples demonstrate the use of this driver.

PCCard I/O Driver

This I/O driver provides a low-level access to the PCCard functionality by using Flexbus and CPLD circuit. The CPLD code can be found in the `<install_dir>/mqx/source/io/pccard/<card_name>`. If enabled in the user configuration, the PCCard device driver is installed as "pccarda:" automatically during the BSP startup.

PCFlash I/O Driver

The Compact Flash Card I/O driver is installed on top of the PCCard low-level driver and enables standard disk drive operations. The MFS file system can be installed on top of this device. If enabled in user configuration, the PCFlash device driver is installed as "pcflasha:" automatically during the BSP startup.

SD Card I/O Driver

This I/O driver implements a subset of the SD protocol v2.0 (SDHC). The driver can use either the MQX RTOS SPI driver or the MQX RTOS (e)SDHC driver to communicate with the SD Card device. Install the driver at the application level, and pass a lower-layer driver handle to it. The MFS file system can be installed on top of this device.

(E)SDHC I/O Driver

This I/O driver covers the (e)SDHC peripheral module and provides low-level communication interface for various types of cards including SD, SDHC, SDIO, SDCOMBO, SDHCCOMBO, MMC, and CE-ATA.

Resistive Touch-Screen Driver
This I/O driver accesses the ADC and GPIO modules to detect touch events and acquire touch coordinates on a resistive touch-screen unit.

I/ODebug Driver

This driver redirects I/O functions, such as printf, to a debug probe-based communication channel. The CodeWarrior 10, IAR EWARM, or Keil µVision debugger consoles are supported. See *Getting Started with Freescale MQX™ RTOS* for details about the setup and use of this feature.

HWTimer Driver

This driver provides a C language API for uniform access to the features of various HW timer modules such as PIT and SysTick.

DMA Driver

This driver provides the C language API and essential functionality to control the DMA peripheral module.

I/O Expander Driver

This driver controls an off-chip I/O expander device and provides a convenient interface for individual pin handling. Currently, it only supports the MAX7310 device.

4.4 Default I/O Channel

An I/O communication device installed by MQX BSP can be used as the standard I/O channel. See *Getting Started with Freescale MQX™ RTOS* for the default console setting for each supported development board.

4.5 MQX RTOS PSP and BSP Directory Structure

RTOS files are located in the mqx subdirectory of the NXP MQX RTOS installation. The directory structure is shown in this image.

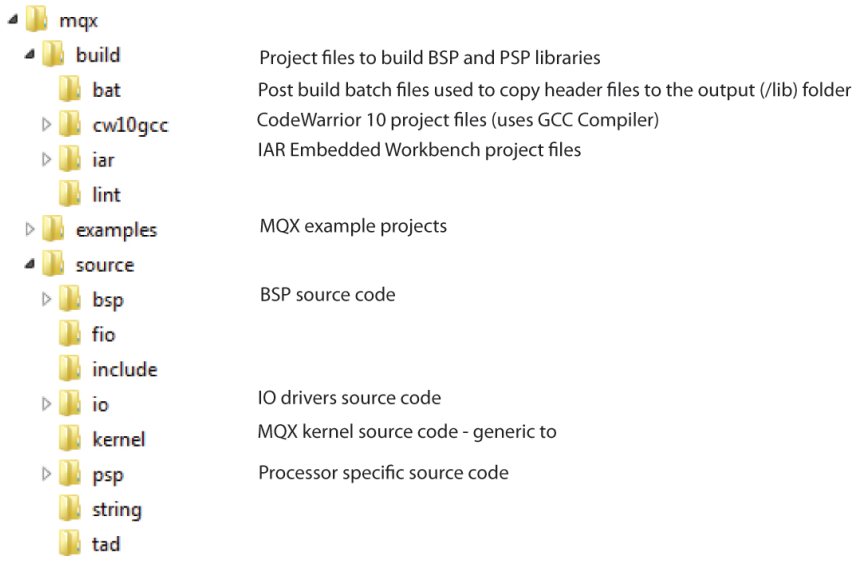

Figure 2. MQX PSP and BSP Directory Structure

4.6 MFS for MQX RTOS

MFS files from the `/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 RTCS for MQX RTOS (with optional IPv6 add-in)

RTCS files from the `/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 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. The IPv6 protocols for RTCS were previously separately licensed for a fee. However, starting with MQXv5 they are included with the MQX package.

4.8 USB Host for MQX RTOS

NXP MQX RTOS release includes the USB Host drivers and USB class drivers. The USB HDK (Host Development Kit) files from the `/usb/host/source` directory are built into a static library. When linked to the user application, the USB HDK library enables the application to communicate with various USB devices attached on the USB bus.

The HDK contains the following USB class drivers:

- USB Hub class used to attach multiple devices to a single host port. If enabled at the application level, the HUB support is fully transparent. Only the user application needs to be modified to handle multiple USB devices simultaneously. A Keyboard/Mouse example application is provided.
- Human-interface Class (HID) used to access mouse, keyboard, and similar devices.
- Mass storage device (MSD) Class used to access USB drives.
- Communication Device Class (CDC) used as a serial communication device implementing virtual “tty” ports.
- Audio Class.
- Basic Printer Class.

4.9 USB Device for MQX RTOS

NXP MQX RTOS release includes the USB Device drivers and example applications implementing various USB devices. The USB DDK (Device Development Kit) files from the `/usb/device/source` directory are built into a static library. When linked to the user application, the USB DDK library enables the application to act as a USB device supporting one or more of the following classes:

- HID (mouse functionality demonstrated)
- MSD (internal RAM area accessed as mass storage device)
- CDC COM (virtual serial line implementation)
- CDC NIC (virtual network interface card implementation)
- PHDC (medical applications)
- Audio

4.10 MQX RTOS Shell

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

4.11 Changing the MQX RTOS source files

The NXP MQX RTOS is distributed in source code form. Do not modify any of the source files other than the compile- time configuration files. This recommendation applies to all files under “source” and “build” sub-directories in all MQX RTOS, RTCS, MFS, USB, and other core components folders.

If you are creating custom board support packages or adding additional I/O drivers, add the new files and subdirectories to the following directories:

```
<install_dir>/mqx/source/bsp
<install_dir>/mqx/source/io
```

4.12 Building the MQX RTOS libraries

For more details about building MQX RTOS libraries and applications, see *Getting Started with Freescale MQX™ RTOS*.

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

4.13 Example applications

Demo applications are in this directory:

```
<install_dir>/demo
```

The examples are written to demonstrate the most frequently used features of the NXP 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.

Table 2. MQX Examples

Name	Description
benchmrk	Contains benchmarks codes for timing and code size for different components.
bootloader	Contains basic functions for boot loader application.
can/flexcan	Shows usage of FlexCAN API functions to transmit and receive CAN frames.
cplus	Shows simple C++ application.
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.
flashx	Demonstration of FlashX driver functionality.
flashx_swap	A demonstration of FlashX driver's swap and reset functionality.
hello	A trivial Hello World application using a single task.
hello2	A trivial Hello World application spread across two tasks.
hwtimer	Shows usage of HW timer driver abstraction. Demonstrates how to initialize HW timer for various modules, set frequency, callback, start, and stop the timer.
i2c	Shows how to read/write data from/to external EEPROM. Additional HW setup is needed.
i2s_demo	Demonstrates use of audio I2S driver. TWR-AUDIO card is needed to run this example.
io	Demonstrates use of an alternate UART port as a console output.
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.
lowpower	Shows how to switch between several predefined low-power operation-modes.
lwadc	Shows usage of the ADC driver, sampling analog values from the two ADC channels.
lwdemo	Same as the "demo" application, but implemented using lightweight components only.
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.
qspi	Demonstrates basic operation of QuadSPI driver, interfacing to QSPI flash.
rtc	Shows the Real Time Clock module API. Demonstrates how to synchronize RTC and MQX RTOS time and how to use RTC alarm interrupts.
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.
test	Shows the self-testing feature of each MQX RTOS component.

Table continues on the next page…

tfs	Shows the usage of ROM-based Trivial File System in an MQX RTOS application.
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.

Table 3. 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.

Table 4. MFS Examples

Name	Description
cfcard	Console shell-based example showing the MFS filesystem used with and CFCard storage.
mfs_ftp	RTCS FTP demo accessing the MFS filesystem mounted on the USB mass storage. For an FTP example without the USB functionality, see the RTCS Shell demo.
mfs_usb	Console shell-based example showing how to access MFS filesystem mounted on the USB mass storage.
ramdisk	Shows use of MFS accessing the external RAM (or MRAM).
sdcard	Shows use of MFS accessing the SDHC or SPI-connect SD Card.

Table 5. USB v2 Host Examples

Name	Description
audio/microphone	Enables connecting a USB microphone and record the sound to SD Card (wav formatl).
audio/speaker	Enables connecting a USB speaker and play the sound from SD Card (wav format).
cdc/cdc_serial	This example demonstrates the virtual serial port capability with abstract control model. Redirects the communication from CDC device, which is connected to the board, to the serial port.
hid/keyboard	This application echoes keys pressed on the USB keyboard onto the serial console.
hid/mouse	Displays USB mouse events on the serial console.
hid/keyboard+mouse	Keyboard and mouse demos combined in a single application.
msd/msd_cmd	Executes the standard "mass storage device" commands to the USB disk and shows the response on the serial console (see MFS examples for USB filesystem access).
msd/msd_fatfs	Console shell-based example showing how to access an MFS filesystem mounted on the u-disk memory.
phdc/11073Manager_Demo	This application demonstrates basic host personal healthcare class functionality.

Table 6. USB v2 Device Examples

Name	Description
audio/generator	Acts as a USB microphone, playing out a short audio loop.
audio/speaker	Receives audio stream data from the host (PC) and plays it out through the I2S driver.
cdc/virtual_com	Implements a virtual serial line loopback.cdc/virtual_nicImplements a virtual network interface cards.
cdc/virtual_nic	Implements a virtual network interface cards.
composite/hid_audio	Shows basic functionality of composite device using hid and audio classes.
composite/msd_cdc	Shows basic functionality of composite device using mass storage and CDC classes.
hid/mouse	Creates a virtual mouse which keeps moving in a square loop, 100 pixels in size.
hid/hid_keyboard	Creates a virtual keyboard which can scroll the screen up and down.
msd	Implements small storage device in internal RAM memory.
phdc/weighscale	Implements personal healthcare device.
msd/disk	Implements small storage device in internal RAM memory.

Table 7. Demo Examples

Name	Description
hvac	Simple implementation of console-based HVAC with optional USB logging and FTP access.
web_hvac	HVAC demo with the HTTP server implementing the GUI. Ajax-based pages demonstrating the advanced use of the HTTP server.
web_hvac_v2	HVAC demo with the HTTP server implementing the GUI. Ajax-based pages demonstrating the advanced use of the HTTP server with new usb_v2 stack.

5 Known Issues and Limitations

There are no new issues with MQXv5, but the following issues and limitations from MQX 4.2 still apply.

USB Host HUB Examples

HUB class support is enabled in HID example applications. The applications run correctly with the USB device attached either directly or through the hub. However, the example code only handles a single device. A combined Mouse+Keyboard demo handles one mouse and one keyboard simultaneously. The same kind of multiple devices, which are attached through the hub, cannot be used in the example applications.

Supporting “Hot Device Uninstall” in MQX I/O Subsystem

In the current implementation of the MQX I/O subsystem, the application is responsible for dealing with application tasks which have opened file handles while uninstalling a device driver. A typical demonstration of the problem is USB mass storage handling: When a USB attach event is detected, an application installs the MFS partition manager and MFS file system "device" on top of the USB driver. The application runs tasks, such as shell, which open and access files provided by the MFS filesystem device. When the user unplugs the USB mass storage device, the application has a limited way to detect an opened file before uninstalling the MFS filesystem device. The file I/O functions start reporting errors when accessing the device after it is physically detached. Design the application code so that the tasks close all files affected by the detach event before the MFS filesystem driver can be uninstalled. If there is an attempt to close the MFS handle prior to closing all related

files, a sharing violation error is returned. An example application "mfs_usb" demonstrates how to close files by retrying the closing operation of the MFS handle. If a task keeps one or more files open for an extensive time period, use a suitable method to notify it about the ongoing filesystem un-installation. This implementation may add additional application overhead. Work is ongoing on the MQX I/O subsystem to ensure that file operations safely return error states even after the underlying device driver is uninstalled. This enhancement will simplify the application code error recovery.

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.

USB EHCI and KHCI Stack Buffer Restrictions

Align the buffers used by KHCI at 4B. Align the buffers used by EHCI at cache line and the size to cache line boundary in the cached area. If the goal is to optimize performance, allocate the buffer used by EHCI in the un-cached memory space.

ARP Entries Issue

When the board is put into a busy Ethernet environment with many ARP requests, the ARP entries cause memory fragmentation, which leads to RTCSERR_TCPIP_NO_BUFFS when connect() is called.

FlexCan Driver Issues

Several issues are identified during the development of the FlexCAN driver: On TWR-K70F120M board, the TX/RX signals are not routed to the elevator by default and the FlexCAN example does not work. To enable the FlexCAN operation, solder the zero-ohm resistors, R22 and R23, on TWR-K70F120M board. The 10-kbit baudrate doesn't generally work. FlexCAN detects bit0 errors in its own transmitted messages.

Android USB MSD Cannot Be Interfaced

If certain types of Android phones are connected to the system, the attach event is not generated. The issue is currently investigated and will be fixed in a future MQX RTOS version.

User Mode Functionality in CW10 GCC

The User Mode functionality is not supported in the GCC compiler.

MFS Does Not Check Validity of Directory Rename

MFS_Rename_file() function does not check the necessary precondition when renaming a directory. If the directory is renamed to its own subdirectory, the directory becomes inaccessible and lost cluster chains are created.

EHCI HUB functionality limitation

The HUB functionality of the EHCI is not fully supported (dynamically attach/detach is not supported).

UTF8 support in MFS

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

DSPI issues related to the DMA usage

When the DSPI uses the eDMA, it may transfer data incorrectly or fail when eDMA is used for another purpose. If the DSPI driver is the only user of eDMA, it should operate correctly. This behavior is a result of the silicon design of the DSPI. DMA usage can be disabled in the DSPI driver by redefining the macro BSPCFG_DSPIx_USE_DMA to 0 in user_config.h.

USB Host CDC function works with limited input.

In the USB Host CDC example, no more than 200 characters are allowed in one line when inputting to the terminal and no more than 160 characters in one line when transferring files. Otherwise, USB Host CDC example may malfunction.

DSPI FIFO length setting

The DSPI driver FIFO length is currently defined as a constant value (DSPI_FIFO_DEPTH = 4) and it is used for all SPI modules on chip. However there exist devices such as K64f and K21F where the FIFO length is different for some SPI modules. Driver currently does support various length per a module. As a workaround it is suggested to decrease FIFO depth to 1 in case channels with smaller FIFO length are used.

How to Reach Us:

Home Page:
nxp.com

MQXv5 Web Page:
nxp.com/MQXv5

Web Support:
nxp.com/support

Document Number MQXRN
Revision 5.0, 12/2016