Freescale Semiconductor
Application Note

AN4190
Rev. 0, 08/2010

Supporting New Toolchains with Freescale MQX RTOS

by: Michal Princ and Michal Hanak
Freescale MSG Software Team
Roznov

1 Introduction

This application note serves as an MQX™ RTOS porting guide for tool vendors. It provides instructions on how to get the MQX supported by alternating toolchains.

This document focuses on Freescale MQX versions released for the Freescale ColdFire® processor family.

Contents

freescale™ semiconductor

2 Freescale MQX RTOS

2.1 What is MQX RTOS

The MQX is a real-time operating system that has been designed for uniprocessor, multiprocessor, and distributed-processor embedded real-time systems. It deals with a runtime library of functions that programs use to become real-time multitasking applications. The main features are its scalable size, component-oriented architecture, and easy use.

The Freescale MQX RTOS offers leading edge software technology for embedded designs based on Freescale processors and MCUs. The Freescale MQX RTOS offers a straightforward application programming interface (API) with a modular component based architecture that makes it simple to fine-tune custom applications. It also allows developers to add Web servers, e-mail, network management, security, and routing to their designs. Components are linked in only if needed, preventing unused functions from bloating the memory footprint. By leveraging Freescale's strong network of partners, Freescale MQX software solutions easily scale across third-party software and tools such as security, industrial protocols, and graphical plug-ins.

For more information, refer to the user guide titled *Freescale MQX Real-Time Operating System User's Guide* (document MQXUG) or the reference manual, *Freescale MQX Real-Time Operating System Reference Manual*, (document MQXRM) or visit www.freescale.com/mqx.

2.2 Directory Structure of the MQX RTOS

One single MQX release contains the MQX operating system plus all the other software components supported for a given microprocessor part (TCP/IP stack, USB host and device stack, file system, and more). Figure 1 shows Freescale MQX RTOS directories installed to the user host computer (subdirectories reduced for clarity).

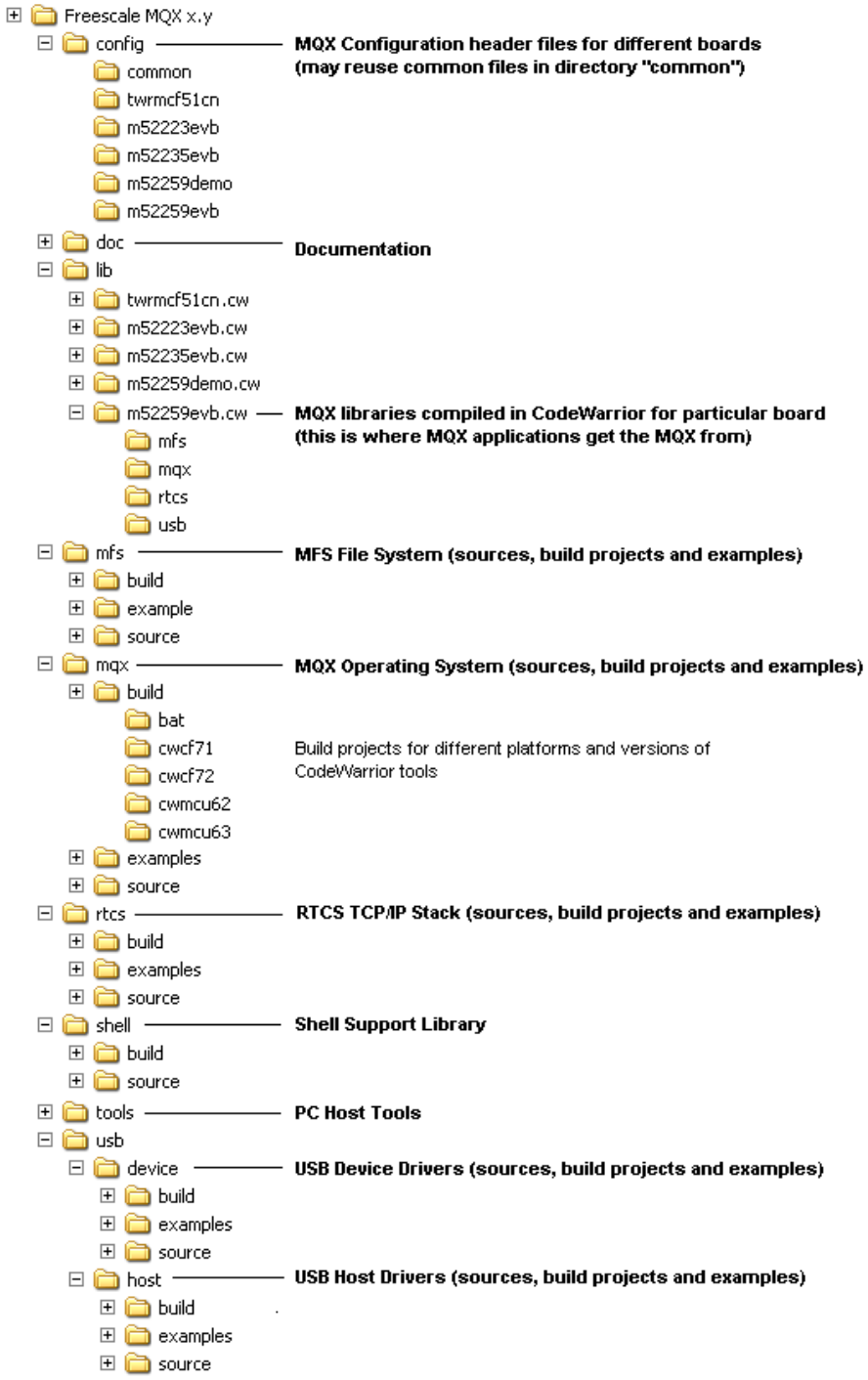

Figure 1. Directory structure of Freescale MQX RTOS

2.3 MQX PSP and BSP Directory Structure

Figure 2 shows the directory structure of the MQX RTOS component located in the top-level **mqx** directory in more detail.

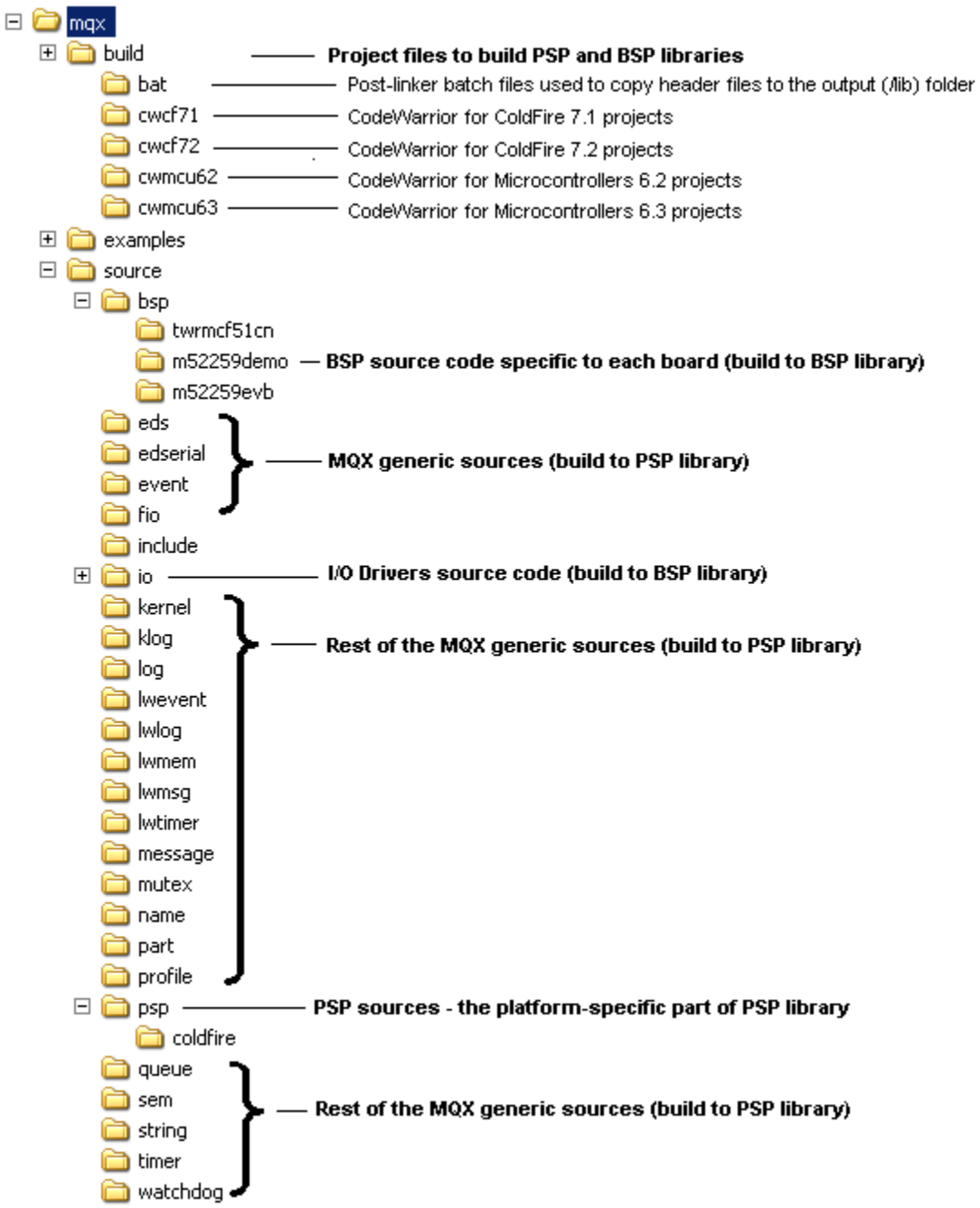

Figure 2. MQX RTOS directory structure

2.3.1 BSP Subdirectories

The subdirectories in **mqx\source\bsp** typically follow the name of the board. Most of the BSP code is located in the **mqx\source\bsp\<board>** subdirectory and among others it contains processor and board

initialization code, and data structures used to initialize various I/O drivers in a way suitable for a given board. The toolchain-specific portion of the BSP as low-level startup code, memory initialization code, and debugger configuration files is located in the **mqx\source\bsp\<board>\<compiler>** subfolder.

2.3.2 PSP Subdirectories

The **mqx\source\psp** directory contains the PSP library platform-dependent code. For example, the ColdFire subdirectory contains the MQX kernel parts, specific to the Freescale ColdFire architecture (core initialization, register save/restore code for interrupt handling, stack handling, cache control functions, and so on). In addition, this directory also contains processor definition files for each supported processor derivative.

2.3.3 I/O Subdirectories

Subdirectories in the **mqx\source\io** contain source code for MQX I/O drivers. Typically, source files in each I/O driver directory are further split to device-specific and device-independent. The I/O drivers suitable for a board are part of the BSP build project and are compiled into the BSP library.

2.3.4 Other Source Subdirectories

All other directories in the source contain generic parts of the MQX RTOS. Together with the platform-dependent PSP code, the generic sources are compiled into the PSP library.

2.4 Freescale MQX Build Projects

All necessary build projects are located in the **mqx\build\<compiler>** directory. For example, CodeWarrior for ColdFire v. 7.2 (cwcf72) project files can be found in the **mqx\build\cwcf72** directory.

For each board, there are two build projects available, PSP and BSP, see Figure 3. The BSP project contains board-specific code, while PSP is platform-specific (for example ColdFire) only. The PSP project does not contain any board-specific code. Despite this, both projects refer to the board name in their file names, and both also generate the binary output file into the same board-specific directory **lib\<board>.<compiler>**.

The board-independent PSP library is also compiled to board-specific output directory because the compile-time configuration file is taken from board-specific directory **config\<board>**. In other words, if the PSP source code itself does not depend on the board features, the user may want to build a different PSP for different boards.

2.4.1 PSP Build Project

The PSP project is used to build the PSP library, which contains the platform-dependent parts from **mqx\source\psp** and also contains generic MQX RTOS code.

2.4.2 BSP Build Project

The BSP project is used to build the BSP library, which contains the board-specific code from **mqx\source\bsp\<board>** and also the selected I/O drivers from the **mqx\source\io** directory.

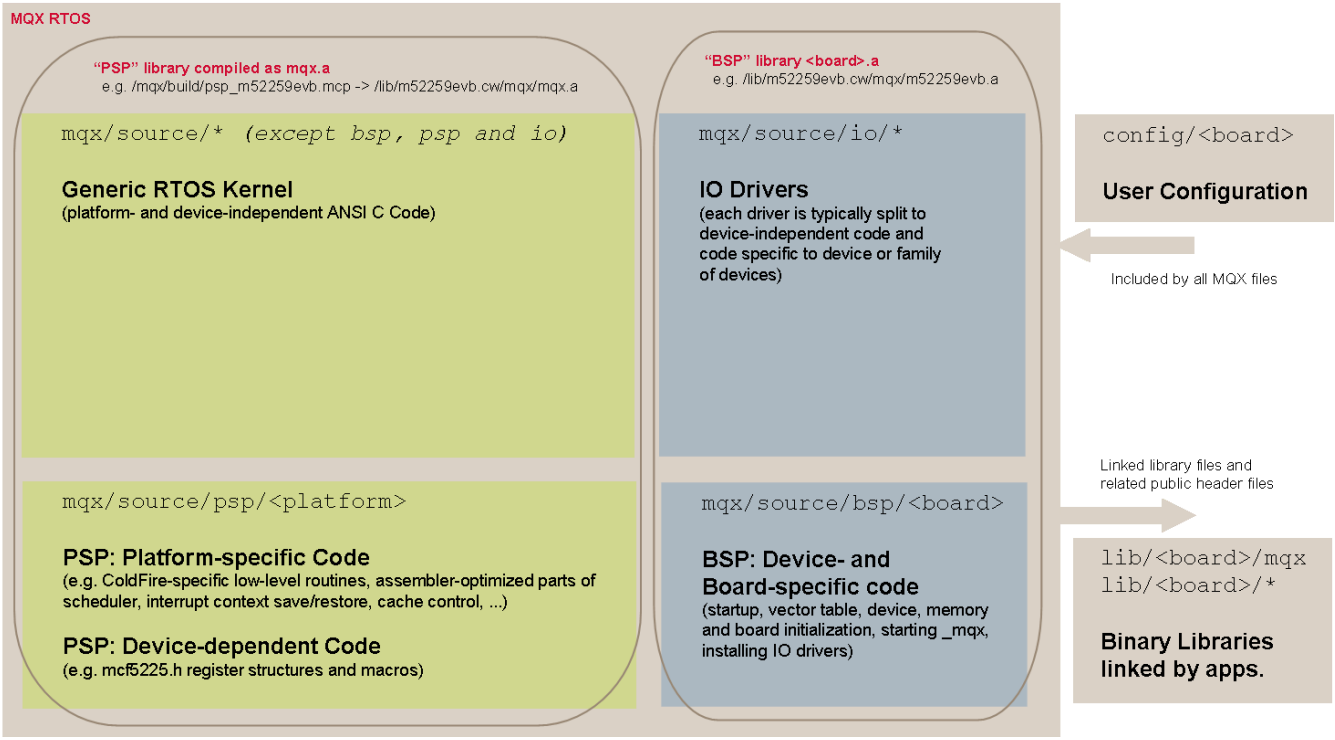

Figure 3. PSP and BSP code of the MQX RTOS

2.4.3 Post-Build Processing

All build projects are configured to generate the resulting binary library file in the top-level **lib\<board>.<compiler>** directory. For example, the CodeWarrior (cw) libraries for the M52259EVB board are built into the **lib\m52259evb.cw** directory.

Both BSP and PSP build projects are also set-up to execute post-build batch file, which copies all the public header files to the destination **lib** directory. This makes the output **/lib** folder the only place accessed by the MQX application code. The MQX application build projects do not need to make any reference to the MQX RTOS source tree.

2.5 Freescale CodeWarrior as a Default Toolchain for MQX

All Freescale MQX RTOS releases were compiled and tested with the default development tool — Freescale CodeWarrior. The following table summarizes all CodeWarrior-specific files and folders that can be found in the MQX installation.

Table 1. Overview of CodeWarrior-specific parts of the MQX installation

CW-related parts of the MQX installation	Location	Examples of CW for ColdFire v.7.2 / M52259EVB
Mass-build projects of all the libraries (where applicable)	config\<board>*cwxxxx*	config\m52259evb\cwcf72\build_m52259evb_libs.mcp
BSP and PSP build projects	mqx\build\cwxxxx	mqx\build\cwcf72\bsp_m52259evb.mcp mqx\build\cwcf72\psp_m52259evb.mcp
CW-related BSP source code (start-up code, memory initialization code, and debugger configuration files)	mqx\source\bsp\<board>\cw	*mqx**source**bsp*\m52259evb*cw* files: boot.c cw.c extmram.lcf intflash.lcf intram.lcf dbg\m52259evb.cfg dbg\m52259evb.mem
CW-related PSP source code	mqx\source\psp\<platform>	mqx\source\psp\coldfire files: cw_comp.h
All demo and example build projects	<application folder>\cwxxxx	mqx\examples\hello\cwcf72\hello_m52259evb.mcp
Binary output files	lib\<board>.cw	lib\m52259evb.cw

2.5.1 Build Targets

Each CodeWarrior project in Freescale MQX™ RTOS contains multiple compiler and linker configurations (the so called build „targets“).

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

- **Debug** target — 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. All output libraries (or executables) have **_d** postfix in the file name (for example: **lib\m52259evb.cw\mqx\mqx_d.a**).
- **Release** target — The compiler optimizations are set to maximum. The compiled code is hard to debug and must be used for final applications only. There is no postfix in the output file name (for example: **lib\m52259evb.cw\mqx\mqx.a**).

Similarly, different types of build targets exist for different application binary interface (ABI) settings. In MQX version 3.6, the StdABI targets are no-longer available. They are not supported by the latest versions of the CodeWarrior tools.

- **StdABI** — The compiler is set to generate function calls with standard parameter (on-stack) passing interface. This is the only available option in the MQX versions prior 3.4.
- **RegABI** — Starting in MQX release 3.4, the register parameter passing is enabled and is set as default in all example and stationery projects. All output libraries have the **_regabi** postfix in the file name (for example, **lib\m52259evb.cw\mqx\mqx_regabi.a**). The register parameter passing option gives generally smaller and faster code.

Considering the naming convention stated above, the following build targets exist in the MQX library build projects:

- **Debug RegABI** — Register parameter passing ABI, easy to debug code
- **Release RegABI** — Register parameter passing ABI, optimization set to maximum
- **Build all** — This target contains all other targets for convenient mass-build

3 Instructions for Porting the MQX to a New Toolchain

This chapter provides general instructions on how to port the MQX RTOS to an alternative toolchain, that is; what needs to be added, changed, or modified to build MQX libraries by any toolchain.

3.1 Folders Dedicated to a New Toolchain

The same way as the CW-specific parts of the MQX are separated in specific folders, the dedicated directories have to be created for a new toolchain. In consistence with all rules defined in Section 2, “Freescale MQX RTOS“ the following new toolchain-related folders have to be created:

- config\<board>\<compiler>
- mqx\build\<compiler>
- mqx\source\bsp\<board>\<compiler>
- lib\<board>.<compiler>
- <any application folder>\<compiler>

3.2 MQX Source Code Changes

Although most of the MQX code is written in C-language, using different compilers should not cause any difficulties. There is also some MQX code written in the assembler code that needs to be changed to follow different assembler syntax. Except for this, the toolchain-specific code as start-up code located in **mqx\source\bsp\<board>\<compiler>** subdirectory must also be changed. All these necessary changes are discussed in the following subchapters.

3.2.1 Changes Required to Compile Assembler Source Files

The following Table 2 summarizes all the MQX assembler source files, their location, and description.

Table 2. MQX Assembler source files

Assembler Source File	Description
mqx\source\psp\<architecture>\dispatch.s	This assembler file contains functions for task scheduling.
mqx\source\psp\<architecture>\ipsum.s	This assembler file contains the implementation for a one's complement checksum.
mqx\source\psp\<architecture>\psp_prv.s	This assembler header file contains private declarations and macros for use with the mqx assembler files.
mqx\source\psp\<architecture>\types.s	This assembler file contains the assembler offsets calculated by the program KRNL_OFF.C. These offsets are to be included in any assembler program that wishes to access kernel data structures.

The MQX assembler files include **.h** files necessary for build. A special preprocessor macro __ASM__ is always defined when a C header file is included from any assembler file. This macro is used to skip C-language syntactical statements during assembler compilation. The header files that are included by MQX assembler source files are listed in Table 3.

Table 3. Header files included by MQX assembler source files

Header File	Description
mqx\source\include\asm_mac.h	This header file is included by all assembler files. It contains macros used by the assembler and resolves differences between different assembler directive syntax (constants, prefixes). The assembler toolchain should be configured in a way that all MQX assembler files are pre-processed by C preprocessor. This is how macros from the asm_mac.h file gets defined and used.
mqx\source\psp\<architecture>\psp.h	This generic header file is included by the dispatch.s. Using the __ASM__ macro just two header files are included by the psp.h: **mqx_cnfg.h** — For the configuration of various optional MQX features. Individual configuration defines can be overridden in the "user_config.h" file. **psp_cpu.h** — For including <processor.h> headers with the CPU-specific information (macros like PSP_HAS_CODE_CACHE and so on.).

To avoid redundancy, the goal must be to reuse existing **.s** files as much as possible. Only create separate **.s** files with the same code (but a different syntax) when absolutely necessary. Try to resolve all tool incompatibilities in the **asm_mac.h** file.

One of the problematic parts to achieve this goal is that different compilers require to mark comments in assembler source files differently. While CodeWarrior uses semicolon “;” other compilers can use different characters, for example, “/* */”, “//” or “;*”. So far the ”;*” sequence seems to be the most portable way to identify single-line comments across different assembler tools.

3.2.2 Changes Required to Compile BSP

The **mqx\source\bsp\<board>\<compiler>** subdirectory contains a toolchain-specific portion of the BSP as:

- Low-level startup code (boot code)
- Memory initialization code
- Debugger configuration files

— Linker command files

All these files must be either modified or replaced based on the used toolchain needs.

3.3 Makefiles and Build Projects

While CodeWarrior uses build projects and there is no need to create makefiles with source file listings and dependency definitions, many other development toolchains do not have this possibility and writing own makefiles is required. The user must know what the makefile structure and the syntax is.

The CodeWarrior tool also gives the possibility to generate the GNU makefile from the build project: open the CW project, go to the menu -> Project -> Export Project as GNU Makefile. You can create the makefile that can serve as a basis for the consecutive tuning.

As described in Section 2.5.1, “Build Targets“ two different ways of parameter passing are supported by the CodeWarrior compiler. While standard parameter (on-stack) passing is considered as obsolete and is no longer supported by the latest CodeWarrior tool versions, register parameter passing is more effective and preferred. However, not all compilers support both ways of parameter passing, this fact has to be considered when defining targets in makefiles.

All makefiles or build projects must be configured to generate the resulting binary library files in the **lib\<board>.<compiler>** directory. Also, paths in post-build batch files must be updated to copy all the public header files to the correct destination **lib** directory.

A detailed description of the makefile structure is out of the scope of this document. For more information see the documentation of the selected toolchain and compiler.

4 MQX Task-Aware Debugging

Task-Aware Debugging (TAD) enables more advanced and user-friendly debugging of MQX-based applications. There are several proprietary solutions: TAD Plug-in for the CodeWarrior Development Studio, for the CW10 (Eclipse based), and for the IAR toolchain.

These plug-ins allow different TAD screens to be opened during the debugging session. The most helpful and frequently used screens are shown in Figure 4.

- **Task Summary** — Overview of all tasks created in the MQX application.
- **Stack Usage Summary** — Displays information of interrupt and task stacks. Typically, stack overflow is a root cause of the vast majority of problems in the MQX user applications.
- **Memory Block Summary** — Displays address, size, and type information about each memory block allocated in the default memory pool by the MQX system or applications. Additional memory pools (if used) may be displayed using the **Memory Pools** screen.

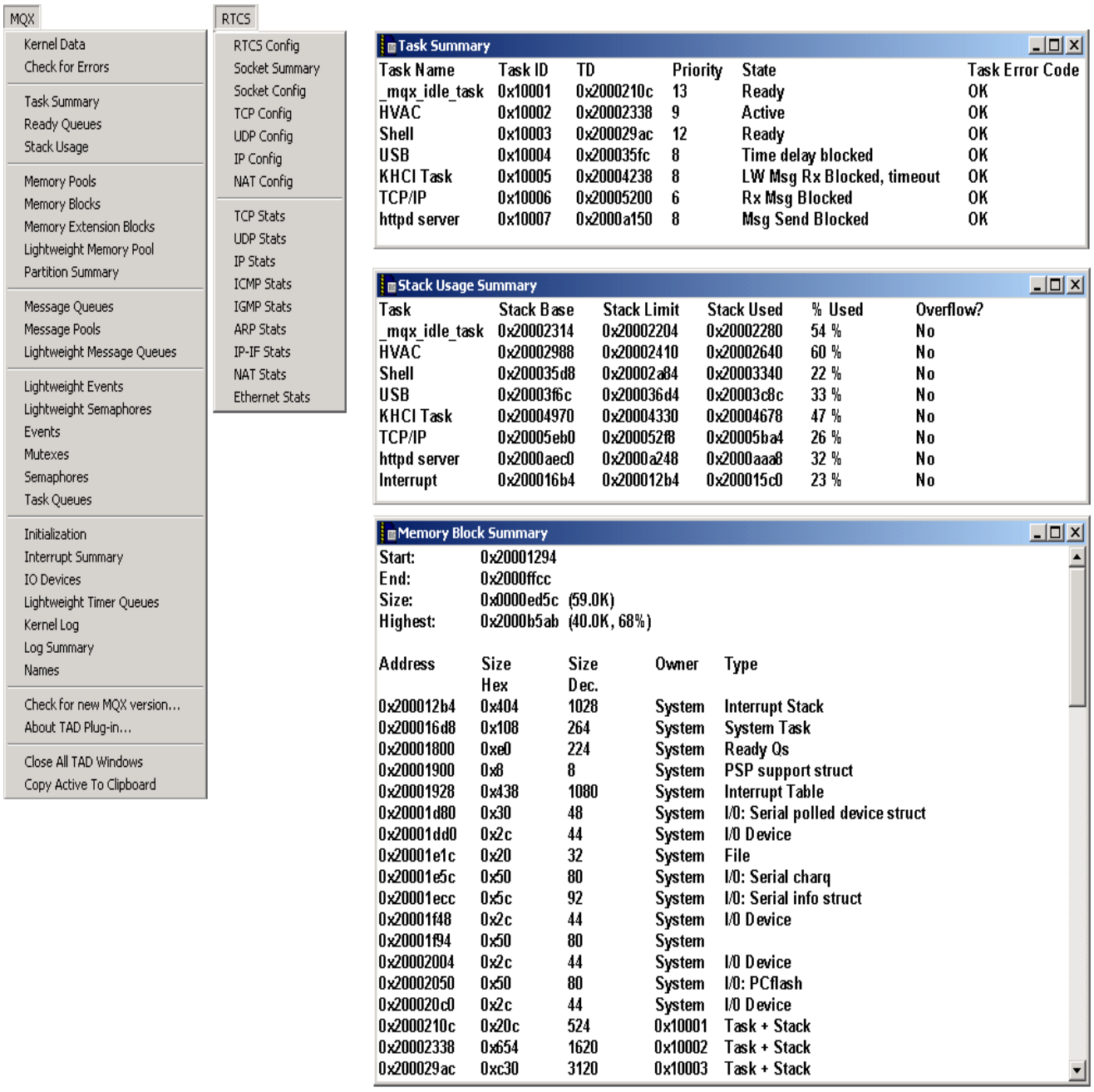

Figure 4. MQX Task-aware debugging screens

TAD also provides native debugger support for the multi-tasking MQX environment. When an application is stopped at breakpoint or by pressing the **Break** button, the name of the active task is displayed in the drop-down list in the debugger window. You also have a chance to use the drop-down list to see the current execution point of any other task.

For more information about TAD, see MQX release notes.

5 Conclusion

Porting the MQX RTOS to new toolchains is discussed in this document.

For more information about Freescale products and updates go to www.freescale.com.

THIS PAGE IS INTENTIONALLY BLANK

How to Reach Us:

Home Page:
www.freescale.com

Web Support:
http://www.freescale.com/support

USA/Europe or Locations Not Listed:
Freescale Semiconductor, Inc.
Technical Information Center, EL516
2100 East Elliot Road
Tempe, Arizona 85284
+1-800-521-6274 or +1-480-768-2130
www.freescale.com/support

Europe, Middle East, and Africa:
Freescale Halbleiter Deutschland GmbH
Technical Information Center
Schatzbogen 7
81829 Muenchen, Germany
+44 1296 380 456 (English)
+46 8 52200080 (English)
+49 89 92103 559 (German)
+33 1 69 35 48 48 (French)
www.freescale.com/support

Japan:
Freescale Semiconductor Japan Ltd.
Headquarters
ARCO Tower 15F
1-8-1, Shimo-Meguro, Meguro-ku,
Tokyo 153-0064
Japan
0120 191014 or +81 3 5437 9125
support.japan@freescale.com

Asia/Pacific:
Freescale Semiconductor Hong Kong Ltd.
Technical Information Center
2 Dai King Street
Tai Po Industrial Estate
Tai Po, N.T., Hong Kong
+800 2666 8080
support.asia@freescale.com

For Literature Requests Only:
Freescale Semiconductor Literature Distribution Center
P.O. Box 5405
Denver, Colorado 80217
1-800-441-2447 or 303-675-2140
Fax: 303-675-2150
LDCForFreescaleSemiconductor@hibbertgroup.com

Document Number: AN4190
Rev. 0
08/2010

freescale™
semiconductor