Writing Your First MQX™ RTOS Application

by:
Freescale Semiconductor

1 Introduction

To develop MQX™ RTOS applications for the first time, it is necessary to understand how the environment works and how to generate new applications for the environment.

The purpose of this Application Note is to provide:

- Information that enables developers to start developing their first application on Freescale MQX RTOS quickly and easily.
- Basic understanding how to develop Freescale MQX applications.

For CodeWarrior, the following boards are used as an example:

- TWR-MCF51JF board for microcontrollers.
- M52259DEMO board for ColdFire.
- TWR-K60D100M board for Kinetis.

- TWR-VF65GS10 board for Vybrid.

2 Freescale MQX RTOS install and setup

Freescale MQX RTOS is a set of source code, example code, CodeWarrior plug-ins and other toolchains, and software tools. It uses an installer to load all the features for the supported platforms for fast and easy installation.

2.1 Download Freescale MQX RTOS

To download the MQX installer:

1. Visit freescale.com and register with Freescale to get the required access.
2. Go to freescale.com/mqx, login, and download the MQX installer.

2.1.1 Download Freescale MQX RTOS for Vybrid in a Linux Host

Download the MQX Linux installer from the Timesys website linuxlink.timesys.com.

Note: Need to install only if you plan to build MQX libraries for Vybrid on a Linux host machine, using DS-5 toolchain. If you build MQX libraries on Windows, you don't need to download this.

2.2 Install Freescale MQX RTOS

To install Freescale MQX RTOS:

1. Execute the installation wizard. The default settings install Freescale MQX at the path: C:\Freescale\Freescale_MQX_4_x
2. See *Freescale MQX™ RTOS Release Notes* (document MQXRN). This document details the changes made in the current release, minimal requirements to install, the boards and drivers supported, known issues and a list of all examples and demos that MQX software contains.

2.2.1 Set up Freescale MQX RTOS for Vybrid in a Linux Host

1. Make the script executable:
   ```
   chmod +x MQX-<version>-linux.sh
   ```
2. Run the script
   ```
   ./MQX-<version>-linux.sh
   ```
3. Select a different default installation directory such as `/home/<login>/mqx`
4. You now have a full MQX installation in your Linux environment.

2.3 Set up Freescale MQX RTOS

Freescale MQX RTOS has a directory structure. Detailed information about this structure can be found in the FSL MQX Release Notes. The developer can change board support package (BSP) and platform support package (PSP) settings using the *user_config.h* file. Adding and removing flags in this file enables the developer to add or remove components, change settings for the different drivers, and add or remove logging settings. The *user_config.h* file can be found in the path: *<install_dir>/config/<board>/user_config.h*.

After a modification is done in the user_config.h file, it is necessary to recompile the libraries. The documentation is available in MQX RTOS version 4.x release in the following locations:

- *C:\Freescale\Freescale_MQX_4_x\doc\tools\ds5\MQX-DS5-Getting-Started.pdf*

- *C:\Freescale\Freescale_MQX_4_x\doc\tools\iar\MQX-IAR-Getting-Started.pdf*

- *C:\Freescale\Freescale_MQX_4_x\doc\tools\uv4\MQX-uVision4-Getting-Started.pdf*

- *C:\Freescale\Freescale_MQX_4_x\doc\tools\cw\FSL_MQX_in_CW_10_x.pdf*

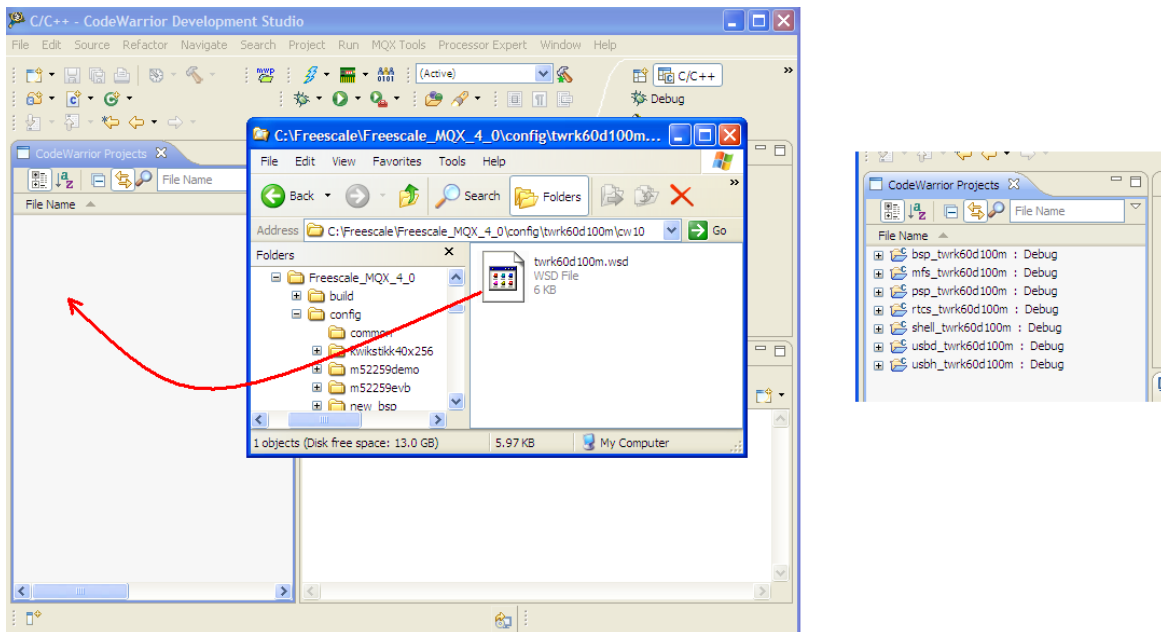All libraries are loaded to the enviroment automatically. After that, it is necessary to build the libraries.

Figure 1. Open Libraries

2.3.1   Set up Freescale MQX RTOS for Vybrid

For IAR, open the *build_libs.eww* found at *<install_dir>\build\<board_name>\iar*
For DS-5, open the *<board_name>.wsd* found at *<install_dir>\build\<board_name>\ds5*

Libraries are saved in the folder *<install_dir>/lib/<board>*. If you try to alter anything inside the *<install_dir>/lib/* folder, the changes will not take effect since this is the output folder after the libraries have been compiled.

3 Freescale MQX stationery projects

Freescale MQX RTOS version 4.0.1 is supported on:

- CodeWarrior Development Studio for Microcontrollers Version 10.3
- IAR Embedded Workbench® for ARM® Version 6.50.2
- ARM Development Studio 5 (DS-5)™ version 5.13
- ARM MDK™ - Keil µVision® Version 4.60

For more information about the development tools supported, see Release Notes. Note that the newer MQX releases support different tools.

3.1 TWR-MCF51JF_MQX_RTOS project in CodeWarrior Development Studio for Microcontrollers 10.5

Follow the steps below to create and configure a Freescale MQX stationery project for the TWR-MCF51JF board:

1. Open CodeWarrior Development Studio for Microcontrollers Version 10.3.
2. Go to 'File' menu and click on File -> New ->MQX 4.0 Project
3. Enter a 'Project name,' select a 'Location,' and click on the 'OK' button. For this example, the following values are set in the dialog window:
    - Project name: TWRMCF51JF_MQX_RTOS

    - Location: <path_location>\workspace\TWRMCF51JF_MQX_RTOS
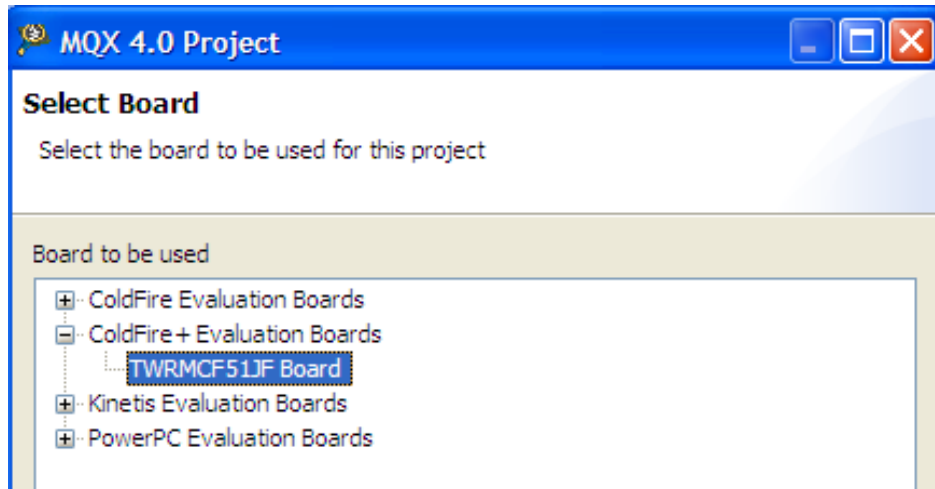4. Expand the selected demo or evaluation board. For this example, TWRMCF51JF board is selected.

Figure 2. TWRMCF51JF board

5. Select the libraries that are used. For this example, select 'MQX Only.' Figure 3 shows the creation of the stationery project.
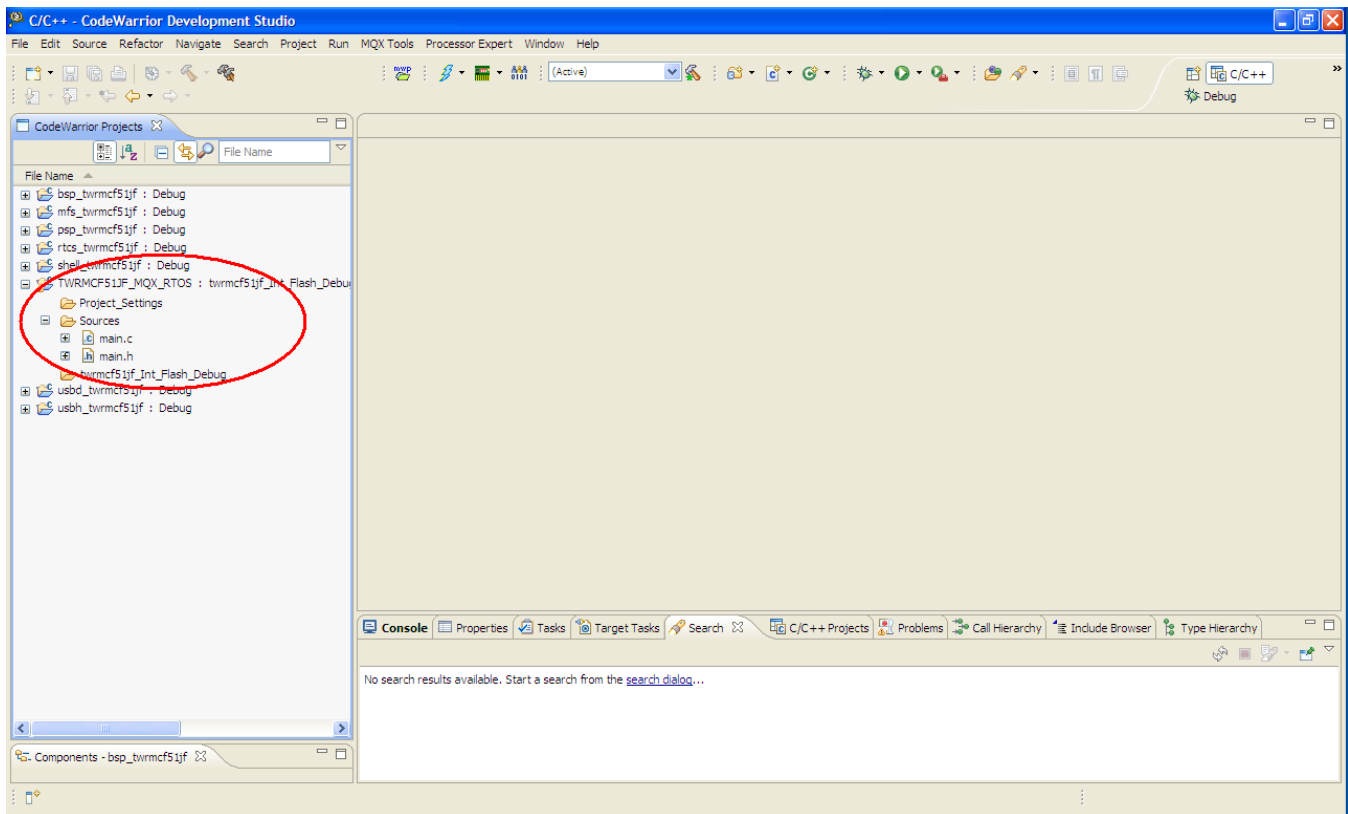

Figure 3. Freescale MQX Stationery Project for TWRMCF51JF Board.

Writing Your First MQX™ RTOS Application, Rev. 3, 02/2014

NOTE

See *Getting Started with Freescale MQX™ RTOS* (document MQXGSRTOS) to configure jumpers and other switches for each board used.

3.2   M52259DEMO_MQX_RTOS project in CodeWarrior Development Studio for Microcontrollers 10.3

Follow the steps below to create and configure a Freescale MQX stationery project for the M52259DEMO board:

1. Open CodeWarrior Development Studio for Microcontrollers Version 10.3.
2. Go to 'File' menu and click on File -> New ->MQX 4.0 Project
3. Enter a 'Project name,' select a 'Location,' and click on the 'Ok' button. For this example, the following values are set in the dialog window:
    — Project name: M52259DEMO _MQX_RTOS
    — Location: <location>\workspace\M52259DEMO _MQX_RTOS
4. Expand the selected demo or evaluation board. For this example, M52259DEMO board is selected.
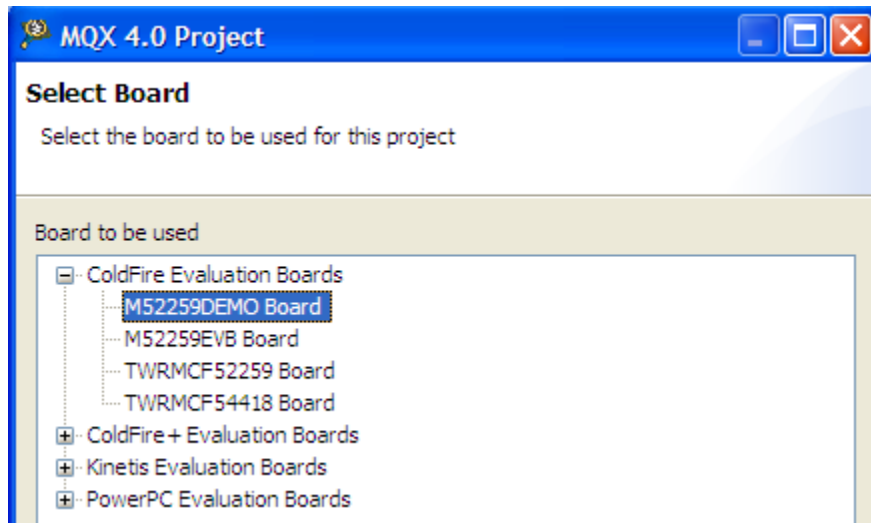

Figure 4. M52259DEMO board.

5. Select the libraries that are used. For this example, select 'MQX Only.'
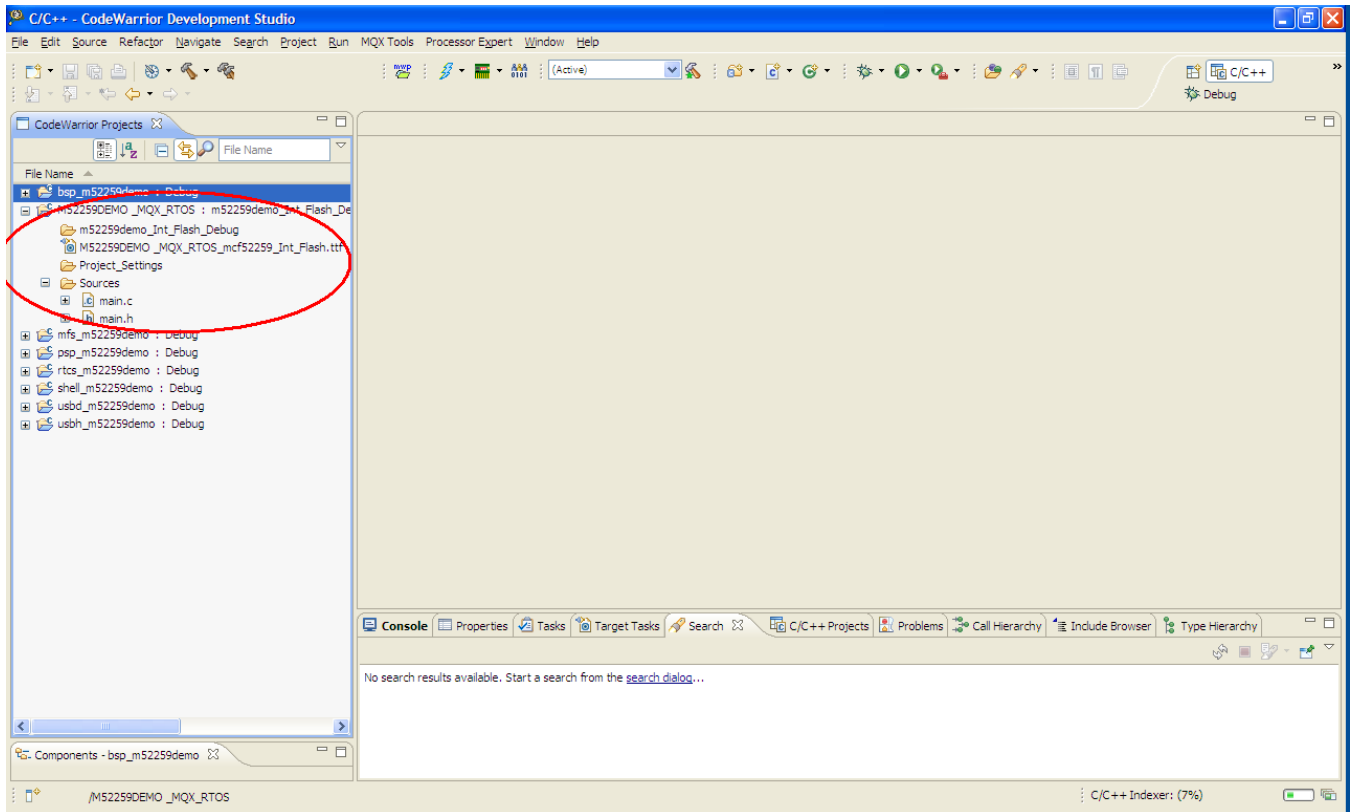Figure 5 shows the creation of the stationery project.

Figure 5. Freescale MQX Stationery Project for M52259DEMO Board.

NOTE

See the *Getting Started with Freescale MQX™ RTOS* (document MQXGSRTOS) to configure jumpers and other switches for each board used.

3.3   TWR-K60D100M_MQX_RTOS project in CodeWarrior Development Studio for Microcontrollers 10.3

Follow these steps to create and configure a Freescale MQX stationery project for the TWR-K60D100M board:

1. Open CodeWarrior Development Studio for Microcontrollers Version 10.3.
2. Go to 'File' menu and click on File -> New ->MQX 4.0 Project
3. Enter a 'Project name,' select a 'Location,' and click on the 'Ok' button. For this example, the following values are set in the dialog window:
   — Project name: TWR-K60D100M_MQX_RTOS
   — Location: <location>\workspace\TWR-K60D100M_MQX_RTOS
4. Expand the selected demo or evaluation board. For this example, TWR-K60D100M board is selected.
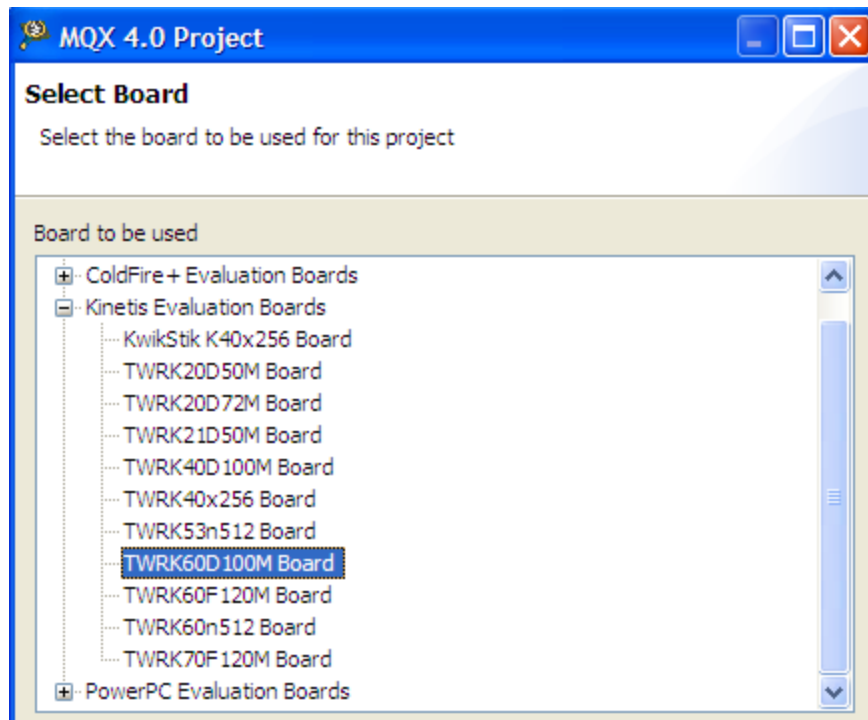
Figure 6. TWR-K60D100M board

5. Select the libraries that are used. For this example, select 'MQX Only.'
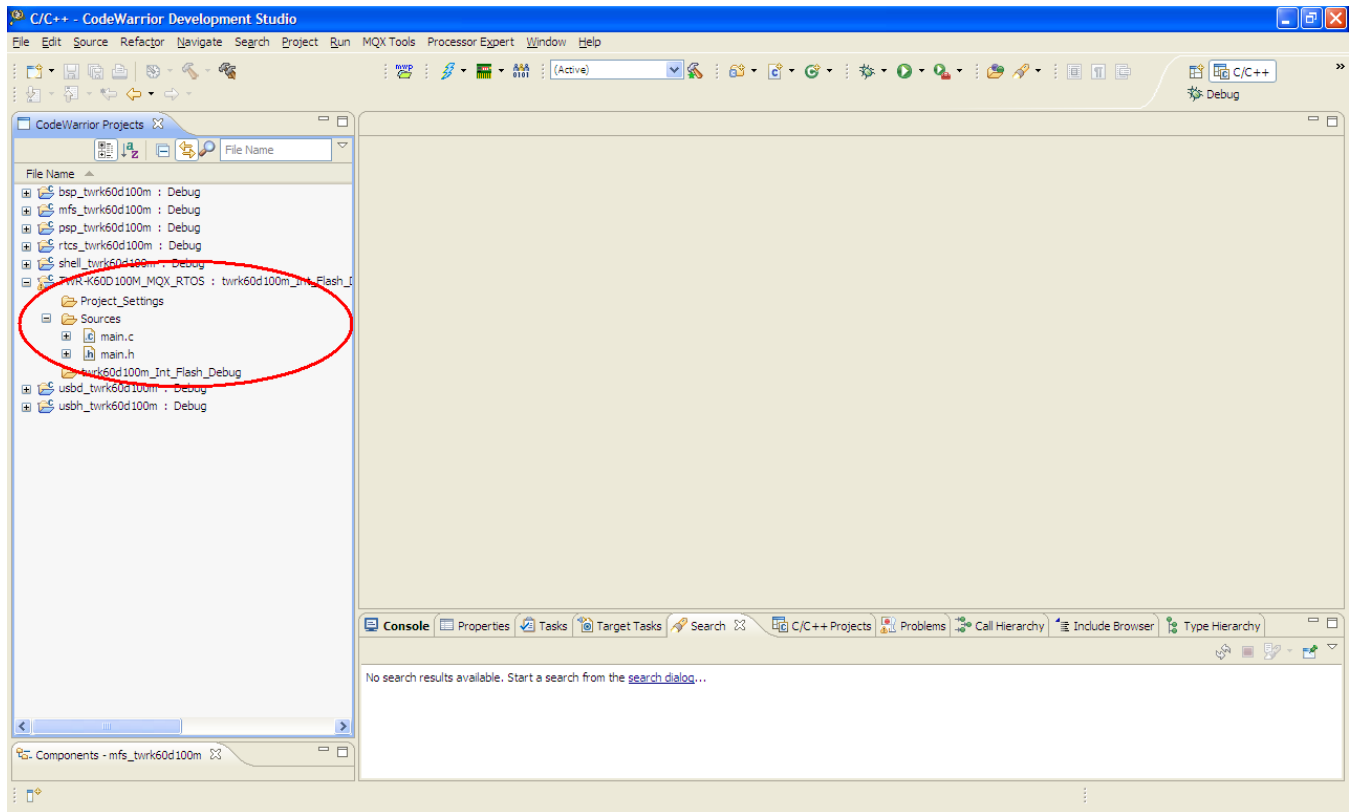Figure 7 shows the creation of the stationery project.

Figure 7. Freescale MQX Stationery Project for TWR-K60D100M Board.

NOTE

See *Getting Started with Freescale MQX™* RTOS (document MQXGSRTOS) to configure jumpers and other switches for each board used.

3.4 Freescale MQX stationery projects for Vybrid in IAR

IAR lacks a wizard to create an MQX stationary project. Therefore, the best procedure to create a new MQX project in IAR is to copy an existing project to a desired folder.
The compiler, assembler, and linker paths need to be changed. The below steps explain how to create a new project:

1. Copy the most similar example of the MQX installation folder in the desired folder:
   In this example the hello example placed at *\Freescale_MQX_4_0\mqx\examples\hello\iar* is copied to a new folder named Applications.
   folder is located at: *Freescale_MQX_4_0\Applications*
Copy the *.c* file found in the example project too. The structure of the files should look like the figure 8.
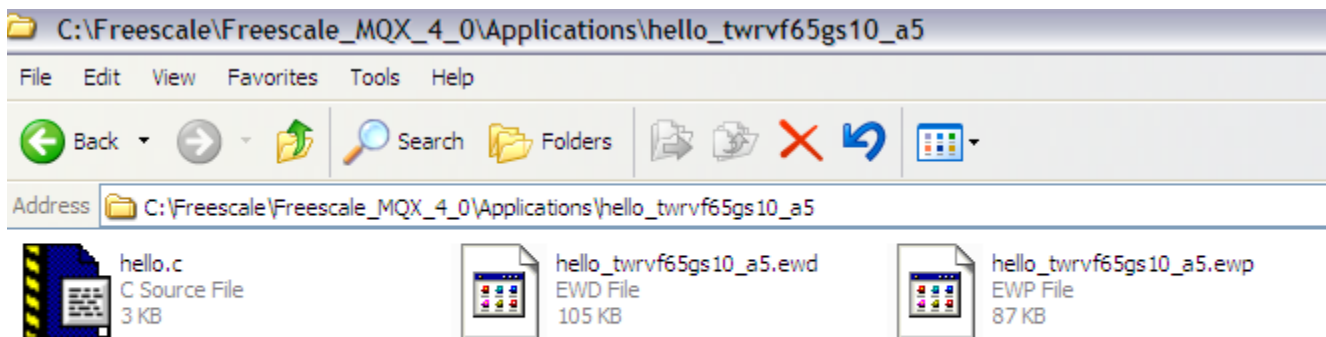

Figure 8. Freescale MQX Stationery Project layout for Vybrid in IAR.

2. Open the IAR Embedded Workbench and the project you copied into the Applications folder. Add to the project the *.c*, *.h* and *.a* files. The *.a* files are found in *Freescale_MQX_4_0\lib\twrvf65gs10_a5.iar\debug\bsp* and *Freescale_MQX_4_0\lib\twrvf65gs10_a5.iar\debug\psp* folder for the debug target. For the release targets, the folders are: *Freescale_MQX_4_0\lib\twrvf65gs10_a5.iar\release\bsp* and *Freescale_MQX_4_0\lib\twrvf65gs10_a5.iar\release\psp*

You can drag and drop the *.c* and *.h* files into the workspace. Then add the files as shown in the figure below.
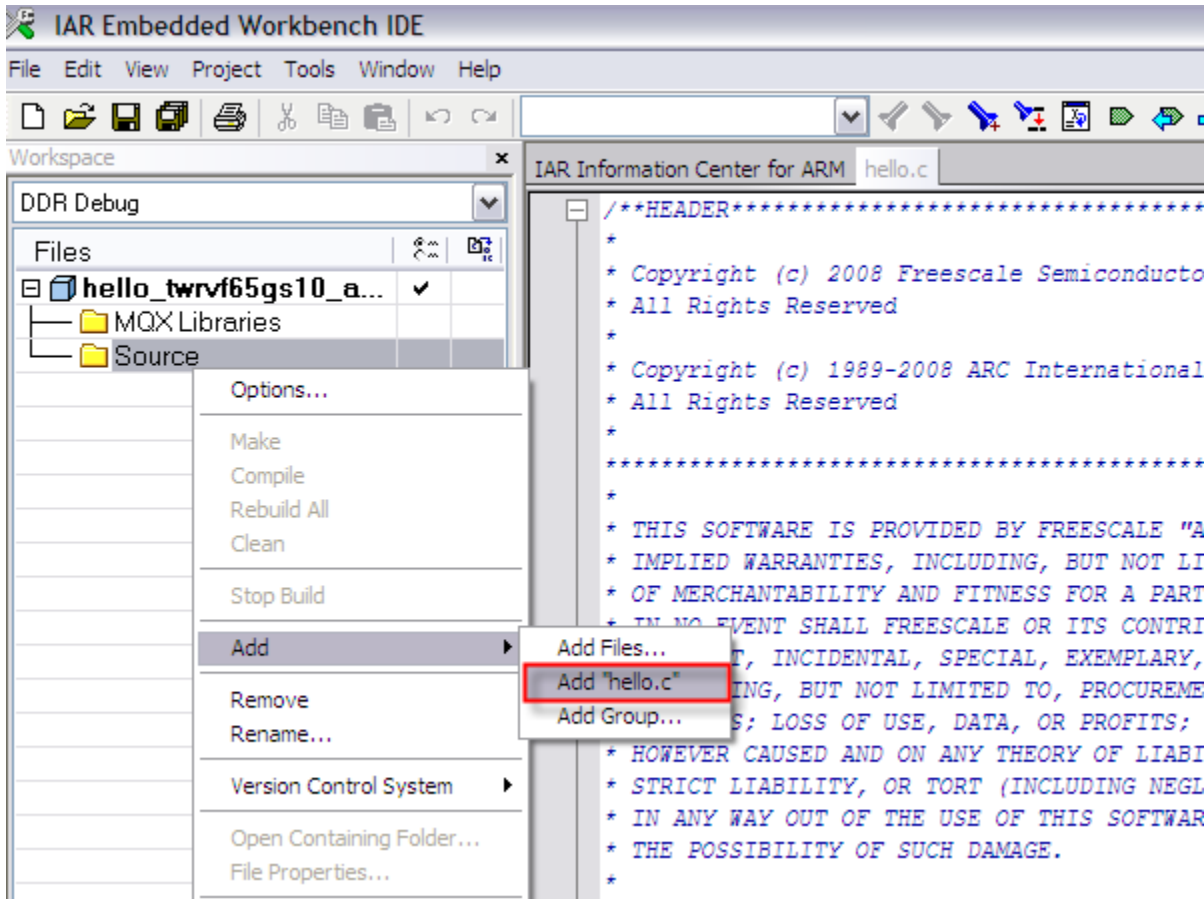
Figure 9. Adding files to MQX IAR project.

3. Under project options change the compiler, assembler, source, and linker paths for your project. Replace the original paths of the assembler and compiler with:

```
$PROJ_DIR$/../../lib/twrvf65gs10_a5.iar/debug/psp
$PROJ_DIR$/../../lib/twrvf65gs10_a5.iar/debug/bsp/Sources
$PROJ_DIR$/../../lib/twrvf65gs10_a5.iar/debug/bsp/Generated_Code
$PROJ_DIR$/../../lib/twrvf65gs10_a5.iar/debug/bsp
$PROJ_DIR$/../../lib/twrvf65gs10_a5.iar/debug
```
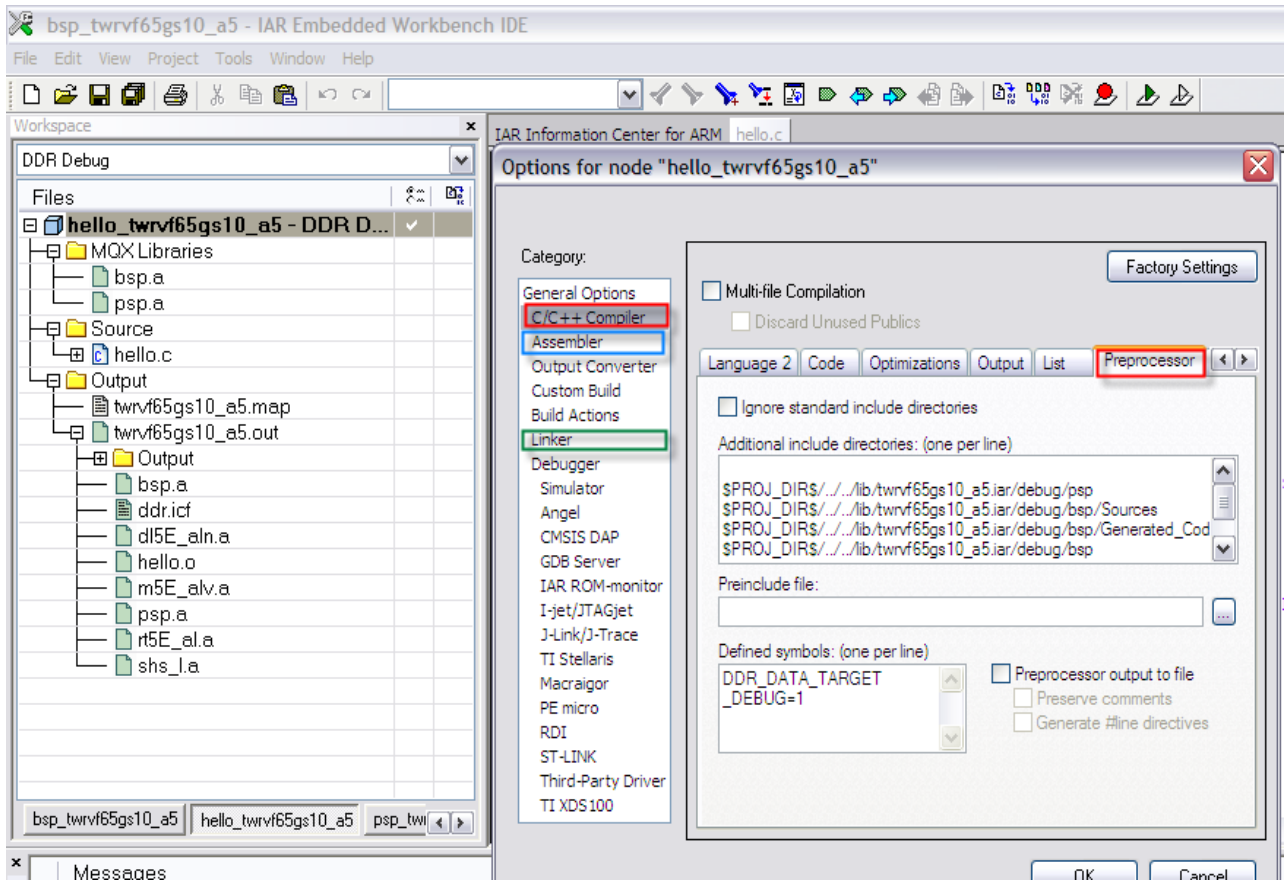
Figure 10. MQX IAR compiler paths.

The path of the linker is:

$PROJ_DIR$/../../lib/twrvf65gs10_a5.iar/debug/bsp/ddr.icf

The *.icf* file may vary depending on the target DDR Debug, DDR Release, Int Ram Debug, or Int Ram Release.
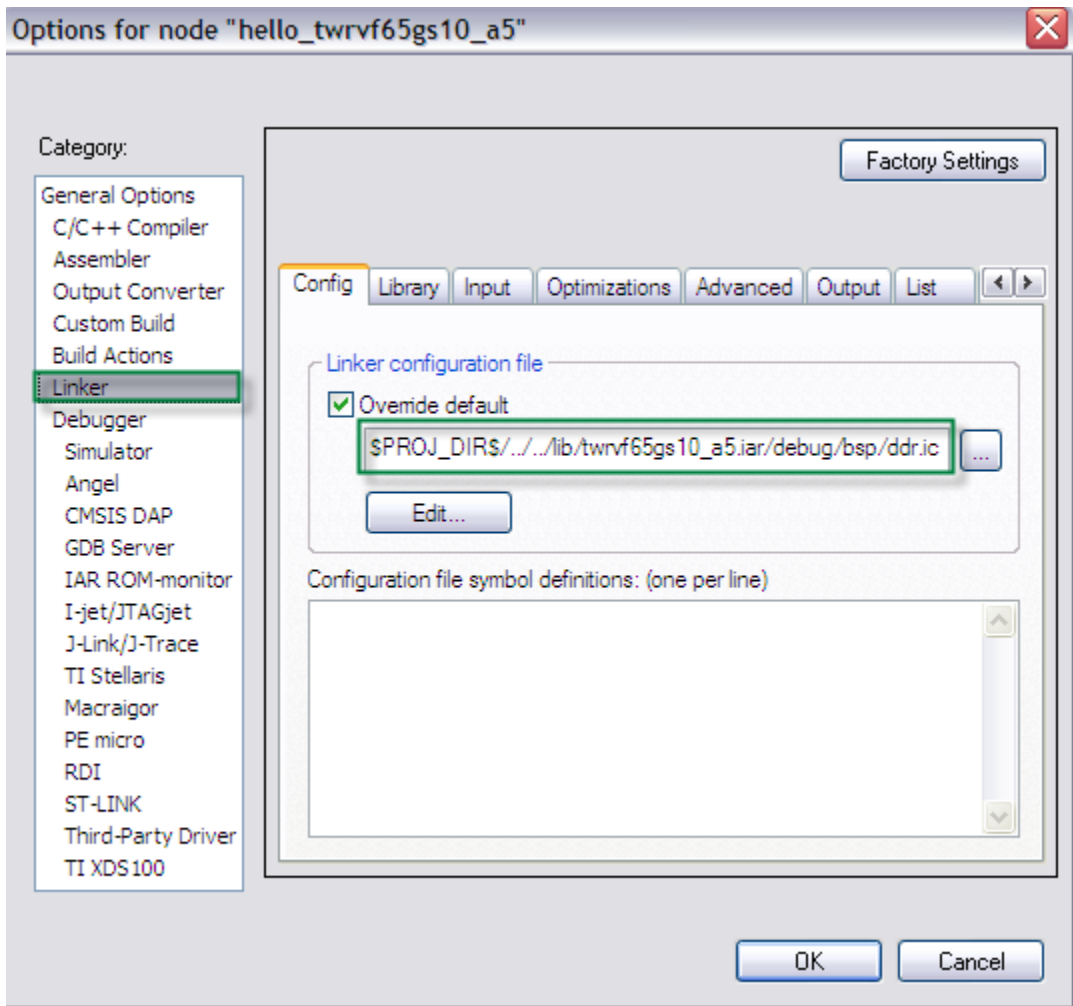
Figure 11. MQX IAR linker paths.

When using the DDR target, you need to change the path of the *.mac* file in the debugger options Replace the original path by:

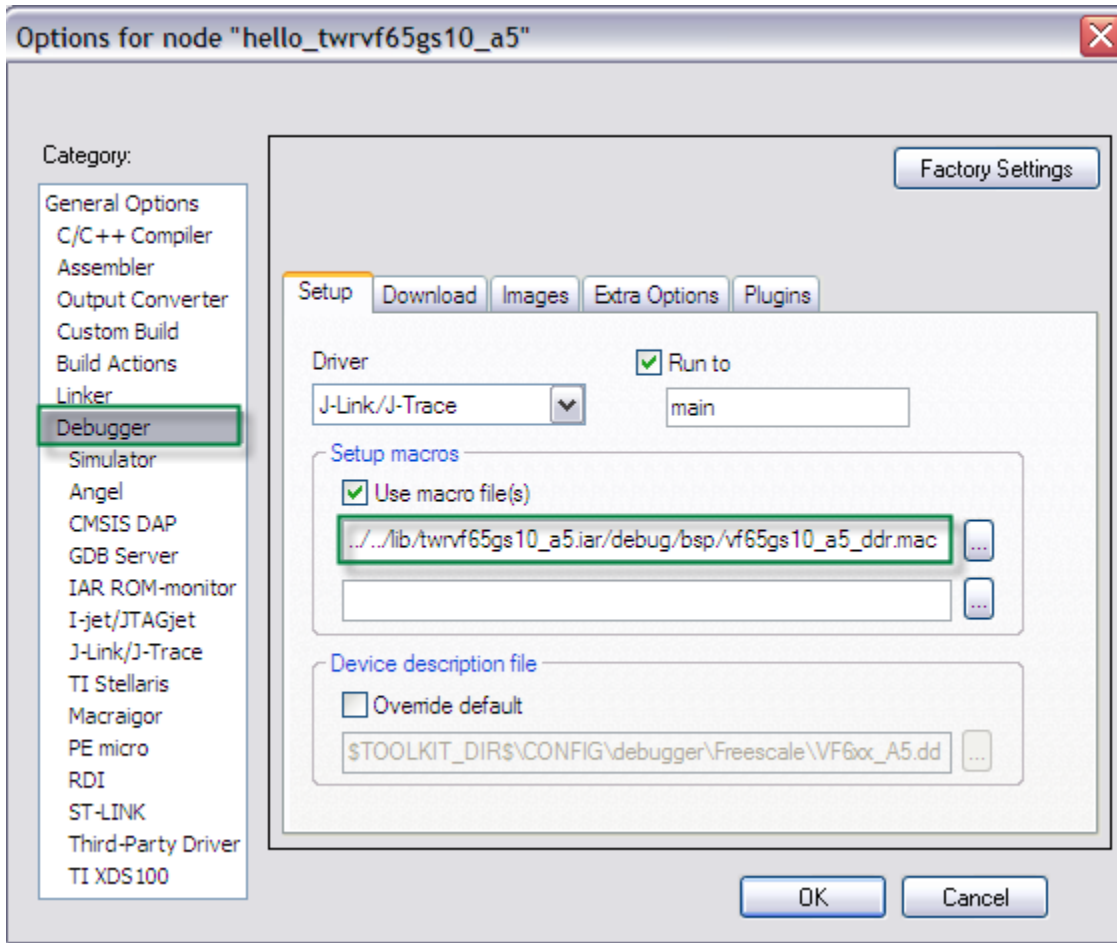*$PROJ_DIR$/../../lib/twrvf65gs10_a5.iar/debug/bsp/vf65gs10_a5_ddr.mac*

Figure 12. MQX IAR debug setup paths.

The above steps should be applied for all different targets.

3.5 Freescale MQX stationery projects for Vybrid in DS-5 in a Linux Host

DS-5 lacks a wizard to create an MQX stationary project. Therefore, the best procedure to create a new MQX project in DS-5 is to copy an existing project to a desired folder.
The compiler, assembler, and linker paths need to be changed. The below steps explain how to create a new project:

1. Copy the most similar example of the MQX installation folder to the desired folder:
In this instance, the hello example found at *\Freescale_MQX_4_0\mqx\examples\hello\ds5* is copied to a new folder named Applications.
This folder is located at: *Freescale_MQX_4_0\Applications*
Copy the *.c* file found in the example project too into the Source folder. The structure of the files should look as shown in the figure 13.
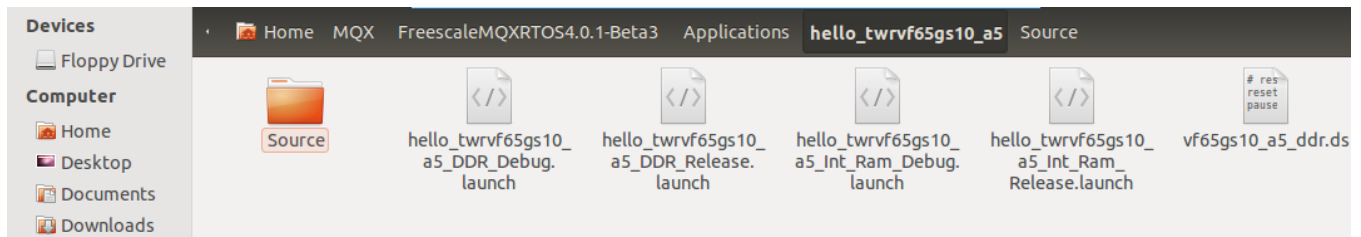

Figure 13. Freescale MQX Stationery Project layout for Vybrid in DS-5.

2. Open DS-5 and point the workspace to the Applications folder created in the step 1. Import the poject into the workspace.
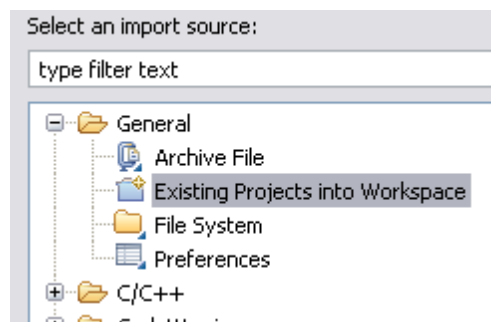Right click project window, Import…->General->Existing Projects into Workspace.

Figure 14. MQX DS-5 import Project.

3. Under project properties change the compiler, linker, and library paths for your project.
Right click on the project window, Properties…->C/C++ Build

Replace the original paths by the ones below under ARM C Compiler…->Inlcudes:

```
"${ProjDirPath}/../../lib/twrvf65gs10_a5.ds5/debug/psp"
"${ProjDirPath}/../../lib/twrvf65gs10_a5.ds5/debug/bsp/Sources"
"${ProjDirPath}/../../lib/twrvf65gs10_a5.ds5/debug/bsp/Generated_Code
"
"${ProjDirPath}/../../lib/twrvf65gs10_a5.ds5/debug/bsp"
"${ProjDirPath}/../../lib/twrvf65gs10_a5.ds5/debug"
"${ds5_install_path}/include"
```

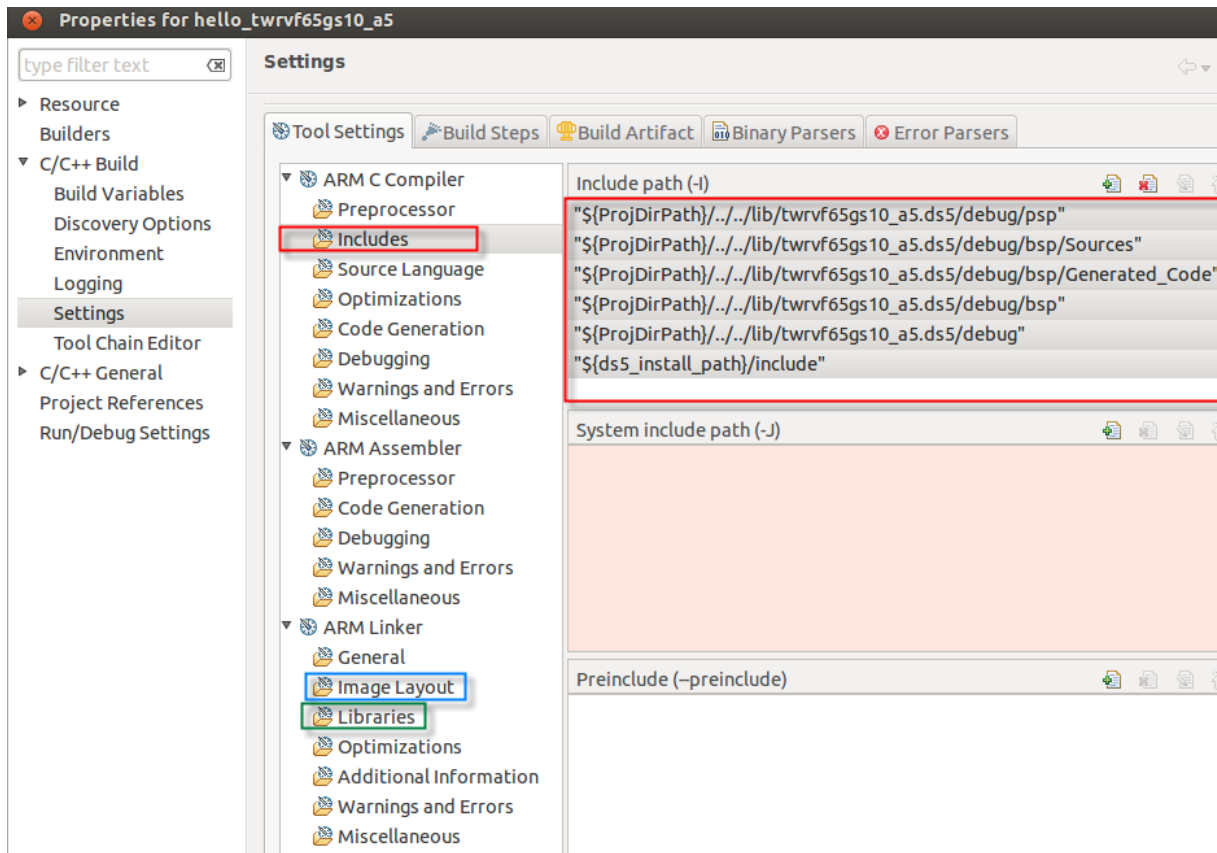

Figure 15. MQX DS-5 compiler paths.

The path of the linker is:

${ProjDirPath}/../../lib/twrvf65gs10_a5.ds5/debug/bsp/ddr.scf
The *.scf* file may vary depending on the target DDR Debug, DDR Release, Int Ram Debug, or Int Ram Release.
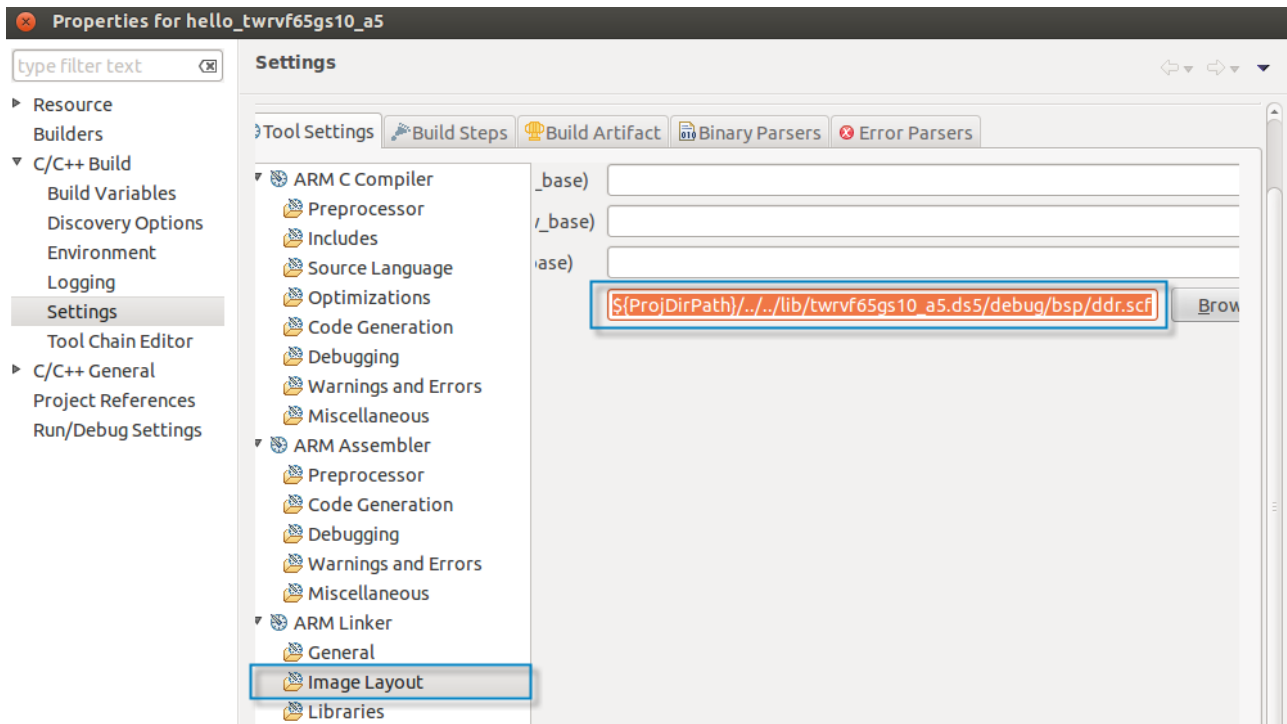
Figure 16. MQX DS-5 linker paths.

The library paths should be replaced by the:

${ProjDirPath}/../../lib/twrvf65gs10_a5.ds5/debug/psp/psp.a
${ProjDirPath}/../../lib/twrvf65gs10_a5.ds5/debug/bsp/bsp.a
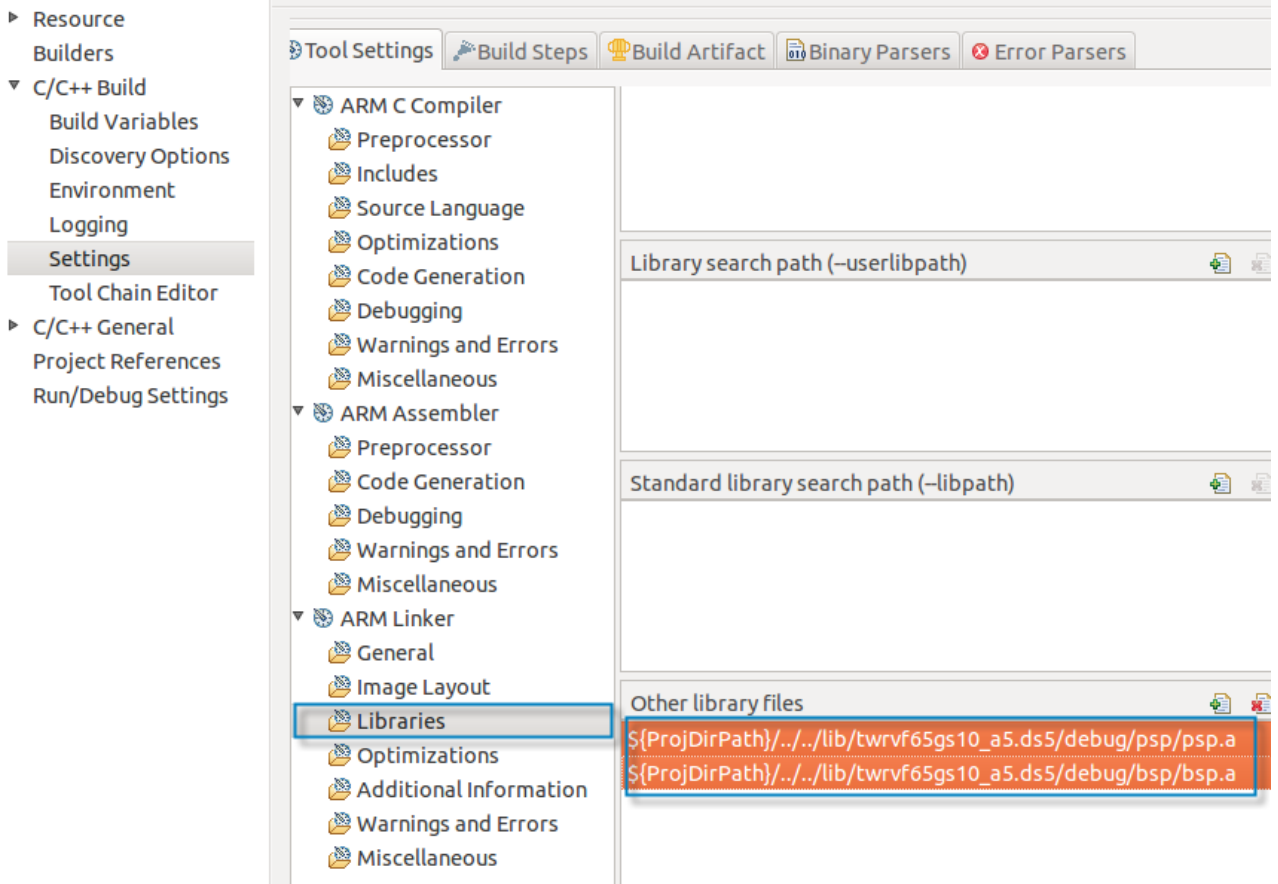
Figure 17. MQX DS-5 Library paths.

The above steps should be applied for all different targets.

4 Developing the first Freescale MQX application

This section describes the creation of a simple application that blinks LEDs on the TWRMCF51JF, M52259DEMO and TWR-K60D100M boards. There are four LEDs on each board and there are four tasks, one for each LED. Each task blinks its own LED and each task has a different time delay.

Each Freescale MQX application can be migrated from one platform to another. All steps in this section apply to the three CodeWarrior projects, TWRMCF51JF_MQX_RTOS, M52259DEMO_MQX_RTOS and TWR-K60D100M_MQX_RTOS. The Board Support Package (BSP) is not the same for all MCUs. This is the only part of the code that has to be different from one project to another. Each MCU has its own BSP and LWGPIO driver.

4.1 Modify main.c and main.h modules

The Main_task is auto-started at initialization. MQX RTOS creates one instance of the task with a creation parameter equaling zero. The application defines the task template index (MAIN_TASK), now named init_task. This task starts the other four tasks for LEDs. In total there are five tasks, the initialization task and one task for each LED.

1. In the main.h module replace the following lines:
   ```
   #define MAIN_TASK 1
   extern void Main_task(uint32_t);
   ```

   With the next lines:
   ```
   #define INIT_TASK 5
   extern void init_task(uint32_t);
   ```

2. In *main.c*, modify the MQX_template_list entry for the Main_Task. Replace the following line:
   ```
   {MAIN_TASK, Main_task, 1500, 9, "main", MQX_AUTO_START_TASK},
   ```

   With the next line:
   ```
   {INIT_TASK, init_task, 1500, 9, "init", MQX_AUTO_START_TASK, 0, 0},
   ```

   The parameters in the MQX_template_list are:

   — INIT_TASK: The identification (ID) number assigned to the task. The init_task has the ID number 5. Each task should have a unique ID number.
   — init_task: The callback function. This function is the code that the task executes once it starts.
   — 1500: Stack size. Memory assigned to the task for stack usage.
   — 9: Task priority. Lower number is for higher priorities. Do not assign higher priorities (0-8) to user tasks. Those are assigned to kernel and other important tasks. Assign consecutive priority numbers to ensure a better performance. More than one task can have the same priority level.
   — "init": Task name.

— MQX_AUTO_START_TASK: Indicates if the task starts automatically after the OS finished the startup.

3. To add the four LED tasks, first define them in the *main.h* module:
```
#define LED1_TASK 6
#define LED2_TASK 7
#define LED3_TASK 8
#define LED4_TASK 9
```

4. In *main.c*, add the entries for four LED tasks in the MQX_template_list after the initialization task entry in the array. At the end, the MQX_template_list looks like the following:
```
TASK_TEMPLATE_STRUCT MQX_template_list[] =
{
     {INIT_TASK, init_task, 1500, 9, "init", MQX_AUTO_START_TASK, 0, 0},
     {LED1_TASK, led1_task, 1500, 10, "led1", 0, 0, 0},
     {LED2_TASK, led2_task, 1500, 11, "led2", 0, 0, 0},
     {LED3_TASK, led3_task, 1500, 12, "led3", 0, 0, 0},
     {LED4_TASK, led4_task, 1500, 13, "led4", 0, 0, 0},
     {0, 0, 0, 0, 0, 0, 0, 0}
};
```
NOTE
The last entry in the MQX_template_list must be all zeros.

5. In *main.h*, add the externs for the ledx_task functions after the line extern void init_task(uint32_t);
```
extern void led1_task(uint32_t);
extern void led2_task(uint32_t);
extern void led3_task(uint32_t);
extern void led4_task(uint32_t);
```

6. In *main.c*, comment out the Main_task.
```
/*void Main_task(uint32_t initial_data)
{
print("/n Hello World /n");
_mqx_exit(0);
}
*/
```

See Appendix A, "main.c File," to see the *main.c* module source code for details.

4.2   Create tasks.c file

For better organization, create a new *.c* file for tasks functions.

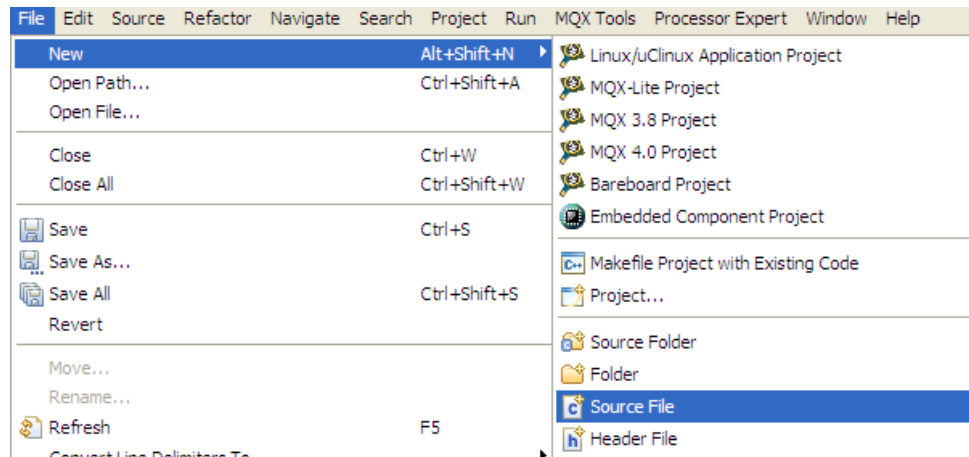1. Go to File menu and click on 'New->Source File.'

Figure 18. New .c File.

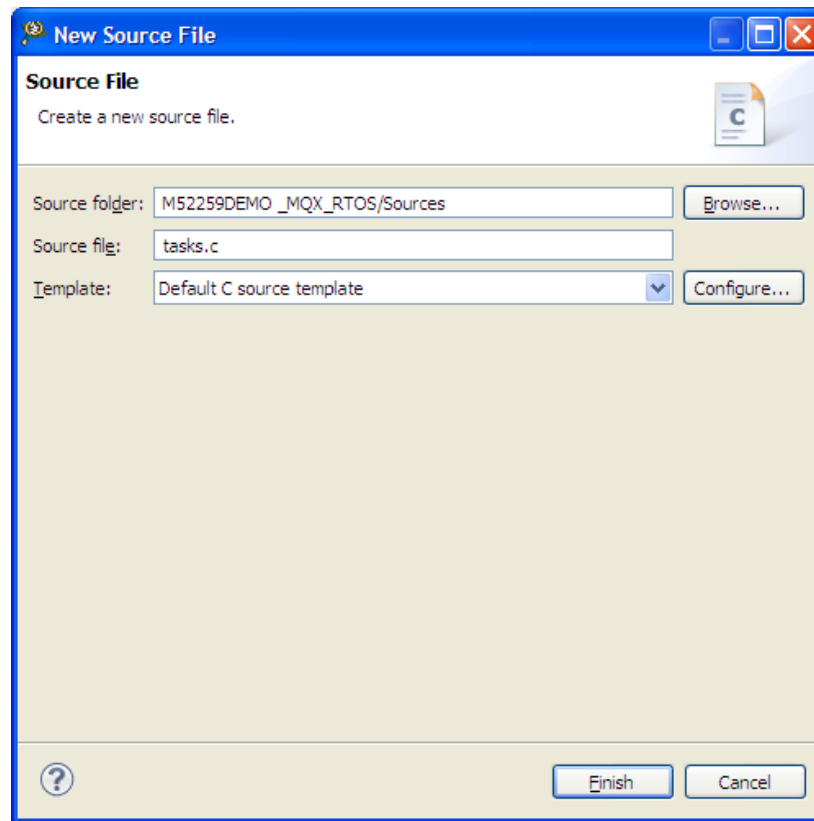2. Select the right source folder and name for the new file.

Figure 19. tasks.c File.

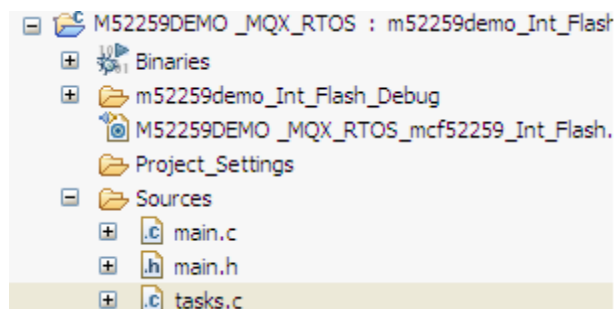3. After the step #2, the directory structure must look as shown in Figure 20.

Figure 20. Adding *tasks.c* File to Project Directory Structure.

4. Open the tasks.c module and add the following includes:
```
#include <main.h>
#include <fio.h>
```

5. Define the function prototypes of the tasks we are about to use.

```
void led1_task(uint32_t);
void led2_task(uint32_t);
void led3_task(uint32_t);
void led4_task(uint32_t);
```

6. Write the code in the init_task function to create the other four tasks that are already defined in the MQX_template_list template. Use the following code to create and run the led1_task in the *tasks.c* module:

```
_task_id task_id;
  printf("\n Starting application \n");

  task_id = _task_create(0, LED1_TASK, 0);
  if (task_id == MQX_NULL_TASK_ID)
  {
    printf("\n Could not create LED1_TASK\n");
  }
  else
  {
    printf(" LED1_TASK created \n");
  }
```

7. The previous code can be reused to create the other three tasks. Change only the second parameter of the _task_create() function with following definitions:

```
task_id = _task_create(0, LED2_TASK, 0);
…
task_id = _task_create(0, LED3_TASK, 0);
…
task_id = _task_create(0, LED4_TASK, 0);
```

8. Write the tasks code. Each task uses LWGPIO driver to toggle a pin.
   The lwgpio_init() function has to be called prior to calling any other API function of the LWGPIO driver. This function initializes the LWGPIO_STRUCT structure. The pointer to the LWGPIO_STRUCT is passed as a handle parameter. To identify the pin, platform-specific LWGPIO_PIN_ID number is used. It is necessary to have a variable to assign the pointer to the LWGPIO_STRUCT. To do that, add the following line:

```
LWGPIO_STRUCT led1;
```

   The following lines initialize lwgpio handle for LWGPIO_MUX_xxx_GPIO pin. The code is the same for all projects; the BSP already knows the address and the pin number for each GPIO PORT and each PIN. This is defined in *mqx/source/bsp/<bsp_name>/<bsp_name>.h file*.

```
if (!lwgpio_init(&led1, BSP_LED1, LWGPIO_DIR_OUTPUT,
LWGPIO_VALUE_NOCHANGE))
    {
```

```
      printf("Initializing Port TC pin0 GPIO of the MCF52259 as output
failed.\n");
      _task_block();
    }
```

The parameters for lwgpio_init() are:
- led1: the pointer to the LWGPIO_STRUCT structure.
- BSP_LED1: LWGPIO_PIN_ID number identifying pin (platform and peripheral specific).
- LWGPIO_DIR_OUTPUT: LWGPIO_DIR enum value for initial direction control.
- LWGPIO_VALUE_NOCHANGE: LWGPIO_VALUE enum value for initial output control.

Once it is initialized, assign the requested functionality to the pin such as GPIO mode or any other peripheral mode. The lwgpio_set_functionality() function sets the functionality of the pin. The value of the functionality parameter represents the number stored in the multiplexer register field which selects the desired functionality. For the GPIO mode, you can use the pre-defined macros which can be found in the lwgpio_ <mcu>.h file. Add the following lines switch pin functionality to GPIO mode:

```
lwgpio_set_functionality(&led1, BSP_LED1_MUX_GPIO);
```

The lwgpio_set_value() function sets the pin state (low or high) of the specified pin. To toggle the pin value write the following code:

```
while (TRUE)
  {
    _time_delay(1111);
    lwgpio_set_value(&led1, value); /* toggle pin value */
    value = value^1;
  }
```

9. Create the other three functions. The name of each function, delay, and the first parameter must be different. In LWGPIO functions use (lwgpio_init(), lwgpio_set_functionality(), lwgpio_set_value()).

See Appendix C, "tasks.c File," to see *tasks.c* module source code for details.
At this point it is possible to build the project, flash it to the board and run the project; for more details about building, flashing and running, see the compiler documentation of your choice located at: *C:\Freescale\Freescale_MQX_4_0\doc\tools*

5   Task Aware Debugging (TAD) Tool

This section discusses CodeWarrior MQX TAD, which is available on CodeWarrior Professional Edition. The Task Aware Debugging (TAD) Tool is a useful and powerful instrument to debug the MQX application. CodeWarrior Special and Standard editions can create and run an MQX application, but cannot run TAD.
To use TAD:

1. Debug the application in CodeWarrior using the debug button.
2. To stop the application, use a breakpoint or use the suspend button.
3. Once the application breaks, the menu enables two options: MQX RTOS and RTCS.

   The MQX menu shows information about the core and tasks running at that moment in MQX RTOS, information such as memory usage, stack usage, semaphores, mutexes, events, kernel data, errors, and so on. The RTCS menu shows information about the TCPIP stack, configuration, sockets, TCP stats, UDP stats, IP stats, and so on. Figure 11 shows both MQX RTOS and RTCS.
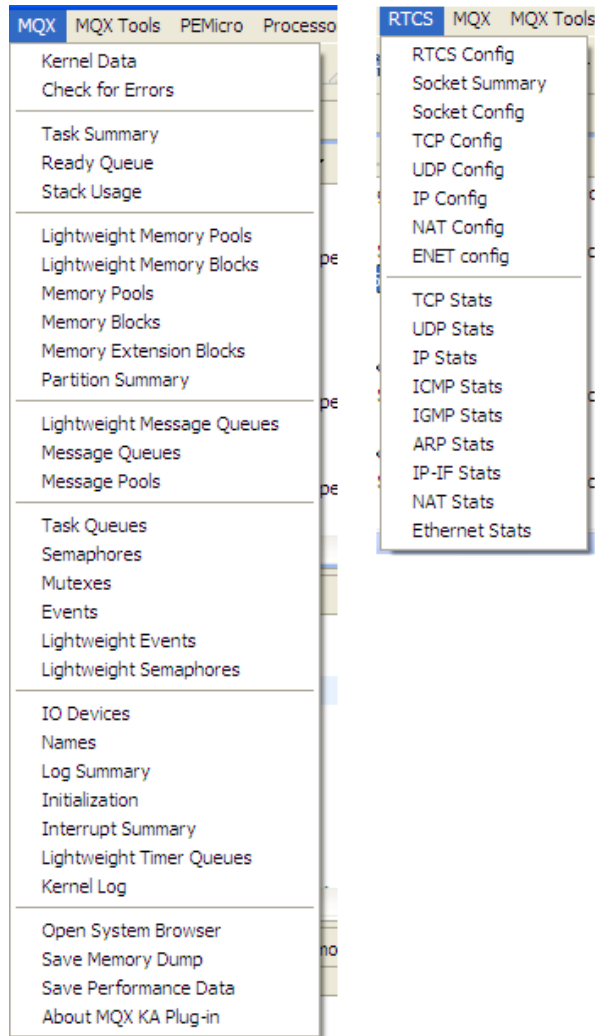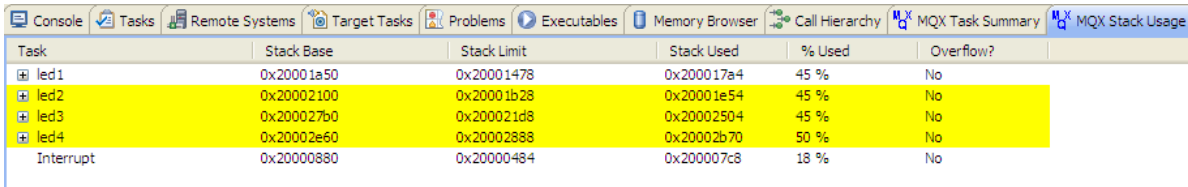
Figure 21. MQX RTOS and RTCS menus for the TAD

4. Go to the MQX menu and click on 'Task Summary' option. Figure 22 shows all tasks in the MQX RTOS. It shows the priority, state, and the task error if any.

Task Name	Task ID	TD	Priority	State	Task Error Code
⊞ led1	0x10002	0x200013b0	10	Time delay blocked	OK (0x0000)
⊞ led2	0x10003	0x20001a60	11	Time delay blocked	OK (0x0000)
⊞ led3	0x10004	0x20002110	12	Time delay blocked	OK (0x0000)
⊞ led4	0x10005	0x200027c0	13	Time delay blocked	OK (0x0000)

Figure 22. Task summary.

5. Go to the MQX menu and click on 'Stack Usage' option. Figure 23 shows the memory used by each task within its stack. It also shows if an overflow happens.

Task	Stack Base	Stack Limit	Stack Used	% Used	Overflow?
⊞ led1	0x20001a50	0x20001478	0x200017a4	45 %	No
⊞ led2	0x20002100	0x20001b28	0x20001e54	45 %	No
⊞ led3	0x200027b0	0x200021d8	0x20002504	45 %	No
⊞ led4	0x20002e60	0x20002888	0x20002b70	50 %	No
Interrupt	0x20000880	0x20000484	0x200007c8	18 %	No

Figure 23. Stack usage.

Appendix A main.c file

```c
/****************************************************************************
*
*    This file contains MQX only stationery code.
*
****************************************************************************/
#include "main.h"


#if !BSPCFG_ENABLE_IO_SUBSYSTEM
#error This application requires BSPCFG_ENABLE_IO_SUBSYSTEM defined non-zero in
user config.h. Please recompile BSP with this option.
#endif


#ifndef BSP_DEFAULT_IO_CHANNEL_DEFINED
#error This application requires BSP_DEFAULT_IO_CHANNEL to be not NULL. Please set
corresponding BSPCFG_ENABLE_TTYx to non-zero in user_config.h and recompile BSP
with this option.
#endif


TASK_TEMPLATE_STRUCT MQX_template_list[] =
{
/*  Task number, Entry point, Stack, Pri, String, Auto? */
   {INIT_TASK, init_task, 1500, 9, "init", MQX_AUTO_START_TASK, 0, 0},
   {LED1_TASK, led1_task, 1500, 10, "led1", 0,                  0, 0},
   {LED2_TASK, led2_task, 1500, 11, "led2", 0,                  0, 0},
   {LED3_TASK, led3_task, 1500, 12, "led3", 0,                  0, 0},
   {LED4_TASK, led4_task, 1500, 13, "led4", 0,                  0, 0},
   {0,          0,          0,    0,   0,     0,                  0, 0}
};


/*TASK*-------------------------------------------------------
*
* Task Name    : Main_task
* Comments     :
*    This task prints " Hello World "
*
*END*-------------------------------------------------------*/

/*void Main_task(uint32_t initial_data)
{
   printf("\n Hello World \n");

   _mqx_exit(0);
}
*/
/* EOF */
```

Appendix B main.h file

```c
#ifndef __main_h_
#define __main_h_
#include <mqx.h>
#include <bsp.h>

#define INIT_TASK 5
#define LED1_TASK 6
#define LED2_TASK 7
#define LED3_TASK 8
#define LED4_TASK 9

extern void init_task(uint32_t);
extern void led1_task(uint32_t);
extern void led2_task(uint32_t);
extern void led3_task(uint32_t);
extern void led4_task(uint32_t);


/* PPP device must be set manually and
** must be different from the default IO channel (BSP_DEFAULT_IO_CHANNEL)
*/
#define PPP_DEVICE       "ittyb:"

/*
** Define PPP_DEVICE_DUN only when using PPP to communicate
** to Win9x Dial-Up Networking over a null-modem
** This is ignored if PPP_DEVICE is not #define'd
*/
#define PPP_DEVICE_DUN  1

/*
** Define the local and remote IP addresses for the PPP link
** These are ignored if PPP_DEVICE is not #define'd
*/
#define PPP_LOCADDR     IPADDR(192,168,0,216)
#define PPP_PEERADDR    IPADDR(192,168,0,217)

/*
** Define a default gateway
*/
#define GATE_ADDR       IPADDR(192,168,0,1)

#endif /* __main_h_ */
```

Appendix C task.c file

```c
#include <main.h>
#include <fio.h>

void init_task(uint32_t);
void led1_task(uint32_t);
void led2_task(uint32_t);
void led3_task(uint32_t);
void led4_task(uint32_t);

void init_task(uint32_t initial_data)
{
  _task_id task_id;
  printf("\n Starting application \n");

  task_id = _task_create(0, LED1_TASK, 0);
  if (task_id == MQX_NULL_TASK_ID)
  {
    printf("\n Could not create LED1_TASK\n");
  }
  else
  {
    printf(" LED1_TASK created \n");
  }

  task_id = _task_create(0, LED2_TASK, 0);
  if (task_id == MQX_NULL_TASK_ID)
  {
    printf("\n Could not create LED2_TASK\n");
  }
  else
  {
    printf(" LED2_TASK created \n");
  }

  task_id = _task_create(0, LED3_TASK, 0);
  if (task_id == MQX_NULL_TASK_ID)
  {
    printf("\n Could not create LED3_TASK\n");
  }
  else
  {
    printf(" LED3_TASK created \n");
  }

  task_id = _task_create(0, LED4_TASK, 0);
  if (task_id == MQX_NULL_TASK_ID)
  {
    printf("\n Could not create LED4_TASK\n");
  }
  else
```

```
  {
    printf(" LED4_TASK created \n");
  }
}


/*TASK*------------------------------------------------------
*
* Task Name : led1_task
* Comments :
* This task toggles the LED1 every 1111 milliseconds
*
*END*--------------------------------------------------------*/
void led1_task(uint32_t initial_data)
{
  int value = 0;
  LWGPIO_STRUCT led1;

  printf("\n Led1 task \n");

  /* initialize lwgpio handle for LWGPIO_MUX_TC0_GPIO pin (defined in
mqx/source/bsp/<bsp_name>/<bsp_name>.h file) */
  if (!lwgpio_init(&led1, BSP_LED1, LWGPIO_DIR_OUTPUT, LWGPIO_VALUE_NOCHANGE))
  {
    printf("Initializing Port TC pin0 GPIO as output failed.\n");
    _task_block();
  }

  /* swich pin functionality to GPIO mode */
  lwgpio_set_functionality(&led1, BSP_LED1_MUX_GPIO);

  while (TRUE)
  {
    _time_delay(1111);
    lwgpio_set_value(&led1, value); /* toggle pin value */
    value = value^1;
  }

}

void led2_task(uint32_t initial_data)
{
  int value = 0;
  LWGPIO_STRUCT led2;

  printf("\n led2 task \n");

  /* initialize lwgpio handle for LWGPIO_MUX_G1_GPIO pin (defined in
mqx/source/bsp/<bsp_name>/<bsp_name>.h file) */
  if (!lwgpio_init(&led2, BSP_LED2, LWGPIO_DIR_OUTPUT, LWGPIO_VALUE_NOCHANGE))
  {
    printf("Initializing Port TC pin1 GPIO as output failed.\n");
    _task_block();
  }
  /* swich pin functionality to GPIO mode */
  lwgpio_set_functionality(&led2, BSP_LED2_MUX_GPIO);
```

```c
  while (TRUE)
  {
    _time_delay(2222);
    lwgpio_set_value(&led2, value); /* toggle pin value */
    value = value^1;
  }
}

void led3_task(uint32_t initial_data)
{
  int value = 0;
  LWGPIO_STRUCT led3;

  printf("\n led3 task \n");

  /* initialize lwgpio handle for LWGPIO_MUX_TC_GPIO pin (defined in
mqx/source/bsp/<bsp_name>/<bsp_name>.h file) */
  if (!lwgpio_init(&led3, BSP_LED3, LWGPIO_DIR_OUTPUT, LWGPIO_VALUE_NOCHANGE))
  {
    printf("Initializing Port TC pin2 GPIO as output failed.\n");
    _task_block();
  }
  /* swich pin functionality (MUX) to GPIO mode */
  lwgpio_set_functionality(&led3, BSP_LED3_MUX_GPIO);

  while (TRUE)
  {
    _time_delay(3333);
    lwgpio_set_value(&led3, value); /* toggle pin value */
    value = value^1;
  }
}

void led4_task(uint32_t initial_data)
{
  int value = 0;
  LWGPIO_STRUCT led4;

  printf("\n led4 task \n");

  /* initialize lwgpio handle for LWGPIO_MUX_TC_GPIO pin (defined in
mqx/source/bsp/<bsp_name>/<bsp_name>.h file) */
  if (!lwgpio_init(&led4, BSP_LED4, LWGPIO_DIR_OUTPUT, LWGPIO_VALUE_NOCHANGE))
  {
    printf("Initializing Port TC pin3 GPIO as output failed.\n");
    _task_block();
  }
  /* swich pin functionality (MUX) to GPIO mode */
  lwgpio_set_functionality(&led4, BSP_LED4_MUX_GPIO);
```

```
  while (TRUE)
  {
    _time_delay(4444);
    lwgpio_set_value(&led4, value); /* toggle pin value */
    value = value^1;
  }
}


/* EOF */
```