Freescale Semiconductor
Application Note

Document Number: AN3905
Rev. 1, 11/2009

Writing Your First MQX Application

by: Antonio Ramos, Luis Garabito,
Applications Engineering
Freescale Technical Support

1 Introduction

The time invested to develop MQX applications for the first time in a new environment can be significant. It is necessary to understand how the environment works and then be able to generate applications for this environment.

The purpose of this application note is to provide:

- The knowledge that enables developers to start developing their first application on Freescale MQX RTOS quickly and easily.
- The basic understanding to develop Freescale MQX applications.

This application note is based on the ColdFire MCF51CN128 and MCF5225x processors. For CodeWarrior, the following boards are used as an example:

- The TWRMCF51CN board for microcontrollers
- The M52259DEMO board for ColdFire

Contents

1 Introduction. 1
2 Freescale MQX RTOS Install and Setup 2
2.1 Download Freescale MQX RTOS 2
2.2 Install Freescale MQX RTOS 2
2.3 Setup Freescale MQX RTOS 2
3 Freescale MQX Stationery Projects 2
3.1 TWRMCF51CN_MQX_RTOS Project in CodeWarrior for Microcontrollers 6.2.2. 3
3.2 M52259DEMO_MQX_RTOS Project in CodeWarrior for ColdFire 7.1.2. 4
4 Developing Your First Freescale MQX Application . . . 5
4.1 Modify main.c and main.h Modules. 5
4.2 Create ioDriver.c module. 6
4.3 Create tasks.c Module. 10
5 Task Aware Debugging (TAD) Tool 12
Appendix A main.c Module 15
Appendix B ioDriver.c Module for TWRMCF51CN_MQX_RTOS Project 16
Appendix C ioDriver.c Module for M52259DEMO_MQX_RTOS Project 18
Appendix D tasks.c Module 20
Appendix E main.h Module 23

freescale™ semiconductor

2 Freescale MQX RTOS Install and Setup

Freescale MQX RTOS is a set of source code, example code, CodeWarrior plug-ins, 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 www.freescale.com and register with Freescale to get the required access.
2. Go to http://www.freescale.com/mqx, login, and download the MQX installer.

2.2 Install Freescale MQX RTOS

To install Freescale MQX RTOS:

1. Execute the installation wizard. The default settings install Freescale MQX in the path: C:\Program Files\Freescale\Freescale MQX 3.x
2. You must read the FSL_MQX_release_notes.pdf file that has important information about the source code and example code delivered in the release.

NOTE

If you have installed MQX in a non-standard location, you will need to rebuild the libraries. Rebuilding the libraries is described in the release notes.

2.3 Setup Freescale MQX RTOS

Freescale MQX has a directory structure. The detailed information about this structure can be found in the FSL_MQX_release_notes.pdf file. The developer can change the 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 that needs to be modified can be found in the path: <install_dir>/config/<board>/user_config.h.

After a modification is done in the user_config.h, it is necessary to recompile the libraries. Open the CodeWarrior project <install_dir>/config/<board>/build_libs.mcp and compile it. The libraries are saved in the folder <install_dir>/lib/<board>. No changes will take effect if you try to change anything inside the <install_dir>/lib/ folder, because this is the output folder after libraries compilation.

3 Freescale MQX Stationery Projects

Freescale MQX is supported on:

- CodeWarrior for microcontrollers 6.2.2
- CodeWarrior for ColdFire 7.1.2

Each CodeWarrior environment supports the platforms delivered with the release.

3.1 TWRMCF51CN_MQX_RTOS Project in CodeWarrior for Microcontrollers 6.2.2

Follow these steps to create and configure a Freescale MQX stationery project for the TWRMCF51CN board:

1. Open CodeWarrior for microcontrollers 6.2.2.
2. Go to ‘File’ menu and click on ‘New Project…’
3. Select ‘Freescale MQX 3.3 Stationery,’ 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: TWRMCF51CN_MQX_RTOS
   — Location: C:\myMQXprojects\TWRMCF51CN_MQX_RTOS
4. Expand the selected demo or evaluation board. For this example, TWRMCF51CN board is selected.
5. It is necessary to select the libraries that are used. For this example, select ‘MQX Only.’

Figure 1 shows the creation of the stationery project.

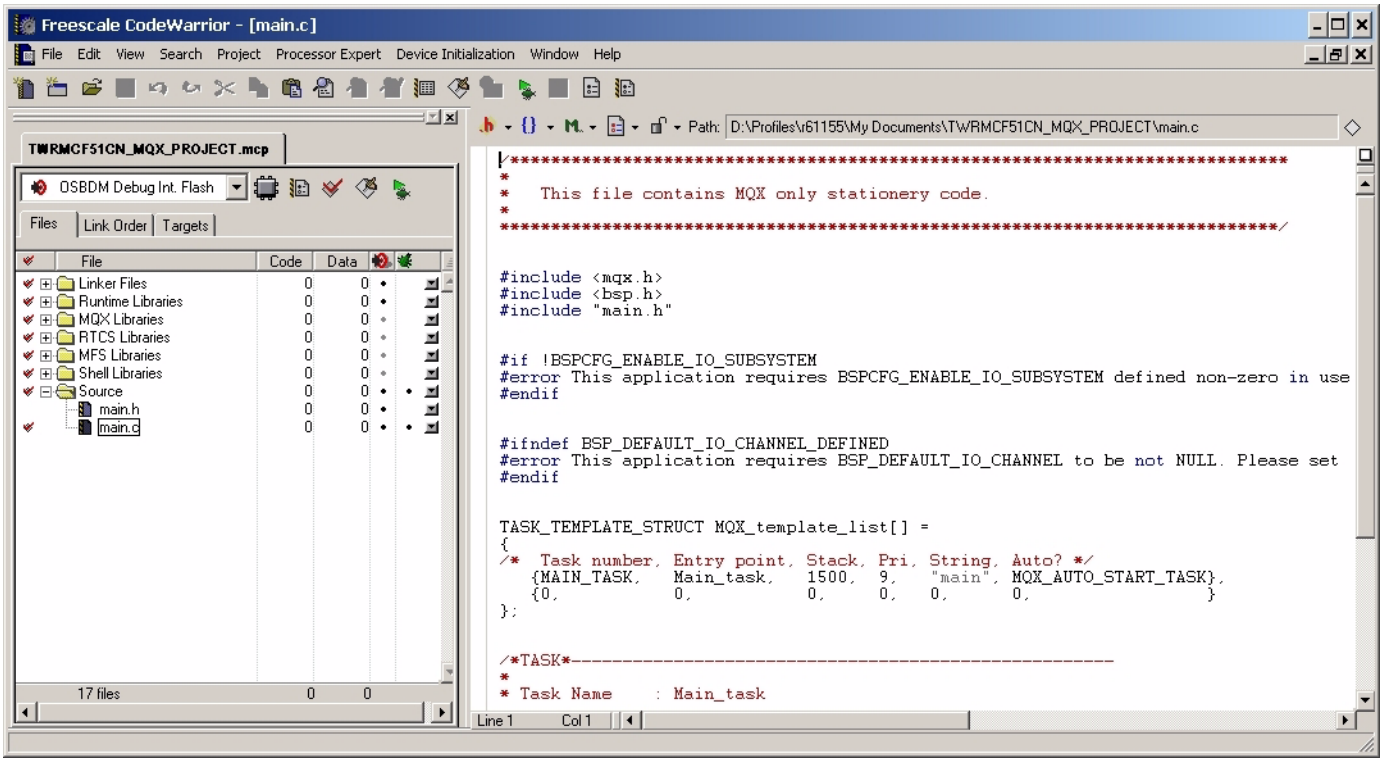

Figure 1. Freescale MQX Stationery Project for TWRMCF51CN Board

3.2 M52259DEMO_MQX_RTOS Project in CodeWarrior for ColdFire 7.1.2

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

1. Open CodeWarrior 7.1.2.
2. Go to ‘File’ menu and click on ‘New Project…’
3. Select ‘Freescale MQX 3.3 Stationery’, enter a ‘Project name,’ select a ‘Location,’ and click on the ‘Ok’ button. For this example, the folowing values are set in the dialog window:
   — Project name: M52259DEMO_MQX_RTOS
   — Location: C:\myMQXprojects\M52259DEMO_MQX_RTOS
4. Expand the selected demo or evaluation board. For this example, the M52259DEMO board is selected.
5. It is necessary to select the libraries that are used. For this example we select ‘MQX Only.’
6. Click on the ‘Ok’ button.

Figure 2 shows the creation of the stationery project.

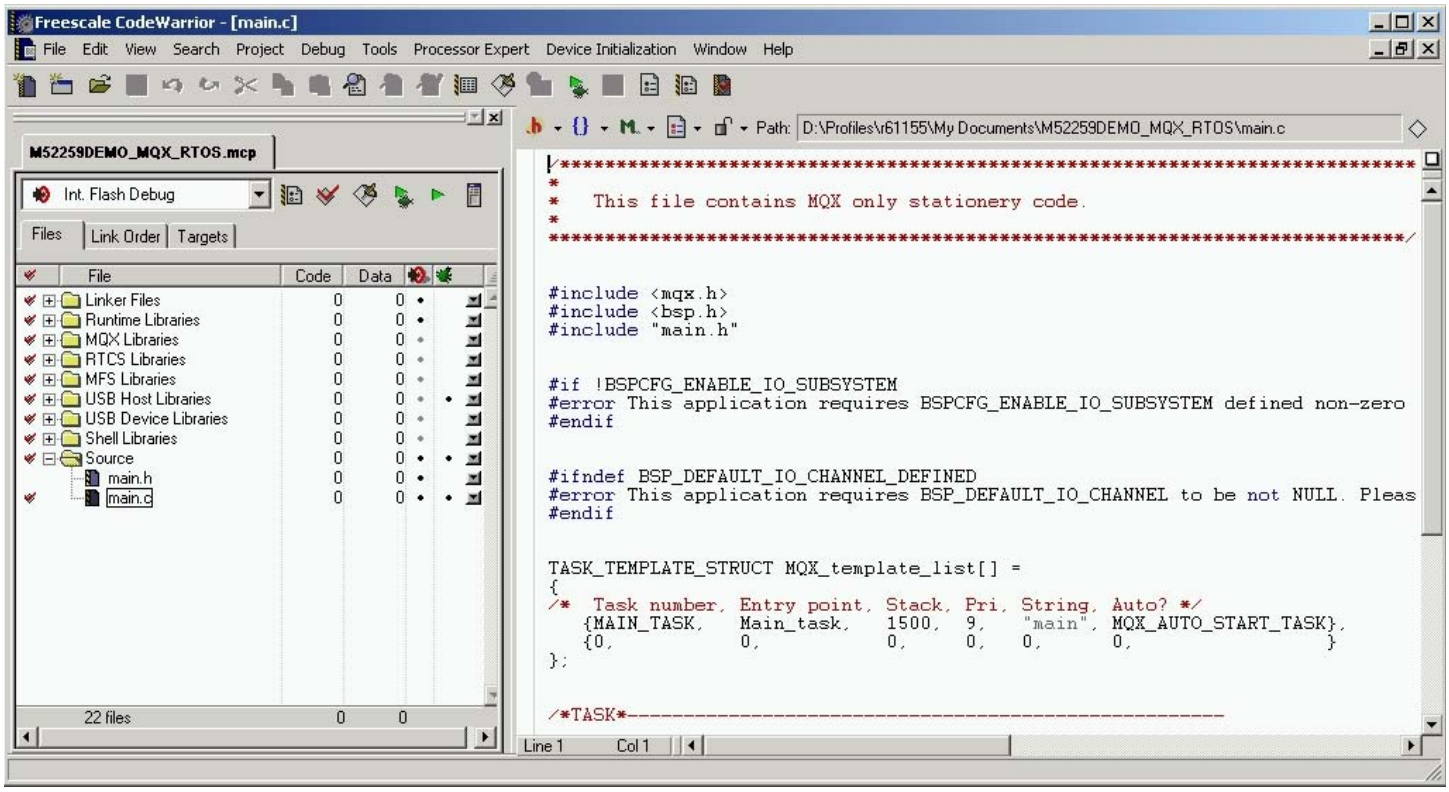

Figure 2. Freescale MQX Stationery Project for M52259DEMO Board

4 Developing Your First Freescale MQX Application

This section describes the creation of a simple application that blinks LEDs on the TWRMCF51CN and M52259DEMO boards. There are four LEDs on each board and there are four tasks, one for each LED. Every task blinks its own LED. Every task has a different time delay.

Each Freescale MQX application can be migrated from one platform to another. Most of the steps in this section apply to both CodeWarrior projects, TWRMCF51CN_MQX_RTOS and M52259DEMO_MQX_RTOS. It means that the same source code applies for both projects. There is only one difference. The Board Support Package (BSP) is not the same for both 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 GPIO driver. The difference between these is explained later.

4.1 Modify main.c and main.h Modules

The GPIO initialization happens within the Main_task, therefore it is necessary to change this task. This task is now named init_task. This task also 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(uint_32);
```

   With the next lines:

```
#define INIT_TASK 5
extern void init_task(uint_32);
```

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. There should not be more than one task with the same 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(uint_32);

```
extern void led1_task(uint_32);
extern void led2_task(uint_32);
extern void led3_task(uint_32);
extern void led4_task(uint_32);
```

6. In main.c, comment out the Main_task.

```
/*void Main_task(uint_32 initial_data)
{
    print("/n Hello World /n");

    _mqx_exit(0);
}
*/
```

Refer to Appendix A, "main.c Module," to see the main.c module source code for details.

4.2 Create ioDriver.c module

For a better organization, it is necessary to create a new module for the GPIO initialization functions.

1. Go to File menu and click on 'New Text File.'
2. Go to File menu and click on 'Save.'
   a) For CodeWarrior 6.2.2, in the 'Save As…' dialog box, search for the folder C:\myMQXprojects\TWRMCF51CN_MQX_RTOS and save the file as ioDriver.c.
   b) For CodeWarrior 7.1.2, in the 'Save As…' dialog box, search for the folder C:\myMQXprojects\M52259DEMO_MQX_RTOS and save the file as ioDriver.c.
3. In the project directory structure, search for the 'Source' folder. Right click and select "Add Files" and add the 'ioDriver.c.' When prompted with a pop up for "Add files to targets," use the default of all targets and click OK. After adding the file, the directory structure must look like Figure 3.

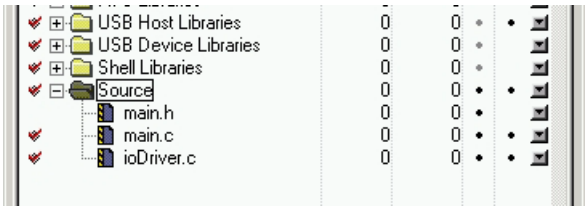

Figure 3. Adding ioDriver.c File to Project Directory Structure

4. Open the ioDriver.c module and add the following includes:

```
#include <mqx.h>
#include <bsp.h>
#include <fio.h>
#include <io_gpio.h>
```

5. It is necessary to have a variable to assign the pointer handler after the GPIO driver is opened. Add the following line after the includes in the ioDriver.c module:

```
static FILE_PTR output_port=NULL;
```

6. For each project, add the four LED defines in the ioDriver.c module, one for each LED. This step is important, because this is the only difference in the code for each project.

   Every define sets the GPIO PORT to be used and the PIN assigned to the LED define. These values are different for each BSP. The BSP already knows the address and the pin number for each GPIO PORT and each PIN. Refer to *TWRMCF51CN and M52259DEMO schematics* to verify the LED connected to each PORT.

   a) For the TWRMCF51CN_MQX_RTOS project, add the following defines.

```
#define LED_1    (GPIO_PORT_TE | GPIO_PIN3)
#define LED_2    (GPIO_PORT_TG | GPIO_PIN5)
#define LED_3    (GPIO_PORT_TE | GPIO_PIN5)
#define LED_4    (GPIO_PORT_TH | GPIO_PIN3)
```

   b) For the M52259DEMO_MQX_RTOS project, add the following defines.

```
#define LED_1    (GPIO_PORT_TC | GPIO_PIN0)
#define LED_2    (GPIO_PORT_TC | GPIO_PIN1)
#define LED_3    (GPIO_PORT_TC | GPIO_PIN2)
#define LED_4    (GPIO_PORT_TC | GPIO_PIN3)
```

7. Create the InitializeIO() function after adding the LED defines in the ioDriver.c module.

```
boolean InitializeIO(void)
{

}
```

8. Define the function prototype after the source line.

```
#define LED_4    (GPIO_PORT_TC | GPIO_PIN3)
```

   by adding the code:

```
boolean InitializeIO(void);
```

9. The GPIO driver receives an array where the PORT, PIN, and STATUS for that PIN are defined. Add the following code in the InitializeIO() function.

```
const uint_32 output_set[] =
{
```

```
  LED_1 | GPIO_PIN_STATUS_0,
  LED_2 | GPIO_PIN_STATUS_0,
  LED_3 | GPIO_PIN_STATUS_0,
  LED_4 | GPIO_PIN_STATUS_0,
  GPIO_LIST_END
};
```

10. To open the GPIO driver, use the gpio:write to indicate that these GPIO pins are output pins. The array output_set also has to be sent to the GPIO driver. Add the following code to the InitializeIO function:

```
/* Open and set port TC as output to drive LEDs */
output_port = fopen("gpio:write", (char_ptr) &output_set);
if (output_port)
{
  ResetOutputs();
}
return (output_port!=NULL);
```

11. Another function is necessary to set the output value for the GPIO pin.

    Create the function SetOutput() in the ioDriver.c module. This function receives two parameters. The first parameter is the PIN number required to set its output value. The second parameter is the value to set, TRUE or FALSE (‘1’ or ‘0’).

```
void SetOutput(int signal, boolean state)
{
   uint_32  set_value=0;

}
```

12. Define the function prototype after the source line.

```
boolean InitializeIO(void);
```

    by adding the code:

```
void SetOutput(int signal,boolean state);
```

13. This function creates four arrays. One for each PIN configured when the GPIO driver was opened. Add the following code to the SetOutput() function.

```
static const uint_32 led1[] = {
    LED_1,
    GPIO_LIST_END
   };
   static const uint_32 led2[] = {
    LED_2,
    GPIO_LIST_END
   };
   static const uint_32 led3[] = {
    LED_3,
    GPIO_LIST_END
   };
   static const uint_32 led4[] = {
    LED_4,
    GPIO_LIST_END
   };
```

14. The SetOutput() function also includes a switch case that executes the ioctl function to indicate the GPIO driver to set ‘0’ or ‘1’ to the pin received in the first parameter. The ioctl function receives three parameters.

— The first parameter is the handler returned by opening the GPIO driver.

— The second parameter is the kind of operation and the value of the operation. There are defines ready for this parameter. The GPIO_IOCTL_WRITE_LOG1 define tells the ioctl to write '1.' The GPIO_IOCTL_WRITE_LOG0 define tells the ioctl to write a '0.'

— The third parameter is the array of the pin to be written. Add the following code to the SetOutput() function.

```
if (output_port) {
    switch (signal) {
      case 1:
       ioctl(output_port, (state) ? GPIO_IOCTL_WRITE_LOG1 : GPIO_IOCTL_WRITE_LOG0,
(pointer) &led1);
      break;
      case 2:
       ioctl(output_port, (state) ? GPIO_IOCTL_WRITE_LOG1 : GPIO_IOCTL_WRITE_LOG0,
(pointer) &led2);
      break;
      case 3:
       ioctl(output_port, (state) ? GPIO_IOCTL_WRITE_LOG1 : GPIO_IOCTL_WRITE_LOG0,
(pointer) &led3);
      break;
      case 4:
       ioctl(output_port, (state) ? GPIO_IOCTL_WRITE_LOG1 : GPIO_IOCTL_WRITE_LOG0,
(pointer) &led4);
      break;
    }
}
```

15. There can also be a complementary function to set all the output values to '0' within the same function. This function calls the SetOutput() function for each pin setting the value to '0.' Add the next function in the ioDriver.c module:

```
void ResetOutputs(void)
{
   uint_32 i;

   for (i=1;i<=4;i++)
   {
      SetOutput(i,FALSE);
   }

}
```

16. Define the function prototype after the source line:

```
void SetOutput(int signal,boolean state);
```

by inserting the code:

```
void ResetOutputs(void);
```

17. Save the file ioDriver.c

Refer to Appendix B, "ioDriver.c Module for TWRMCF51CN_MQX_RTOS Project," to see ioDriver.c module source code for TWRMCF51CN_MQX_RTOS project.

Refer to Appendix C, "ioDriver.c Module for M52259DEMO_MQX_RTOS Project," to see ioDriver.c module source code for M52259DEMO_MQX_RTOS project.

4.3 Create tasks.c Module

For a better organization, it is necessary to create a new module for the tasks functions.

1. Go to File menu and click on ‘New Text File.’
2. Go to File menu and click on ‘Save.’
   a) For CodeWarrior 6.2.2, in the ‘Save As…’ dialog box, look for the folder C:\myMQXprojects\TWRMCF51CN_MQX_RTOS and save the file as tasks.c.
   b) For CodeWarrior 7.1.2, in the ‘Save As…’ dialog box, look for the folder C:\myMQXprojects\M52259DEMO_MQX_RTOS and save the file as tasks.c.
3. In the project directory structure, search for the ‘Source’ folder. Right click and select “Add Files” and add the ‘tasks.c.’ When prompted with a pop up for “Add files to targets,” use the default of all targets and click OK. After adding the file, the structure directory must look like Figure 4.

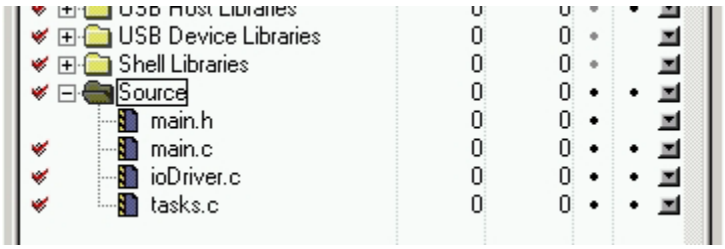

Figure 4. Adding tasks.c to Project Directory Structure

4. Open the tasks.c module and write the following includes and the init_task function.

```
#include <mqx.h>
#include <bsp.h>
#include <fio.h>

void init_task(uint_32 initial_data)
{

}
```

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

```
/* Task IDs */
#define INIT_TASK 5
#define LED1_TASK 6
#define LED2_TASK 7
#define LED3_TASK 8
#define LED4_TASK 9

void led1_task(uint_32);
void led2_task(uint_32);
void led3_task(uint_32);
void led4_task(uint_32);

extern boolean InitializeIO(void);
extern void SetOutput(int signal,boolean state);
```

6. Add the call to the function InitializeIO in the init_task function so that your code looks like the following.

```
void init_task(uint_32 initial_data)
```

```
{
    printf("\n Initialize IO \n");
    InitializeIO();
}
```

7. After the init_task called, the InitializeIO function, it is possible to start creating 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 Initialize IO \n");
    InitializeIO();
    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("\n LED1_TASK created \n");
    }
}
```

8. The previous code can be reused for the code that creates the other three tasks. Change only the second parameter of the _task_create() function with following defines:

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

NOTE

The printable text from the 2 x printf's must also be incremented.

9. Write the tasks code. The four LED tasks are similar. The only difference is the time delay for which each task waits to set the value in the respective pin it has assigned. The value is toggled from '1' to '0' and vice versa. The following is the code of the led1_task that can be included in the tasks.c module:

```
/*TASK*-------------------------------------------------------
*
* Task Name    : led1_task
* Comments     :
*    This task toggles the LED1 every 1111 milliseconds
*
*END*-------------------------------------------------------*/
void led1_task(uint_32 initial_data)
{
    int value = 0;
    printf("\n led1 task \n");

    while (TRUE)
    {
        _time_delay(1111);
```

```
            SetOutput(1,value);
            value = value^1;
        }
    }
```

10. Create the other three functions. The name of each function, delay, and the first parameter must be different in the SerOutput().

NOTE

The ... denotes the same source lines are used as in led1_task() except that the printrf text must be changed to the relevant task number.

```
void led2_task(uint_32 initial_data)
{
    …
    _time_delay(2222);
    SetOutput(2,value);
    …
}
…
void led3_task(uint_32 initial_data)
{
    …
    _time_delay(3333);
    SetOutput(3,value);
    …
}
…
void led4_task(uint_32 initial_data)
{
    …
    _time_delay(4444);
    SetOutput(4,value);
    …
}
```

11. Save tasks.c

Refer to Appendix D, “tasks.c Module,” to see tasks.c module source code for details.

See the Targeting_ColdFire.pdf in the CodeWarrior help directory to complete the remaining steps:

1. Build your project
2. Flash your project to the board
3. Run your code

5 Task Aware Debugging (TAD) Tool

This section only applies for CodeWarrior 7.1.2 Professional Edition. The Task Aware Debugging (TAD) tool is a useful and powerful instrument to debug the MQX application. The TAD is only available on CodeWarrior Professional Edition. Special and Standard editions can create and run an MQX application, but can not run the TAD.

To use the TAD:

1. Debug the application in CodeWarrior using the  debug button.

2. To stop the application, use a breakpoint or use the  break button:
3. Once the application breaks, the menu enables two options: MQX and RTCS.

   The MQX menu shows information about the core and tasks running at that moment in MQX, information like 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 5 shows both MQX and RTCS menus.

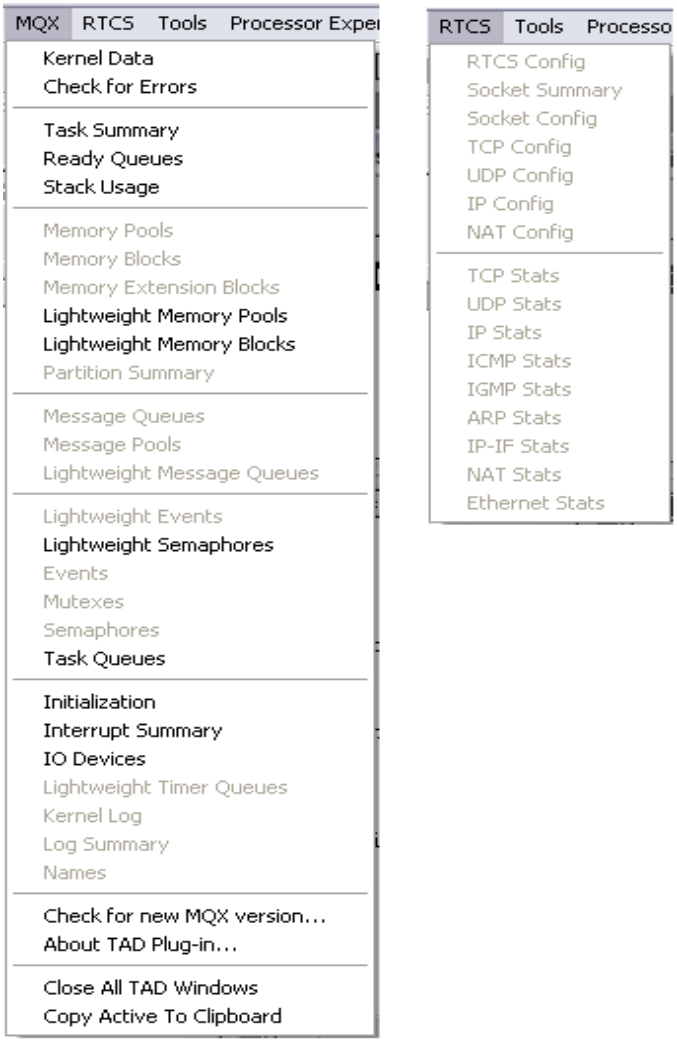

Figure 5. MQX and RTCS menus for the TAD

4. Go to the MQX menu and click on ‘Task Summary’ option. Figure 6 shows all tasks in the MQX. It shows the priority, state, and the task error if any.

Task Summary

Task Name	Task ID	TD	Priority	State	Task Error Code
init	0x10001	0x20001084	9	Terminating	OK (0x0)
led1	0x10002	0x20001770	10	Time delay blocked	OK (0x0)
led2	0x10003	0x20001e08	11	Time delay blocked	OK (0x0)
led3	0x10004	0x200024a0	12	Time delay blocked	OK (0x0)
led4	0x10005	0x20002b38	13	Time delay blocked	OK (0x0)

Figure 6. Task Summary from TAD

5. Go to the MQX menu and click on ‘Stack Usage’ option. Figure 7 shows the memory used by each task within its stack. It also shows if an overflow happens.

Stack Usage Summary

Task	Stack Base	Stack Limit	Stack Used	% Used	Overflow?
init	0x2000170c	0x20001130	0x20001564	28 %	No
led1	0x20001df8	0x2000181c	0x20001c50	28 %	No
led2	0x20002490	0x20001eb4	0x2000232c	23 %	No
led3	0x20002b28	0x2000254c	0x200029c4	23 %	No
led4	0x200031c0	0x20002be4	0x2000305c	23 %	No
Interrupt	0x20000884	0x20000484	0x200007e4	15 %	No

Figure 7. Stack Usage Summary from TAD

Appendix A main.c Module

```
/**************************************************************************
*
*   This file contains MQX only stationery code.
*
**************************************************************************/


#include <mqx.h>
#include <bsp.h>
#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[] =
{
            {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(uint_32 initial_data)
{
   printf("\n Hello World \n");
   _mqx_exit(0);
}
*/
/* EOF */
```

Appendix B ioDriver.c Module for TWRMCF51CN_MQX_RTOS Project

```
#include <mqx.h>
#include <bsp.h>
#include <fio.h>
#include <io_gpio.h>

static FILE_PTR output_port=NULL;

#define LED_1 (GPIO_PORT_TE | GPIO_PIN3)
#define LED_2 (GPIO_PORT_TG | GPIO_PIN5)
#define LED_3 (GPIO_PORT_TE | GPIO_PIN5)
#define LED_4 (GPIO_PORT_TH | GPIO_PIN3)

boolean InitializeIO(void);
void SetOutput(int signal,boolean state);
void ResetOutputs(void);

boolean InitializeIO(void)
{

            const uint_32 output_set[] =
            {
                        LED_1 | GPIO_PIN_STATUS_0,
                        LED_2 | GPIO_PIN_STATUS_0,
                        LED_3 | GPIO_PIN_STATUS_0,
                        LED_4 | GPIO_PIN_STATUS_0,
                        GPIO_LIST_END
            };

            /* Open and set port TC as output to drive LEDs */
            output_port = fopen("gpio:write", (char_ptr) &output_set);

            if (output_port)
                        {
                        ResetOutputs();
                        }

                        return (output_port!=NULL);

}

void SetOutput(int signal, boolean state)
{
            uint_32 set_value=0;

            static const uint_32 led1[] = {
            LED_1,
            GPIO_LIST_END
            };
```

```
            static const uint_32 led2[] = {
            LED_2,
            GPIO_LIST_END
            };

            static const uint_32 led3[] = {
            LED_3,
            GPIO_LIST_END
            };

            static const uint_32 led4[] = {
            LED_4,
            GPIO_LIST_END
            };
if (output_port) {
            switch (signal) {

                        case 1:
                                    ioctl(output_port, (state) ?
GPIO_IOCTL_WRITE_LOG1 : GPIO_IOCTL_WRITE_LOG0,
(pointer) &led1);
                        break;

                        case 2:
                                    ioctl(output_port, (state) ?
GPIO_IOCTL_WRITE_LOG1 : GPIO_IOCTL_WRITE_LOG0,
(pointer) &led2);
                        break;

                        case 3:
                                    ioctl(output_port, (state) ?
GPIO_IOCTL_WRITE_LOG1 : GPIO_IOCTL_WRITE_LOG0,
(pointer) &led3);
                        break;

                        case 4:
                                    ioctl(output_port, (state) ?
GPIO_IOCTL_WRITE_LOG1 : GPIO_IOCTL_WRITE_LOG0,
(pointer) &led4);
                        break;
                        }
            }
}

void ResetOutputs(void)
{
            uint_32 i;

            for (i=1;i<=4;i++)
            {
                        SetOutput(i,FALSE);
            }
}
```

Appendix C ioDriver.c Module for M52259DEMO_MQX_RTOS Project

```
#include <mqx.h>
#include <bsp.h>
#include <fio.h>
#include <io_gpio.h>

static FILE_PTR output_port=NULL;

#define LED_1 (GPIO_PORT_TC | GPIO_PIN0)
#define LED_2 (GPIO_PORT_TC | GPIO_PIN1)
#define LED_3 (GPIO_PORT_TC | GPIO_PIN2)
#define LED_4 (GPIO_PORT_TC | GPIO_PIN3)

boolean InitializeIO(void);
void SetOutput(int signal,boolean state);
void ResetOutputs(void);

boolean InitializeIO(void)
{

            const uint_32 output_set[] =
            {
                        LED_1 | GPIO_PIN_STATUS_0,
                        LED_2 | GPIO_PIN_STATUS_0,
                        LED_3 | GPIO_PIN_STATUS_0,
                        LED_4 | GPIO_PIN_STATUS_0,
                        GPIO_LIST_END
            };

            /* Open and set port TC as output to drive LEDs */
            output_port = fopen("gpio:write", (char_ptr) &output_set);

            if (output_port)
                        {
                        ResetOutputs();
                        }

                        return (output_port!=NULL);

}

void SetOutput(int signal, boolean state)
{
            uint_32 set_value=0;

            static const uint_32 led1[] = {
            LED_1,
            GPIO_LIST_END
            };
```

```
            static const uint_32 led2[] = {
            LED_2,
            GPIO_LIST_END
            };

            static const uint_32 led3[] = {
            LED_3,
            GPIO_LIST_END
            };

            static const uint_32 led4[] = {
            LED_4,
            GPIO_LIST_END
            };
if (output_port) {
            switch (signal) {

                        case 1:
                                    ioctl(output_port, (state) ?
GPIO_IOCTL_WRITE_LOG1 : GPIO_IOCTL_WRITE_LOG0,
(pointer) &led1);
                        break;

                        case 2:
                                    ioctl(output_port, (state) ?
GPIO_IOCTL_WRITE_LOG1 : GPIO_IOCTL_WRITE_LOG0,
(pointer) &led2);
                        break;

                        case 3:
                                    ioctl(output_port, (state) ?
GPIO_IOCTL_WRITE_LOG1 : GPIO_IOCTL_WRITE_LOG0,
(pointer) &led3);
                        break;

                        case 4:
                                    ioctl(output_port, (state) ?
GPIO_IOCTL_WRITE_LOG1 : GPIO_IOCTL_WRITE_LOG0,
(pointer) &led4);
                        break;
                        }
            }
}

void ResetOutputs(void)
{
            uint_32 i;

            for (i=1;i<=4;i++)
            {
                        SetOutput(i,FALSE);
            }
}
```

Appendix D tasks.c Module

```
#define LED1_TASK 6
#define LED2_TASK 7
#define LED3_TASK 8
#define LED4_TASK 9

void init_task(uint_32);
void led1_task(uint_32);
void led2_task(uint_32);
void led3_task(uint_32);
void led4_task(uint_32);

extern boolean InitializeIO(void);
extern void SetOutput(int signal,boolean state);

void init_task(uint_32 initial_data)
{
            _task_id task_id;
            printf("\n Initialize IO \n");
            InitializeIO();

            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(uint_32 initial_data)
{
            int value = 0;
            printf("\n led1 task \n");

            while (TRUE)
            {
                        _time_delay(1111);
                        SetOutput(1,value);
                        value = value^1;
            }
}

void led2_task(uint_32 initial_data)
{
            int value = 0;
            printf("\n led2 task \n");

            while (TRUE)
            {
                        _time_delay(2222);
                        SetOutput(2,value);
                        value = value^1;
            }
}

void led3_task(uint_32 initial_data)
{
            int value = 0;
            printf("\n led3 task \n");

            while (TRUE)
            {
```

```
                        _time_delay(3333);
                        SetOutput(3,value);
                        value = value^1;
            }
}

void led4_task(uint_32 initial_data)
{
            int value = 0;
            printf("\n led4 task \n");

            while (TRUE)
            {
                        _time_delay(4444);
                        SetOutput(4,value);
                        value = value^1;
            }
}
```

Appendix E main.h Module

```
#ifndef __main_h_
#define __main_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(uint_32);
extern void led1_task(uint_32);
extern void led2_task(uint_32);
extern void led3_task(uint_32);
extern void led4_task(uint_32);


#endif /* __main_h_ */
```

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 China Ltd.
Exchange Building 23F
No. 118 Jianguo Road
Chaoyang District
Beijing 100022
China
+86 10 5879 8000
support.asia@freescale.com

For Literature Requests Only:
Freescale Semiconductor Literature Distribution Center
1-800-441-2447 or 303-675-2140
Fax: 303-675-2150
LDCForFreescaleSemiconductor@hibbertgroup.com

Document Number: AN3905
Rev. 1
11/2009

freescale™
semiconductor