

How to Develop a Project With TI SYS/BIOS

Jeanne Yi

MCU SAE Team

ABSTRACT

TI SYS/BIOS is a real-time operating system kernel. It is also a component of TI RTOS. TI RTOS is an integration system; it includes TI SYS/BIOS kernel, XDCtools, middleware, MCU driver library, and other kinds of components. TI-RTOS is a one-stop solution for developing applications for TI embedded processors and is tightly integrated with TI's Code Composer Studio™ (CCS) development environment. TI-RTOS also provides many example project packages with which you may start to develop a project with TI SYS/BIOS.

Contents

Overview	3
1 Preparation	4
1.1 System Requirements	4
1.2 Download the Installation Packages	4
1.3 Installing CCS 5.1.1.00031, SYS/BIOS 6.33.4.39, and XDCtools 3.23.4.27	4
2 Create Your First SYS/BIOS Project.....	5
2.1 Startup Code Composer Studio	5
2.2 Create a SYS/BIOS CCS Project.....	5
2.3 Link the Memory Map File for the Project.....	7
2.4 Build the Project	9
2.5 Debug the Project.....	10
2.6 For More Reference.....	11
3 Use of the Basic Components for a SYS/BIOS Project.....	11
3.1 Use .cfg File to Configure the System.....	11
3.2 Entrance of the System	13
3.3 Dynamically or Statically Create the Task.....	14
3.3.1 Dynamically Creating the Task.....	14
3.3.2 Statically Creating the Task.....	14
3.4 Usage of the Semaphores	16
3.5 Use of the Mailboxes	17
3.5.1 Introduction About the Mailboxes	17
3.5.2 Use Mailboxes in Your Project	17
3.5.3 Often-used SYS/BIOS API Function for Mailbox	18
3.6 Use of the Hwi Tasks.....	19
3.6.1 Introduction to the Hwi.....	19
3.6.2 Add Hwi Module in Your Project	19
3.6.3 An Hwi Instance Example Based on Concerto™ F28M35x	20
References.....	23

Figures

Figure 1. Folders in the Directory c:\ti\.....	4
Figure 2. CCS 5.1 Welcome Display	5
Figure 3. CCS Workspace Launcher.....	5
Figure 4. New CCS Project	6
Figure 5. RTSC Configuration Settings for the SYS/BIOS Project.....	7
Figure 6. General Setting for the Project.....	8
Figure 7. CCS Project Explorer Window.....	9
Figure 8. CCS Console Window	10
Figure 9. CCS Problems Window.....	10
Figure 10. ccxml File Edit Window.....	10
Figure 11. CCS Debug Window.....	11
Figure 12. XDCtools GUI Window (SYS/BIOS System Component Overview)	12
Figure 13. XDCtools GUI Window (SYS/BIOS Runtime Options).....	12
Figure 14. XDCtools GUI Window (Task Thread Management).....	15
Figure 15. XDCtools GUI Window (Script Source Code for Task)	15
Figure 16. XDCtools GUI Window (Semaphore Management)	17
Figure 17. XDCtools GUI Window (Script Source Code for Semaphore)	17
Figure 18. Selection of Synchronization Modules	17
Figure 19. XDCtools GUI Window (Mailbox Module Settings).....	18
Figure 20. XDCtools GUI Window (Mailbox Instance Settings).....	18
Figure 21. XDCtools GUI Window (Hardware Interrupt Module Settings)	20
Figure 22. XDCtools GUI Window (Hardware Interrupt Instance Settings)	21
Figure 23. Interrupt 0 to 91 Priority (PRI0 to PRI22) Registers.....	22

Tables

Table 1. Devices Supported by TI-RTOS and TI SYS/BIOS	3
Table 2. Interrupts from NVIC to Cortex-M3	21
Table 3. Peripherals Register Map.....	22
Table 4. Interrupt 0 to 91 Priority (PRI0 to PRI22) Registers Field Descriptions	22

Overview

TI-RTOS is a real-time operating system for TI devices. The first released package of TI-RTOS was MCU-SDK V1.00.00.68. The most recent released package is TI-RTOS 1.21.00.19. TI-RTOS facilitates the development of applications for TI microcontrollers. This product contains several software components and examples. The components include

- SYS/BIOS
- IPC
- MWare
- NDK
- StellarisWare
- UIA
- XDCtools

The examples within TI-RTOS have used the software components together.

TI SYS/BIOS is the real-time OS kernel of TI-RTOS. TI SYS/BIOS is a scalable real-time kernel designed for applications that require real-time scheduling and synchronization or real-time instrumentation. TI SYS/BIOS provides preemptive multithreading, hardware abstraction, real-time analysis, and configuration tools. TI SYS/BIOS is designed to minimize memory and CPU requirements on the target. It must be used together with the proper version of XDCtools.

The devices supported by SYS/BIOS include MSP430™ MCUs, F28M35X, TM4C MCUs, and AM335X. The devices supported by TI-RTOS only include F28M35X and TM4C MCUs. Table 1 lists the devices supported by TI-RTOS and TI SYS/BIOS. (For more details, see on the website <http://www.ti.com/lsds/ti/tools-software/rtos.page#support>.)

Table 1. Devices Supported by TI-RTOS and TI SYS/BIOS

	MSP430 MCUs	F28M35x	TM4C	AM335x
TI-RTOS		Yes	Yes	
SYS/BIOS kernel	Yes	Yes	Yes	Yes

This document primarily discusses how to use the basic elements of SYS/BIOS, such as tasks, semaphores, mailboxes, Hwi, and so forth, together with the CCS and XDCtools to develop a project. SYS/BIOS 6.33.4.39 is the version we used. We used the F28M35x device and we worked only with its Cortex™-M3 MCU core.

For more training detail about SYS/BIOS, download the reference resource from www.ti.com. If you have questions when developing your project, we strongly recommend that you raise your questions on the BIOS forum of TI's E2E Community:

<http://e2e.ti.com/support/embedded/bios/f/355.aspx> or on the Chinese technical support forum <http://www.devisupport.com>.

1 Preparation

1.1 System Requirements

We used is Windows 7™, but Windows Vista or Windows XP (SP2 or SP3) is acceptable. To install CCS and TI-RTOS, at least 4GB of free disk space are required.

1.2 Download the Installation Packages

To use SYS/BIOS, you must install TI CCS on your PC. To support SYS/BIOS 6.33.4.39, you must install the CCS 5.1.1.00031 or higher version. You can download the CCS installation packages from the website http://processors.wiki.ti.com/index.php/Download_CCS.

Download SYS/BIOS installation package from the website http://software-dl.ti.com/dsp/dsp_sw/sdo_sb/targetcontent/bios/sysbios/6_33_04_39/index_FDS.html. SYS/BIOS 6.33.4.39 is recommended for use with XDCtools 3.23.4.27, so you should download the XDCtools installation package from the website http://software-dl.ti.com/dsp/dsp_sw/sdo_sb/targetcontent/rtsc/3_23_02_47/index_FDS.html. Choose and download the correct version of the package for your PC environment.

If you want to use TI-RTOS package for your devices, download the CCS 5.3.0.00039 or higher version. You could download the TI-RTOS package from the website http://software-dl.ti.com/dsp/dsp_sw/sdo_sb/targetcontent/tirtos/index.html.

1.3 Installing CCS 5.1.1.00031, SYS/BIOS 6.33.4.39, and XDCtools 3.23.4.27

Perform the following steps:

1. Install CCS and install it in the default installation directory of c:\ti. When selecting components, do not choose any SYS/BIOS components.
2. Install the SYS/BIOS also in the default installation directory of c:\ti.
3. Install the XDCtools in the default installation directory of c:\ti.

When the installation completes successfully, there are three folders in the directory of c:\ti, as shown in Figure 1.

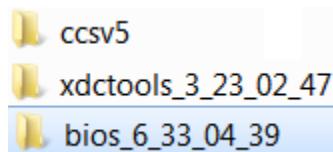


Figure 1. Folders in the Directory c:\ti

2 Create Your First SYS/BIOS Project

2.1 Startup Code Composer Studio

Open Code Composer Studio v5 from the start menu or your desktop shortcut (see Figure 2). After a few seconds of the welcome display, the Workspace Launcher is displayed (see Figure 3). Input your desired workspace path and click OK.



Figure 2. CCS 5.1 Welcome Display

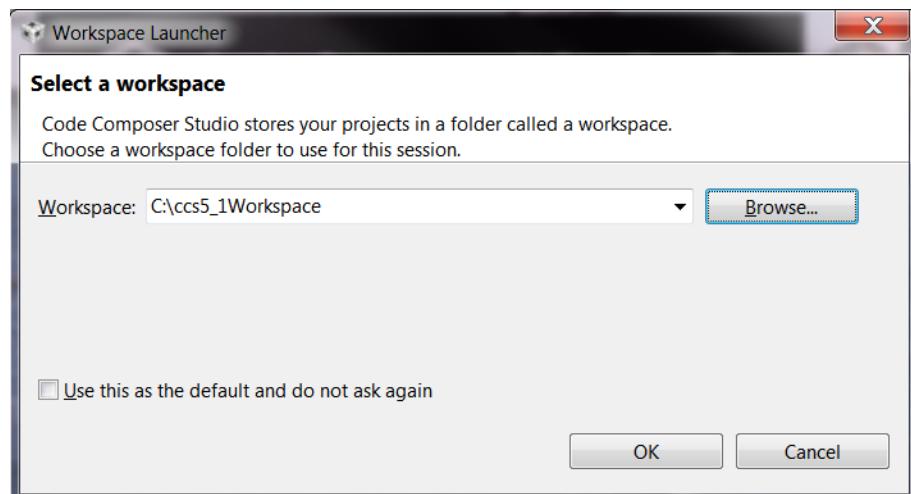


Figure 3. CCS Workspace Launcher

2.2 Create a SYS/BIOS CCS Project

1. In the CCS menus, select File→New→CCS Project. The New CCS Project dialog box is displayed.

2. Choose the device family that is supported by SYS/BIOS; Figure 4 shows the SYS/BIOS project templates list. You can choose a template from this list as the template of your project. We choose a Task Mutex example.

NOTE: It is best to define a new location for your project; otherwise, an uncomfortable situation may arise later.

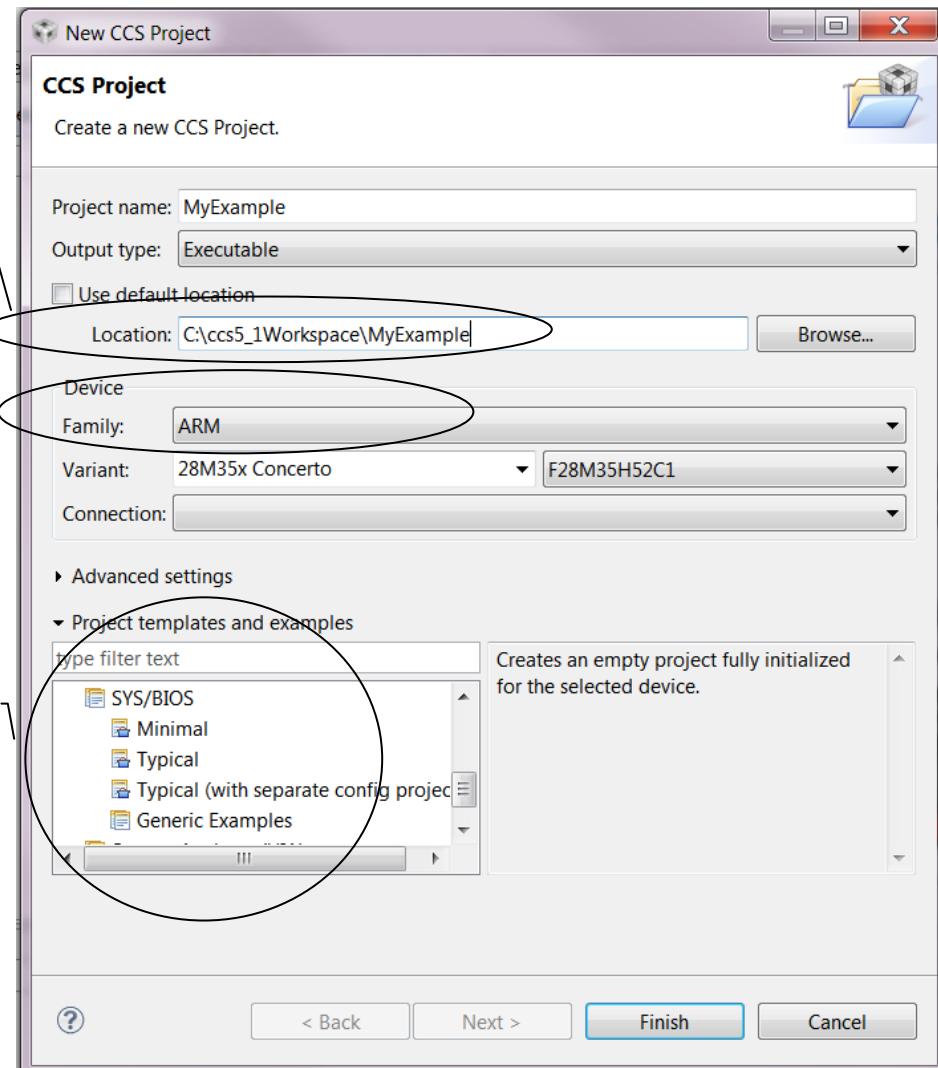


Figure 4. New CCS Project

3. When all items are entered, the [Next>] button is enabled. Click the [Next>] button; the RTSC Configuration Settings are displayed, as shown in Figure 5.
4. Select the correct version for XDCtools and SYS/BIOS components shown in the figure.
5. Click the [Finish] button; the code files mutex.c and mutex.cfg for this project will be generated automatically.

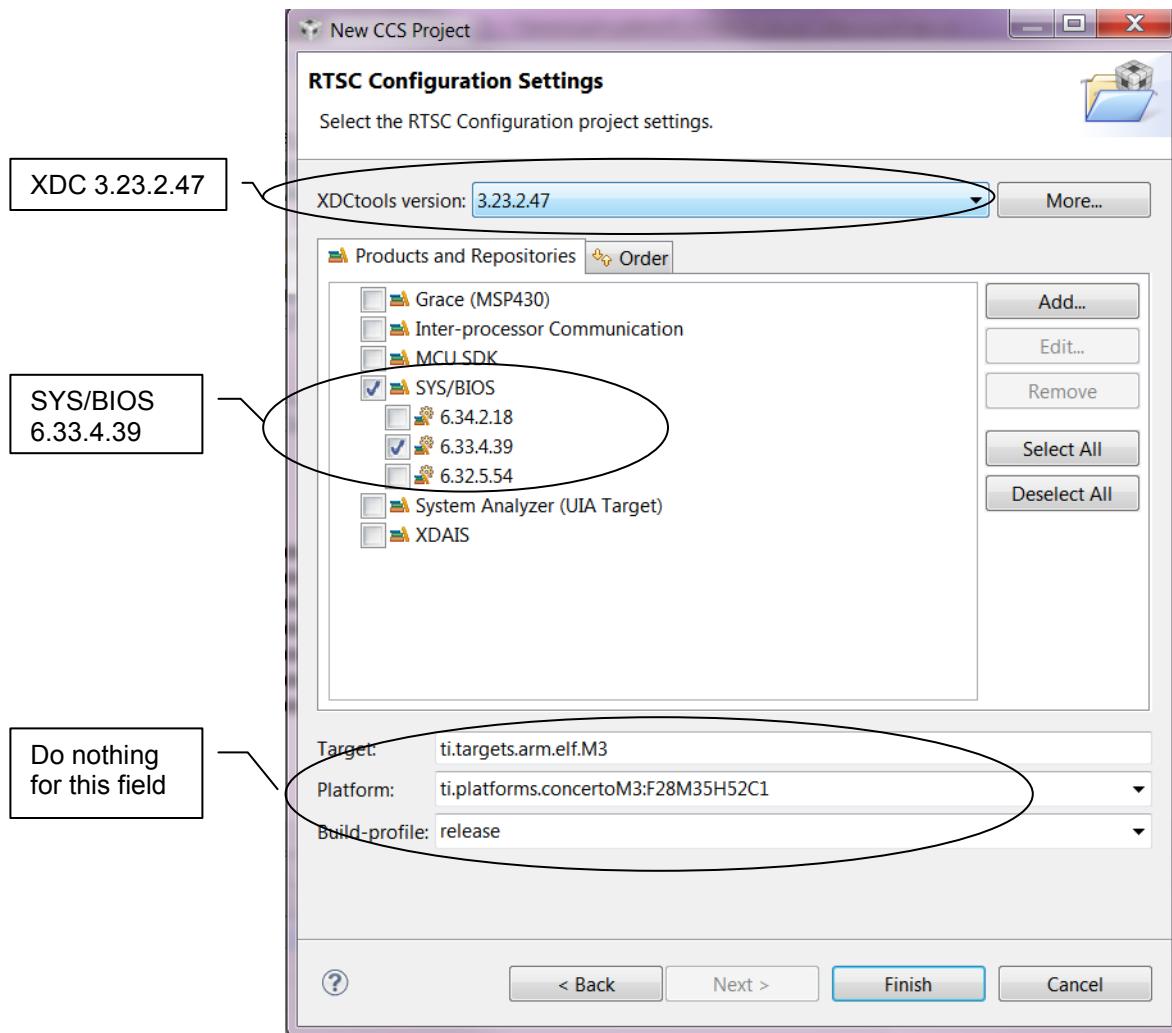


Figure 5. RTSC Configuration Settings for the SYS/BIOS Project

2.3 Link the Memory Map File for the Project

1. Now the project and the files should be listed in the Project Explorer. If you cannot see the Project Explorer, open it by clicking the tool menu of CCS View→Project Explorer.
2. Right-click the project name, select Properties, and click General in the left block to open the General settings page (see Figure 6).
3. Choose the emulator connection method from the connection pulldown list.
4. Click the [Browse...] button to the right of the Linker command file: setting line and find the correct link file F28M35H52C1.cmd in the SYS/BIOS installation directory "C:\ti\bios_6_33_04_39\packages\ti\sysbios\examples\m3\include".
5. After making these settings, click the [Apply] button and close the dialog box. Two more files, F28M35H52C1.cmd and F28M35H52C1.ccxml., are automatically added into the project files list. The F28M35H52C1.cmd file is the memory map file while the F28M35H52C1.ccxml file is the target configuration file.

6. To customize the memory map for your project, simply double-click the F28M35H52C1.cmd file to open it. Then you can edit this file according to your memory arrangement.

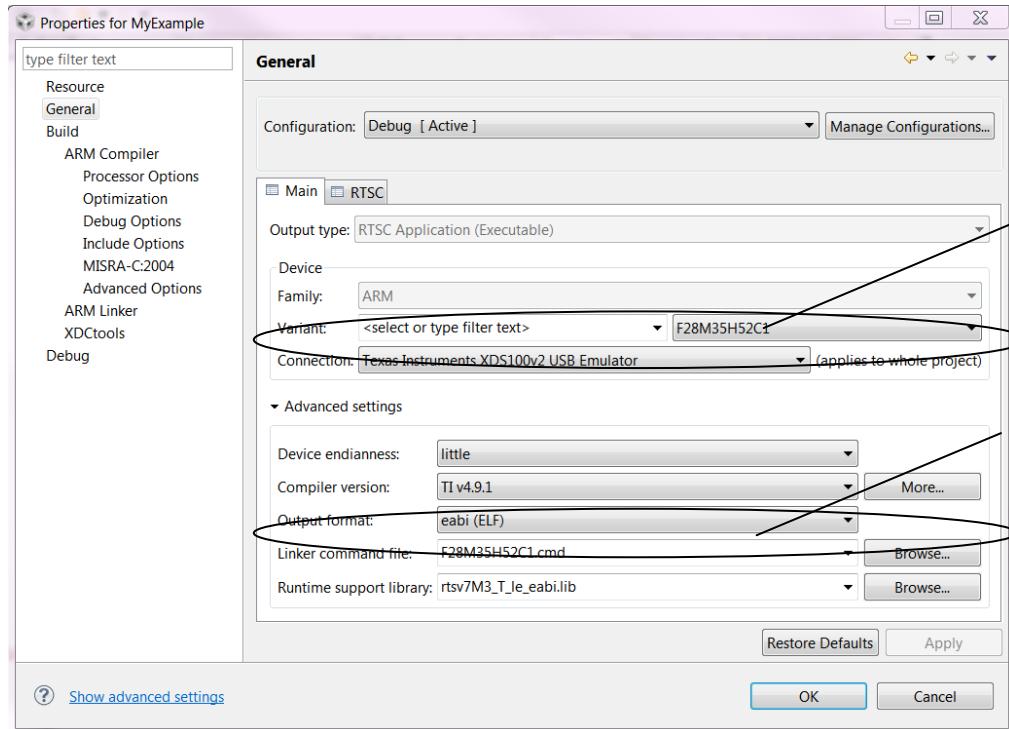


Figure 6. General Setting for the Project

The contents of the F28M35H52C1.cmd follow:

```
MEMORY
{
    FLASH_BOOT (RWX): origin = 0x200030, length = 0x4
    FLASH (RWX) : origin = 0x200034, length = 0x7FF9C
    C03SRAM (RWX) : origin = 0x20000000, length = 0x8000
    S07SHRAM (RWX) : origin = 0x20008000, length = 0x10000
    CTOMMSGRAM (R) : origin = 0x2007F000, length = 0x800
    MTOCMSGRAM (RW) : origin = 0x2007F800, length = 0x800
}

SECTIONS
{
    /* Allocate program areas: */
    .text      : > FLASH
    .cinit     : > FLASH
    .pinit     : > FLASH
    .init_array : > FLASH

    /* Initialized sections go in Flash */
    .const     : > FLASH

    /* Allocate uninitialized data sections: */
    .data      : > C03SRAM
    .bss       : > C03SRAM
    .sysmem   : > C03SRAM
    .stack    : > C03SRAM
```

```

.cio      : > C03SRAM
.neardata : > C03SRAM
.rodata   : > C03SRAM
.args     : > C03SRAM
}

__STACK_TOP = __stack + 256;

```

You can modify this file to customize the memory map for your device.

2.4 Build the Project

1. Open the Project Explorer window.
2. Right-click on the project name and select the Build Project command in the popup menu (see Figure 7).

The build message is displayed in the Console window (see Figure 8) and the detailed error and warning message is displayed in the Problems window (see Figure 9). If the building is done successfully, an .out program file is generated in the related folder.

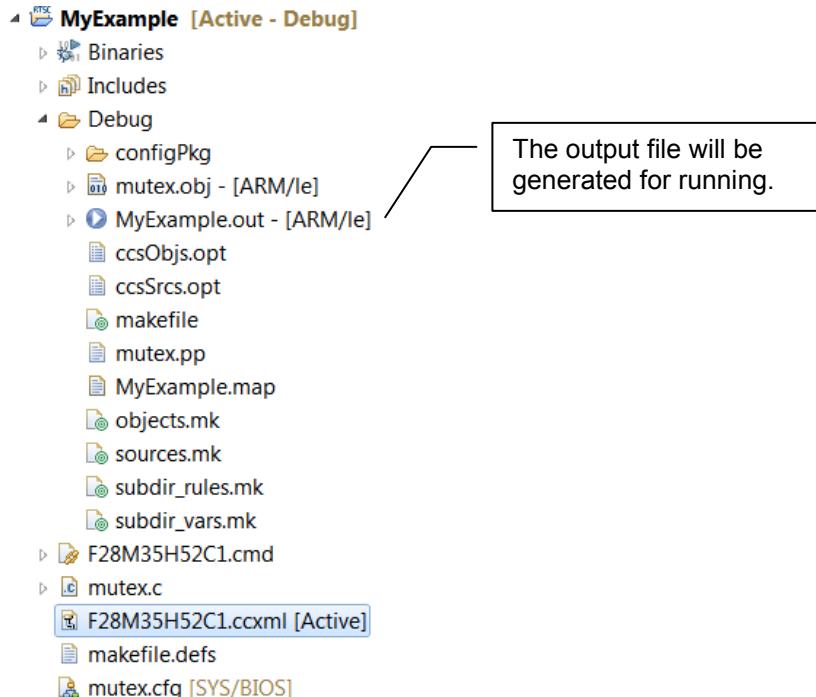
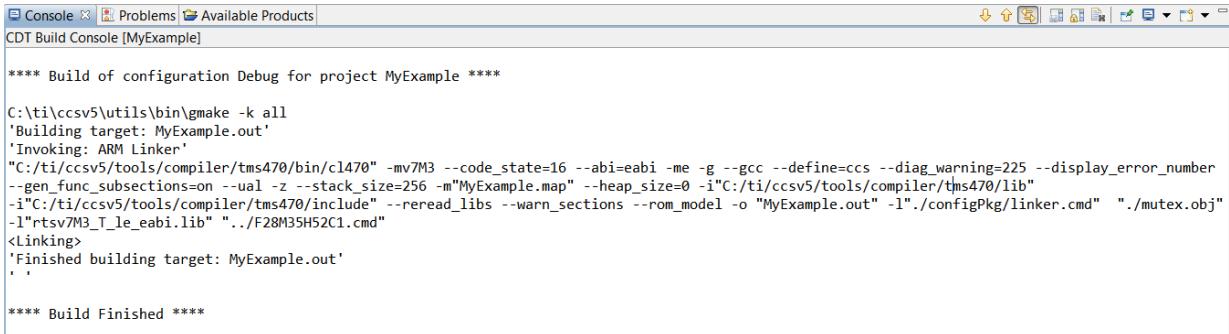


Figure 7. CCS Project Explorer Window



```
**** Build of configuration Debug for project MyExample ****

C:\ti\ccsv5\utils\bin\gmake -k all
'Building target: MyExample.out'
'Invoking: ARM Linker'
"C:\ti\ccsv5\tools\compiler\tms470\bin\cl470" -mv7M3 --code_state=16 --abi=eabi -me g --gcc --define=ccs --diag_warning=225 --display_error_number
--gen_func_subsections=on --unal -z --stack_size=256 -m"MyExample.map" --heap_size=0 -i"C:\ti\ccsv5\tools\compiler\tms470\lib"
-i"C:\ti\ccsv5\tools\compiler\tms470\include" --reread_libs --warn_sections --rom_model o "MyExample.out" -l"./configPkg/linker.cmd" "./mutex.obj"
-l"rtsv7M3_T_le_eabi.lib" "../F28M35H52C1.cmd"
<Linking>
'Finished building target: MyExample.out'
'

**** Build Finished ****
```

Figure 8. CCS Console Window

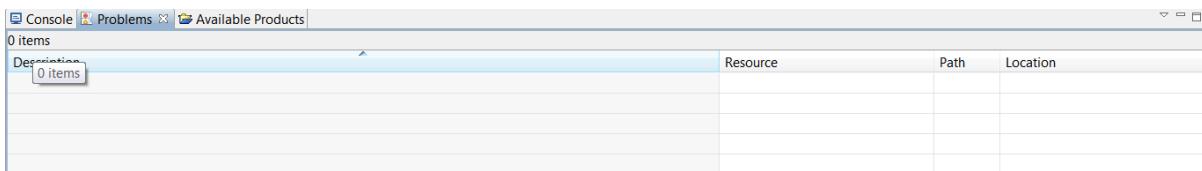


Figure 9. CCS Problems Window

2.5 Debug the Project

The F28M35H52C1.ccxml file is a Target Configuration file. By double-clicking on the file name in the Project Explorer window, you can get the UI display for this file (see Figure 10). Before downloading the .out file to the device, you can test the connection of the emulator by clicking the [Test Connection] button.

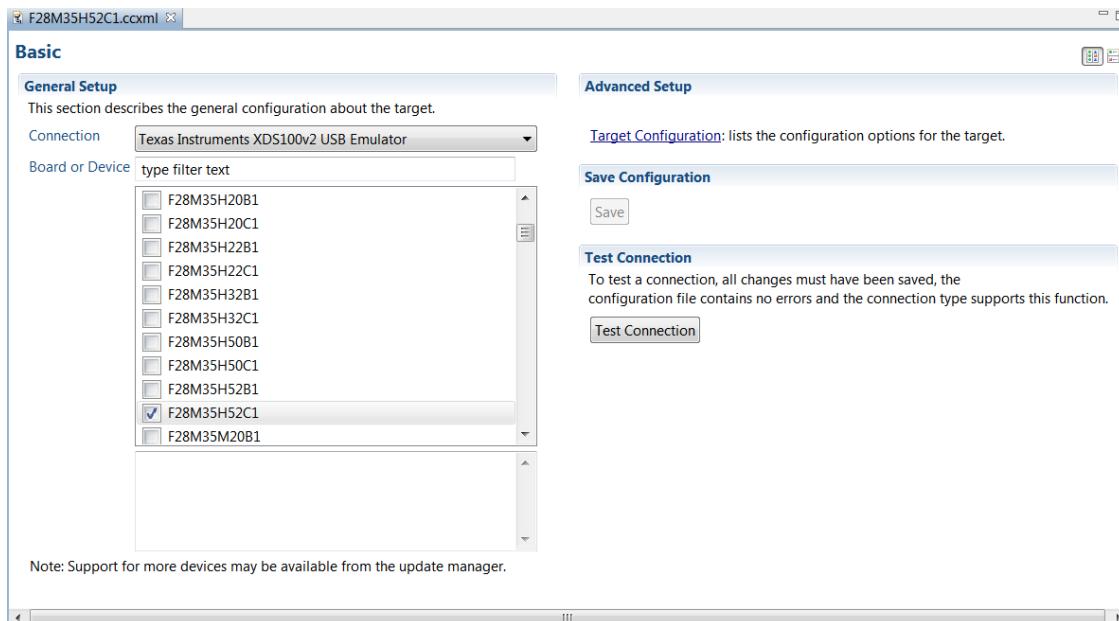


Figure 10. ccxml File Edit Window

1. Connect one JTAG emulator with the PC.

2. Open the Target Configuration window.
3. Right-click the ccxml file and choose Launch Selected Configuration to open the CCS Debug window.
4. Right-click the device core name and choose Connect Target from the popup menu to connect CCS with the emulator.
5. After the connection is made, download the .out program file to the target emulator, and then begin debugging the program.

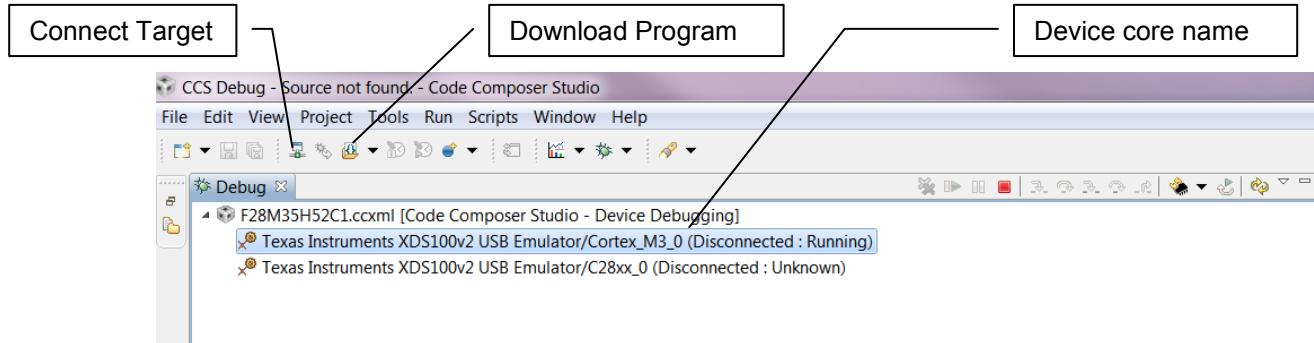


Figure 11. CCS Debug Window

2.6 For More Reference

For more details about project building and debugging, refer the document TI SYS/BIOS Real-time Operating System v6.x User's Guide (<http://www.ti.com/lit/ug/spruex3m/spruex3m.pdf>).

3 Use of the Basic Components for a SYS/BIOS Project

3.1 Use .cfg File to Configure the System

You can configure SYS/BIOS applications by modifying the *.cfg configuration file in the project. This file is written in the XDCscript language, which is a superset of JavaScript. CCS provides a graphical configuration editor called XGCONF. You can edit the file by using the XGCONF GUI interface. XGCONF is the default editor for files listed in the Project Explorer window. With the GUI editor, it is easier to view and configure the system components. You can choose the required components simply by clicking the icon, or you can configure the parameters by choosing the related options.

For details about how to configure the parameters for your project with the XDC GUI editor, refer the TI SYS/BIOS user guide (SPRUEX3M <http://www.ti.com/lit/ug/spruex3m/spruex3m.pdf>). If you are familiar with the JavaScript syntax, you can also edit the file with a text editor.

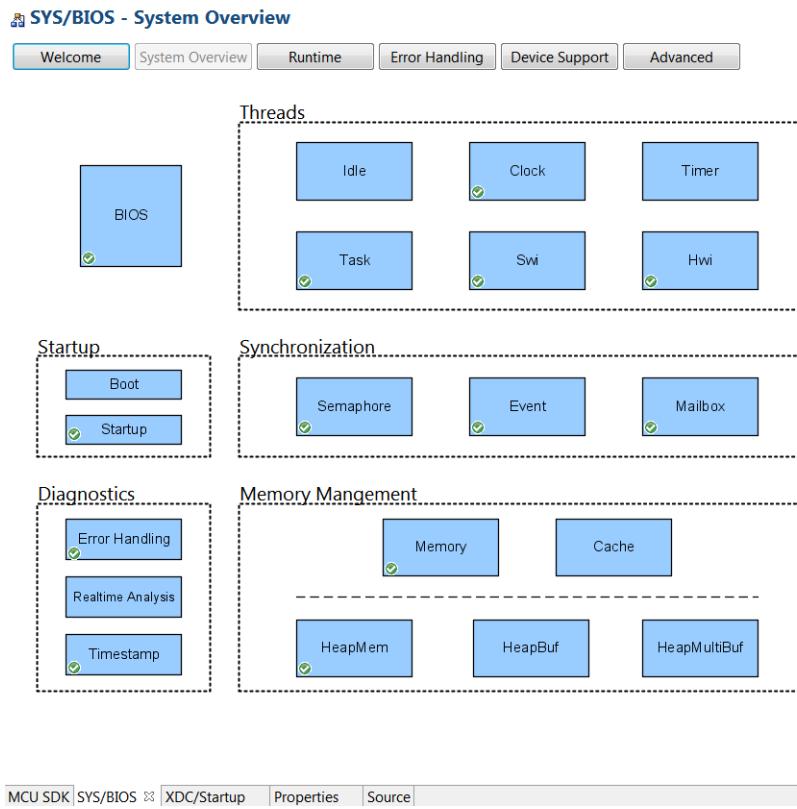


Figure 12. XDCtools GUI Window (SYS/BIOS System Component Overview)

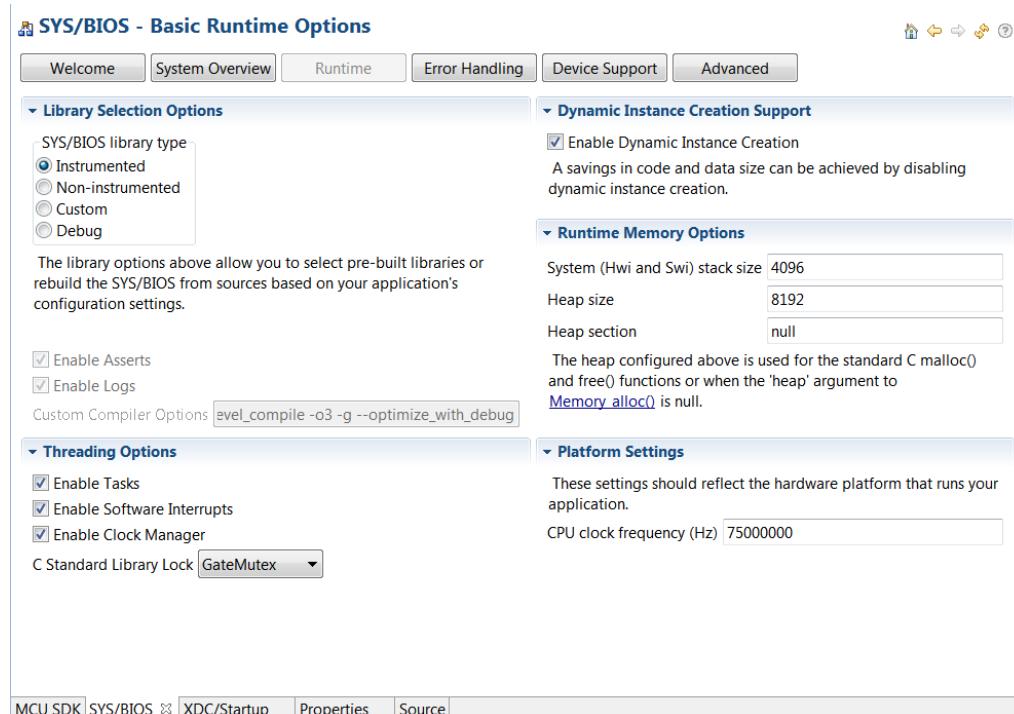


Figure 13. XDCtools GUI Window (SYS/BIOS Runtime Options)

3.2 Entrance of the System

Like most systems, the entrance of the system is the function of main(). But the real entrance of SYS/BIOS OS is the caller of BIOS_start() at the end of main(). The initialization for the hardware of MCU, such as the GPIO port setting, peripheral interface configuration, must be done before BIOS_start() is called.

A typical example of main() follows:

```
Int main(Void)           //Entrance of the system, the return type could be any type you want.
{
    Error_Block eb;
    Task_Parms taskParams;

    /* Call board init functions. */
    Board_initGeneral();
    Board_initGPIO();
    Board_initUART();

    /* Turn on user LED */
    GPIO_write(Board_LED, Board_LED_ON);

    /* Create the task */
    Error_init(&eb);
    Task_Parms_init(&taskParams);
    taskParams.instance->name = "echo";
    echo = Task_create(echoFxn, &taskParams, &eb);
    if (echo == NULL) {
        System_printf("Task was not created\n");
        System_abort("Aborting...\n");
    }

    /* Enable interrupts and start SYS/BIOS */
    BIOS_start();           //Entrance of the SYS/BIOS

    return (0);
}
```

To ensure the project can be built successfully, generally four head files must be included in your file. These head files are:

- <xdc/std.h>
- <xdc/runtime/System.h>
- <xdc/cfg/global.h>
- <ti/sysbios/BIOS.h>

3.3 Dynamically or Statically Create the Task

The task is basically a thread unit in the SYS/BIOS system. You can put your desired action, which is not related with **to?** the interrupt, into a task. Sixteen priorities (0 through 15) were defined for the task. Because task 0 is used by the default system idle task, you can define the priority of your task from 1 to 15. The task with a priority of 0 has the lowest priority in the tasks schedule. If you want a task to be initially inactive, set its priority to -1. Such tasks are not scheduled to run until their priority is increased at runtime.

The tasks could be created dynamically or statically. Certainly, a task must be created once dynamically or statically within a project.

If you will use tasks in your project, you must include the head file of <ti/sysbios/knl/Task.h> in your code file.

3.3.1 Dynamically Creating the Task

Dynamically creating the task means the tasks are created in the .c code file. The task is created when the program calls the calling function in which the task code is created.

A typical example of creating the task dynamically follows:

```
Task_Handle taskHandle;
Task_Params taskParams;
Error_Block eb;
Task_Params_init(&taskParams);
taskParams.stackSize = 640;           //define the stack size for the task
taskParams.priority = 2;             //define the priority of the task
taskParams.instance->name = "Serial";
Error_init(&eb);
taskHandle = Task_create(SerialMain, &taskParams, &eb);
if (taskHandle == NULL) {
    System_printf("Task_create() failed!\n");
    BIOS_exit(0);
}
```

3.3.2 Statically Creating the Task

Statically creating the task means the tasks are created in the .cfg file. Once you add a task instance in the .cfg file, the task is created when the main() is run.

As described in Section 3.1, you can edit the .cfg file by the GUI tool or script text editor.

Open the Outline window and double-click the Task icon; the GUI of the Task Thread Management is displayed (see Figure 14). Tasks could be added or removed in the GUI. Also, the parameters such as Function entry, Priority, Stack size, and so forth for the task could be defined visually in the editor boxes. When you modify the value of the settings in the GUI window, the script source code is automatically generated in the source window of the .cfg file (see Figure 15).

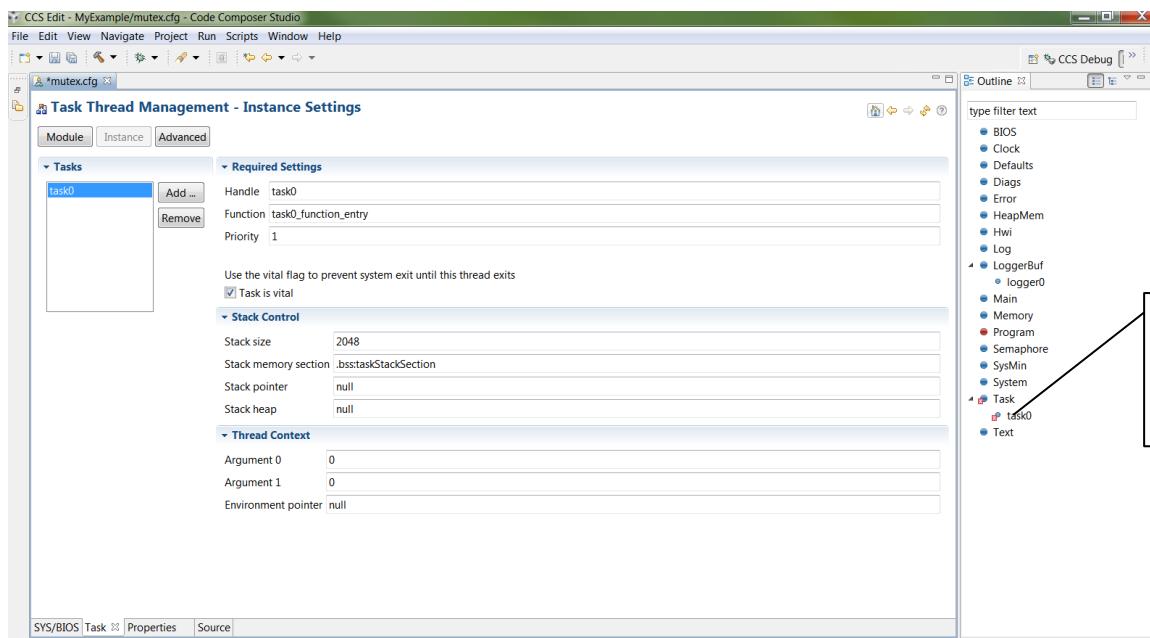


Figure 14. XDCtools GUI Window (Task Thread Management)

```

68 * Uncomment this line to disable the output of characters by SysMin
69 * when the program exits. SysMin writes characters to a circular buffer.
70 * This buffer can be viewed using the SysMin Output view in ROV.
71 SysMin.flushAtExit = false;
72 */
73
74/*
75 * The BIOS module will create the default heap for the system.
76 * Specify the size of this default heap.
77 */
78 BIOS.heapSize = 0x2000;
79
80/* System stack size (used by ISRs and Swis) */
81 Program.stack = 0x1000;
82
83/* Circular buffer size for System_printf() */
84 SysMin.bufSize = 0x400;
85
86/*
87 * Create and install logger for the whole system
88 */
89 var loggerBufParams = new LoggerBuf.Params();
90 loggerBufParams.numEntries = 32;
91 var logger0 = LoggerBuf.create(loggerBufParams);
92 Defaults.common$.logger = logger0;
93 Main.common$.diags_INFO = Diags.ALWAYS_ON;
94
95 System.SupportProxy = SysMin;
96
97 var task0Params = new Task.Params();
98 task0Params.instance.name = "task0";
99 task0Params.priority = 3;
100 task0Params.stackSize = 1024;
101 Program.global.task0 = Task.create("&task0_function_entry", task0Params);
102

```

A red oval highlights the creation of 'task0Params' in the script source code. A callout box with the text 'Script source code is generated automatically for task creation.' points to the highlighted code.

Figure 15. XDCtools GUI Window (Script Source Code for Task)

Once the task has been created statically, it is not necessary to dynamically add creation code in the .c file.

Statically creating the task is a good method because you could view the list of all of the tasks in your project in the Task Thread Management XDC GUI window. But if the task would not be lively during the system running period, it should be created dynamically.

For more detailed introduction about task thread, refer to the TI SYS/BIOS v6.35 Real-time Operating System User's Guide (SPRUEX3M <http://www.ti.com/lit/ug/spruex3m/spruex3m.pdf>).

3.4 Usage of the Semaphores

Semaphore is one of the synchronization modules in the TI SYS/BIOS system. Semaphores are often used to coordinate access to a shared resource among a set of competing tasks.

Semaphore objects can be declared as either counting or binary semaphores. For more details about semaphores, refer to the TI SYS/BIOS v6.35 Real-time Operating System User's Guide (SPRUEX3M) (<http://www.ti.com/lit/ug/spruex3m/spruex3m.pdf>).

The head file <ti/sysbios/knl/Semaphore.h> must be included in your code file if you intend to use semaphores to do synchronization between the competing tasks.

Two often-used functions for the semaphores are:

```
Bool Semaphore_pend(Semaphore_Handle sem, UInt timeout);
Void Semaphore_post(Semaphore_Handle sem);
```

You could call `Semaphore_pend(semHandle, BIOS_WAIT_FOREVER)` in a task thread to wait for the other task thread releasing the share resource by calling `Semaphore_post(semHandle)`.

Open the Outline window and double-click the Semaphore icon; the GUI of the Semaphore Management is displayed (see Figure 16). Semaphores could be added or removed by simply click the [Add] or [Remove] button in the Semaphore Management interface. The type of semaphores and the other required settings could be set by clicking the selection box or input characters in the edit box. After a semaphore is added, the related script source code is also generated automatically (see Figure 17). Handle will be the first parameter of the `Semaphore_pend()` and `Semaphore_post()` functions.

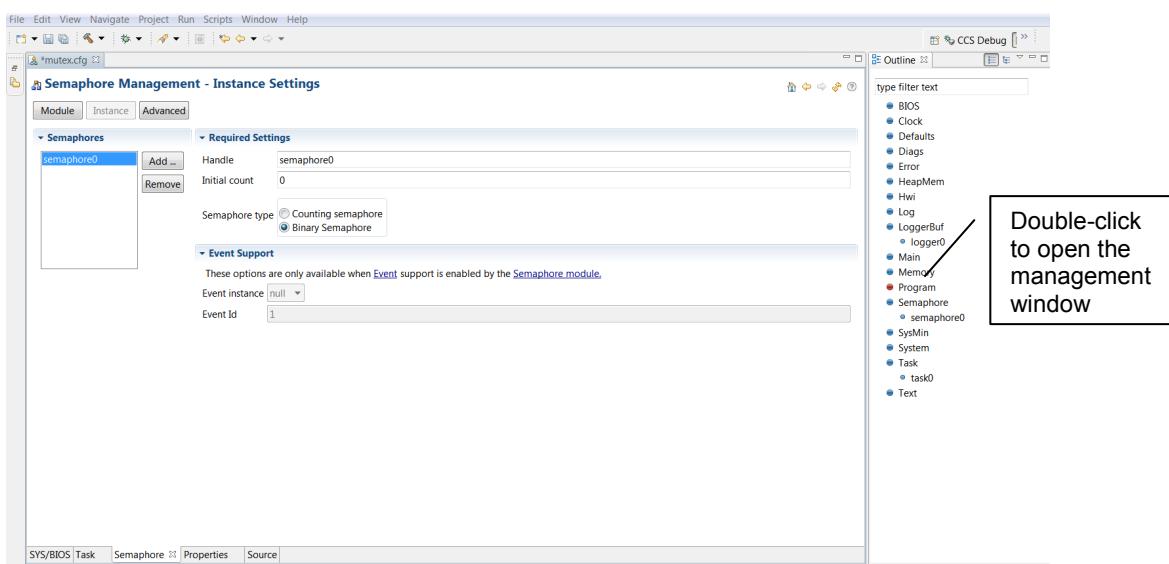
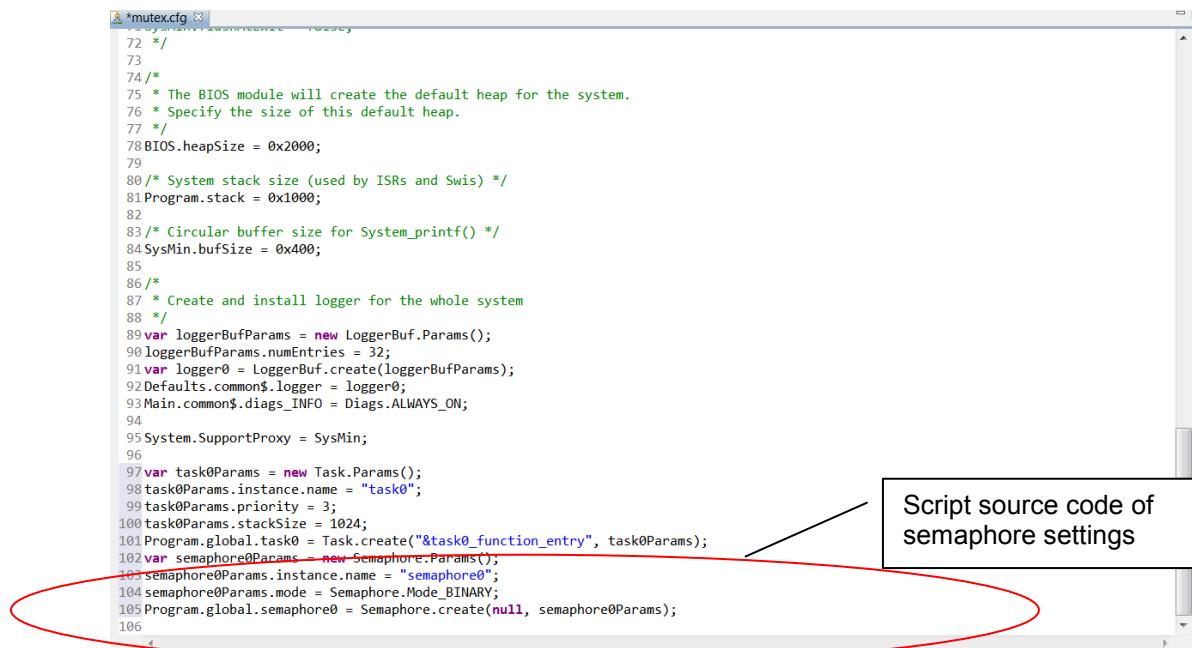


Figure 16. XDCtools GUI Window (Semaphore Management)


```
*mutex.cfg
72 */
73
74/*
75 * The BIOS module will create the default heap for the system.
76 * Specify the size of this default heap.
77 */
78 BIOS.heapSize = 0x2000;
79
80/* System stack size (used by ISRs and Swis) */
81 Program.stack = 0x1000;
82
83/* Circular buffer size for System_printf() */
84 SysMin.bufSize = 0x400;
85
86/*
87 * Create and install logger for the whole system
88 */
89 var loggerBufParams = new LoggerBuf.Params();
90 loggerBufParams.numEntries = 32;
91 var logger0 = LoggerBuf.create(loggerBufParams);
92 Defaults.common$.logger = logger0;
93 Main.common$.diags_INFO = Diags.ALWAYS_ON;
94
95 System.SupportProxy = SysMin;
96
97 var task0Params = new Task.Params();
98 task0Params.instance.name = "task0";
99 task0Params.priority = 3;
100 task0Params.stackSize = 1024;
101 Program.global.task0 = Task.create("task0_function_entry", task0Params);
102 var semaphore0Params = new Semaphore.Params();
103 semaphore0Params.instance.name = "semaphore0";
104 semaphore0Params.mode = Semaphore.Mode_BINARY;
105 Program.global.semaphore0 = Semaphore.create(null, semaphore0Params);
106
```

Figure 17. XDCtools GUI Window (Script Source Code for Semaphore)

3.5 Use of the Mailboxes

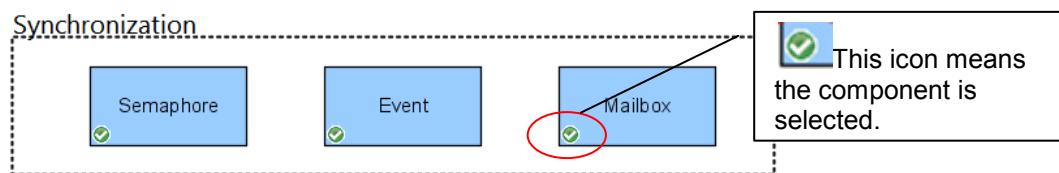
3.5.1 Introduction About the Mailboxes

Mailboxes are used to pass buffers from one task to another task in the system. The mailbox module copies the buffer to fixed-size internal buffers. The size and number of these buffers are specified when a mailbox instance is created. All buffers sent and received with a same mailbox instance must be of the same size.

3.5.2 Use Mailboxes in Your Project

To use mailboxes in the project, you must ensure the mailbox component has been selected in the System Overview window (see Figure 18). Once the mailbox is added into your configuration (see Figure 19), the related script code is added in the source code window. On the other hand, the head file <ti/sysbios/knl/Mailbox.h> must be included in your code file.

```
var Mailbox = xdc.useModule('ti.sysbios.knl.Mailbox'); //this script code adds the mailbox module into the project.
```


Figure 18. Selection of Synchronization Modules

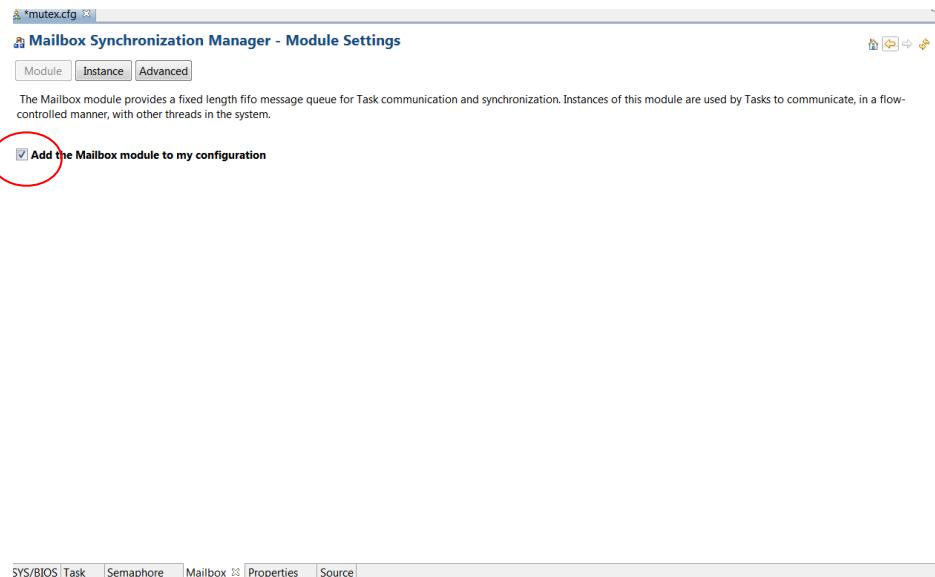


Figure 19. XDCtools GUI Window (Mailbox Module Settings)

Open the Outline window and double-click the Mailbox icon, the GUI of the Semaphore Management is displayed (see Figure 20). Mailboxes can be added or removed by simply clicking the [Add] or [Remove] button. Normally, it is necessary to define only the Handle name, the size, and the number parameters for a mailbox. However, the size of the mailbox must be same as the size of the buffer which you will operate with this mailbox.

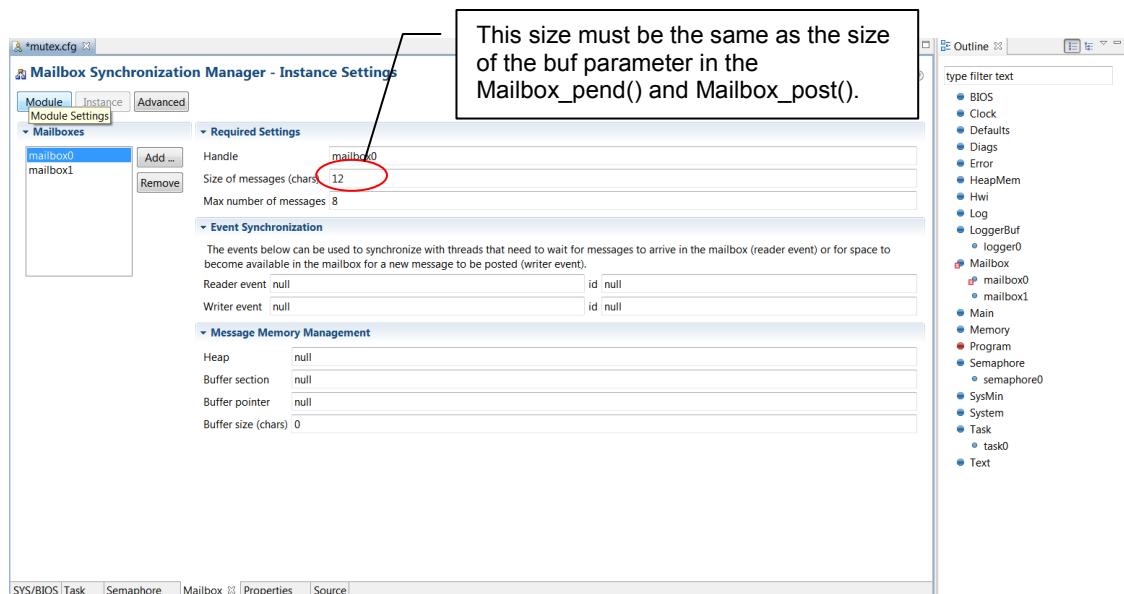


Figure 20. XDCtools GUI Window (Mailbox Instance Settings)

3.5.3 Often-used SYS/BIOS API Function for Mailbox

Two often-used functions for mailbox are:

```
Bool Mailbox_pend(Mailbox_Handle handle, Ptr buf, UInt timeout);
Bool Mailbox_post(Mailbox_Handle handle, Ptr buf, UInt timeout);
```

Mailbox_pend() is used to read a buffer from a mailbox. If no buffer is available (that is, the mailbox is empty) and the time-out parameter is WAIT_FOREVER, Mailbox_pend() blocks.

Mailbox_post() is used to post a buffer to the mailbox. If no buffer slots are available (that is, the mailbox is full) and the time-out parameter is WAIT_FOREVER, Mailbox_post() blocks.

3.6 Use of the Hwi Tasks

3.6.1 *Introduction to the Hwi*

The Hwi is a special type of target- and device-specific task. Hwis are used to manage hardware interrupts, and they must be applied in response to external asynchronous events. For a detailed introduction, refer Section 3.3 in the TI SYS/BIOS v6.33 Real-time Operating System User's Guide (SPRUEX3M) (<http://www.ti.com/lit/ug/spruex3m/spruex3m.pdf>).

3.6.2 *Add Hwi Module in Your Project*

To use Hwi tasks, the target-specific hardware interrupts module must be added into the configuration of the project. We use Cortex-M3 core as the example. Figure 21 shows the module settings. If you are not familiar with the configuration, try to keep the default values of most options.

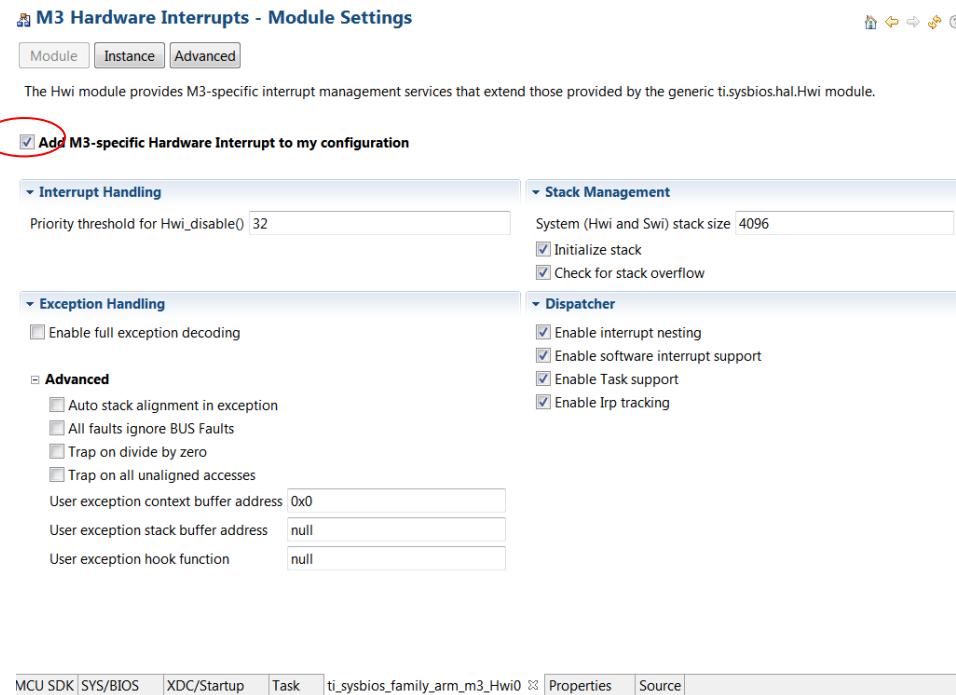


Figure 21. XDCtools GUI Window (Hardware Interrupt Module Settings)

The following script code is automatically added into the source of the .cfg file after the “Add M3-specific Hardware Interrupt to my configuration” checkbox is selected.

```
var ti_sysbios_family_arm_m3_Hwi = xdc.useModule('ti.sysbios.family.arm.m3.Hwi');
```

3.6.3 An Hwi Instance Example Based on Concerto™ F28M35x

We use the IPC3 interrupt as the example to introduce how to configure the Hwi task for it.

Open the Outline window and double-click the Hwi icon, the GUI of the M3 Hardware Interrupts – Instance Settings is displayed (see Figure 22). Hwis can be added or removed by simply clicking the [Add] or [Remove] button. The “ISR function” processes the respond for the interrupt. For the Interrupt number and Interrupt priority settings, we must refer the data sheet and the Reference Manual about the device.

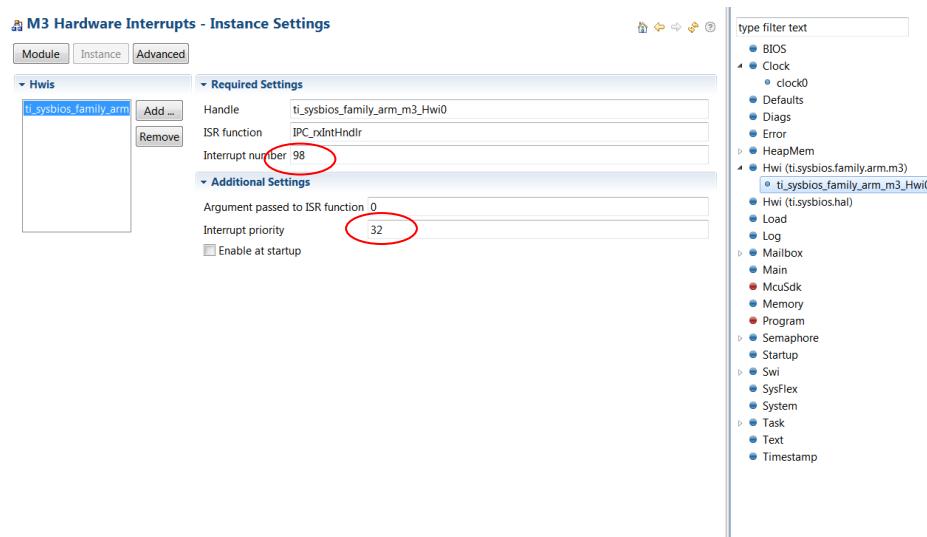


Figure 22. XDCtools GUI Window (Hardware Interrupt Instance Settings)

Table 2 is an interrupts table taken from SPRS742H (<http://www.ti.com/lit/ds/symlink/f28m35h52c.pdf>). In this table, we see that CTOMIPC3 has a vector number of 98 and an interrupt number of 82. The vector number 98 is the interrupt number 98 which was entered in the XDCtool shown as Figure 22.

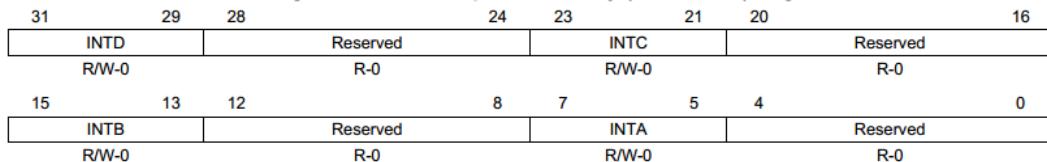
Table 2. Interrupts from NVIC to Cortex-M3

Interrupt Number (Bit in Interrupt Registers)	Vector Number	Vector Address or Offset	Description
59	75	0x0000.012C	UART3
60	76	0x0000.0130	UART4
61–63	77–79	–	Reserved
64	80	0x0000.0140	CAN1 INT0
65	81	0x0000.0144	CAN1 INT1
66	82	0x0000.0148	CAN1 INT0
67	83	0x0000.014C	CAN1 INT1
68–71	84–87	–	Reserved
72	88	0x0000.0160	ADCINT1
73	89	0x0000.0164	ADCINT2
74	90	0x0000.0168	ADCINT3
75	91	0x0000.016C	ADCINT4
76	92	0x0000.0170	ADCINT5
77	93	0x0000.0174	ADCINT6
78	94	0x0000.0178	ADCINT7
79	95	0x0000.017C	ADCINT8
80	96	0x0000.0180	CTOMIPC1
81	97	0x0000.0184	CTOMIPC2
82	98	0x0000.0188	CTOMIPC3
83	99	0x0000.018C	CTOMIPC4

In SPRUH22B (<http://www.ti.com/lit/ug/spruh22f/spruh22f.pdf>), the priority register for interrupt 82 has been assigned to PRI20. See Table 3 for the definition. Then, we determined that bit 22 in Table 4, which is bit 6 of the PRI20 register, is used for the interrupt priority setting for interrupt 82. To set the priority of interrupt 82, the value of PRI20 should be set to hex value 0x20 (equal to decimal value 32). This value is the value 32, which should be entered into the “Interrupt priority” edit box of the Instance Settings GUI (see Figure 22).

Table 3. Peripherals Register Map

Offset	Name	Type	Reset	Description
0x408	PRI2	R/W	0x0000.0000	Interrupt 8-11 Priority
0x40C	PRI3	R/W	0x0000.0000	Interrupt 12-15 Priority
0x410	PRI4	R/W	0x0000.0000	Interrupt 16-19 Priority
0x414	PRI5	R/W	0x0000.0000	Interrupt 20-23 Priority
0x418	PRI6	R/W	0x0000.0000	Interrupt 24-27 Priority
0x41C	PRI7	R/W	0x0000.0000	Interrupt 28-31 Priority
0x420	PRI8	R/W	0x0000.0000	Interrupt 32-35 Priority
0x424	PRI9	R/W	0x0000.0000	Interrupt 36-39 Priority
0x428	PRI10	R/W	0x0000.0000	Interrupt 40-43 Priority
0x42C	PRI11	R/W	0x0000.0000	Interrupt 44-47 Priority
0x430	PRI12	R/W	0x0000.0000	Interrupt 48-51 Priority
0x434	PRI13	R/W	0x0000.0000	Interrupt 52-55 Priority
0x438	PRI14	R/W	0x0000.0000	Interrupt 56-59 Priority
0x43C	PRI15	R/W	0x0000.0000	Interrupt 60-63 Priority
0x440	PRI16	R/W	0x0000.0000	Interrupt 64-67 Priority
0x444	PRI17	R/W	0x0000.0000	Interrupt 68-71 Priority
0x448	PRI18	R/W	0x0000.0000	Interrupt 72-75 Priority
0x44C	PRI19	R/W	0x0000.0000	Interrupt 76-79 Priority
0x450	PRI20	R/W	0x0000.0000	Interrupt 80-83 Priority

**Figure 23. Interrupt 0 to 91 Priority (PRI0 to PRI22) Registers****Table 4. Interrupt 0 to 91 Priority (PRI0 to PRI22) Registers Field Descriptions**

Bit	Field	Value	Description
31-29	INTD		Interrupt Priority for Interrupt [4n+3] This field holds a priority value, 0-7, for the interrupt with the number [4n+3], where n is the number of the Interrupt Priority register (n=0 for PRI0, and so on). The lower the value, the greater the priority of the corresponding interrupt.
28-24	Reserved		Reserved
23-21	INTC		Interrupt Priority for Interrupt [4n+2] This field holds a priority value, 0-7, for the interrupt with the number [4n+2], where n is the number of the Interrupt Priority register (n=0 for PRI0, and so on). The lower the value, the greater the priority of the corresponding interrupt.
20-16	Reserved		Reserved
15-13	INTB		Interrupt Priority for Interrupt [4n+1] This field holds a priority value, 0-7, for the interrupt with the number [4n+1], where n is the number of the Interrupt Priority register (n=0 for PRI0, and so on). The lower the value, the greater the priority of the corresponding interrupt.
12-8	Reserved		Reserved
7-5	INTA		Interrupt Priority for Interrupt [4n] This field holds a priority value, 0-7, for the interrupt with the number [4n], where n is the number of the Interrupt Priority register (n=0 for PRI0, and so on). The lower the value, the greater the priority of the corresponding interrupt.
4-0	Reserved		Reserved

To use the Hwi tasks in your project, the head file <ti/sysbios/family/arm/m3/Hwi.h> must be included in your .c file. The other hardware-related head files such as hw_ints.h, which is in the "MWare\inc", should also be included in your .c files.

In this example, the ISR function body is as follows:

```
void IPC_rxIntHndl(void)
{
    // stat
    IPC_ctxt.rx_isr++;

    // mark CM3 is busy
    IPC_setBusyIndn();

    // post semaphore to task
    CM3_Post_HCT_RX_Semaphore();

    // Acknowledge IPC INT3 from C28 to ACK
    HWREG(MTOCIPC_BASE + IPC_O_CTOMIPCACK) |= IPC_CTOMIPCACK_IPC3;
}
```

To make a Hwi task work normally, it is very important to set the correct configuration for the task.

References

1. *TI-RTOS 1.21 Getting Started Guide* (SPRUHD3G)
2. *TI SYS/BIOS v6.35 Real-time Operating System User's Guide* (SPRUEX3M)
3. *TI SYS/BIOS Real-time Operating System v6.x User's Guide* (SPRUEX3J)
4. *Concerto Microcontrollers* (SPRS742H)
5. *Concerto F28M35x Technical Reference Manual* (SPRUH22B)

IMPORTANT NOTICE

Texas Instruments Incorporated and its subsidiaries (TI) reserve the right to make corrections, enhancements, improvements and other changes to its semiconductor products and services per JESD46, latest issue, and to discontinue any product or service per JESD48, latest issue. Buyers should obtain the latest relevant information before placing orders and should verify that such information is current and complete. All semiconductor products (also referred to herein as "components") are sold subject to TI's terms and conditions of sale supplied at the time of order acknowledgment.

TI warrants performance of its components to the specifications applicable at the time of sale, in accordance with the warranty in TI's terms and conditions of sale of semiconductor products. Testing and other quality control techniques are used to the extent TI deems necessary to support this warranty. Except where mandated by applicable law, testing of all parameters of each component is not necessarily performed.

TI assumes no liability for applications assistance or the design of Buyers' products. Buyers are responsible for their products and applications using TI components. To minimize the risks associated with Buyers' products and applications, Buyers should provide adequate design and operating safeguards.

TI does not warrant or represent that any license, either express or implied, is granted under any patent right, copyright, mask work right, or other intellectual property right relating to any combination, machine, or process in which TI components or services are used. Information published by TI regarding third-party products or services does not constitute a license to use such products or services or a warranty or endorsement thereof. Use of such information may require a license from a third party under the patents or other intellectual property of the third party, or a license from TI under the patents or other intellectual property of TI.

Reproduction of significant portions of TI information in TI data books or data sheets is permissible only if reproduction is without alteration and is accompanied by all associated warranties, conditions, limitations, and notices. TI is not responsible or liable for such altered documentation. Information of third parties may be subject to additional restrictions.

Resale of TI components or services with statements different from or beyond the parameters stated by TI for that component or service voids all express and any implied warranties for the associated TI component or service and is an unfair and deceptive business practice. TI is not responsible or liable for any such statements.

Buyer acknowledges and agrees that it is solely responsible for compliance with all legal, regulatory and safety-related requirements concerning its products, and any use of TI components in its applications, notwithstanding any applications-related information or support that may be provided by TI. Buyer represents and agrees that it has all the necessary expertise to create and implement safeguards which anticipate dangerous consequences of failures, monitor failures and their consequences, lessen the likelihood of failures that might cause harm and take appropriate remedial actions. Buyer will fully indemnify TI and its representatives against any damages arising out of the use of any TI components in safety-critical applications.

In some cases, TI components may be promoted specifically to facilitate safety-related applications. With such components, TI's goal is to help enable customers to design and create their own end-product solutions that meet applicable functional safety standards and requirements. Nonetheless, such components are subject to these terms.

No TI components are authorized for use in FDA Class III (or similar life-critical medical equipment) unless authorized officers of the parties have executed a special agreement specifically governing such use.

Only those TI components which TI has specifically designated as military grade or "enhanced plastic" are designed and intended for use in military/aerospace applications or environments. Buyer acknowledges and agrees that any military or aerospace use of TI components which have **not** been so designated is solely at the Buyer's risk, and that Buyer is solely responsible for compliance with all legal and regulatory requirements in connection with such use.

TI has specifically designated certain components as meeting ISO/TS16949 requirements, mainly for automotive use. In any case of use of non-designated products, TI will not be responsible for any failure to meet ISO/TS16949.

Products	Applications		
Audio	www.ti.com/audio	Automotive and Transportation	www.ti.com/automotive
Amplifiers	amplifier.ti.com	Communications and Telecom	www.ti.com/communications
Data Converters	dataconverter.ti.com	Computers and Peripherals	www.ti.com/computers
DLP® Products	www.dlp.com	Consumer Electronics	www.ti.com/consumer-apps
DSP	dsp.ti.com	Energy and Lighting	www.ti.com/energy
Clocks and Timers	www.ti.com/clocks	Industrial	www.ti.com/industrial
Interface	interface.ti.com	Medical	www.ti.com/medical
Logic	logic.ti.com	Security	www.ti.com/security
Power Mgmt	power.ti.com	Space, Avionics and Defense	www.ti.com/space-avionics-defense
Microcontrollers	microcontroller.ti.com	Video and Imaging	www.ti.com/video
RFID	www.ti-rfid.com	TI E2E Community	
OMAP Applications Processors	www.ti.com/omap	e2e.ti.com	
Wireless Connectivity	www.ti.com/wirelessconnectivity		