

使用具有 **Stellaris® ARM® Cortex™-M3** 微控制器的 **SYS/BIOS**

Peggy Liska

Stellaris® Microcontrollers

摘要

本文档简要概括了德州仪器 SYS/BIOS 实时操作系统 (RTOS) 并概述了如何在 Stellaris® ARM® Cortex™-M3 系列微控制器上执行 SYS/BIOS。除了在 SYS/BIOS 软件包内建立和运行一个示例项目，本文档还概述了使用 Code Composer Studio™ v4.2 (CCS) 集成开发环境 (IDE) 创建一个新的 SYS/BIOS 项目的过程。作为最后一个步骤，本文档概述了将 StellarisWare® 应用编程结果 (API) 集成到一个 SYS/BIOS 项目中。

内容

1	简介	2
2	SYS/BIOS 概述	2
3	下载 SYS/BIOS	16
4	运行一个现有的 SYS/BIOS 项目	18
5	创建一个实例 SYS/BIOS 项目	22
6	结论	41
7	参考书目	41

图片列表

1	打开 .cfg 文件	3
2	配置工具概述	3
3	添加和删除模块	4
4	创建新的模块示例	4
5	编辑模块选项	5
6	基于文本的编辑器	5
7	内在线程优先级	6
8	前台和后台调度	7
9	硬件与软件中断间的关系	8
10	HWI 和 SWI 示例	9
11	同一优先级 SWI	9
12	任务状态图	11
13	任务在 Semaphore_Pend() 之后继续运行	12
14	用一个信号量来阻断任务	12
15	使用信号量解除对任务的阻断	13
16	当多个任务被阻断时，使用一个信号量来将一个任务解除阻断	13
17	实时分析工具	15

Code Composer Studio is a trademark of Texas Instruments.
Stellaris, StellarisWare are registered trademarks of Texas Instruments.
Cortex is a trademark of ARM Limited.
ARM is a registered trademark of ARM Limited.
All other trademarks are the property of their respective owners.

1 简介

SYS/BIOS 是一款由德州仪器创建的实时操作系统 (RTOS) 内核，此内核用于多种 TI 微处理器和包括 Stellaris Cortex-M3 系列器件在内的微控制器上的嵌入式应用。SYS/BIOS RTOS 的几个关键优势包括：

- 调度
- 优先、确定多任务
- 配置工具
- 存储器管理
- 针对交叉平台应用的硬件抽象

SYS/BIOS 包括在 Code Composer Studio (CCS) 的安装包内，所以创建一个新项目是非常简便的。内核的配置由名为 XGConf（基于开源代码 XDCT 工具）的 SYS/BIOS 配置工具从硬件中抽取。这是一个图形用户接口 (GUI)，此接口可实现对诸如任务、中断和信号量等 RTOS 组件的快速和简便操纵。这个 GUI 被装在 CCS 内并自动生成配置 SYS/BIOS 内核所需的初始化代码。

本文档的剩余部分详细描述了 SYS/BIOS 内核以及在一个 Stellaris ARM Cortex-M3 微控制器上执行一个基于 SYS/BIOS 的项目的过程。与 SYS/BIOS 有关的更多详细信息请参阅7 节，参考内的文档。

2 SYS/BIOS 概述

SYS/BIOS 是一款由德州仪器设计的可在多种 TI 微控制器上运行的 RTOS 内核。SYS/BIOS 提供以下特性：

- 占先，基于优先级的调度调度程序
- 存储器分配和堆栈管理
- I/O 处理

占先、基于优先级的调度程序确保以及为运行做好准备的最高优先级线程由系统执行。这个配置确保对时间敏感的的计算的处理延迟最小。为了改进最终应用的代码尺寸，SYS/BIOS 内核可针对定制项目进行优化。详细时序和尺寸基准信息可在[SYS/BIOS 维基](#)网页内找到，此网页位于[德州仪器嵌入式处理器维基网页内](#)。

使用 SYS/BIOS 的一个优势是 TI 数字信号处理器 (DSP)，基于 ARM 的微控制器和 MSP430 微控制器间应用程序的可移植性。由 RTOS 完成的硬件抽象可在无需或只需很少修改的情况下实现不同产品系列间的代码移植。

本概述是基于 SYS/BIOS 版本 6.32.02.39。SYS/BIOS 的不同版本也许会包含稍微不同的特性。

2.1 配置工具

SYS/BIOS 内核的配置被保存在项目的 *.cfg 文件内。这个文件定义了系统中所使用的模块以及任一静态定义的模块示例内使用的基本参数。

配置文件可使用两个方法中的一个进行修改：

- 图形方法（使用 XGCONF 配置工具）
- 文本方法（使用 XDCScript 编辑器或其它文本编辑器）

在一个项目被载入到 CCS 后，通过右键点击项目中的文件并如[图 1](#)所示来选择 Open with > XGCONF 来打开 *.cfg 文件。对于在 CCS 中打开一个 SYS/BIOS 项目的命令，请参见[4 节](#)运行一个已有的 SYS/BIOS 项目。

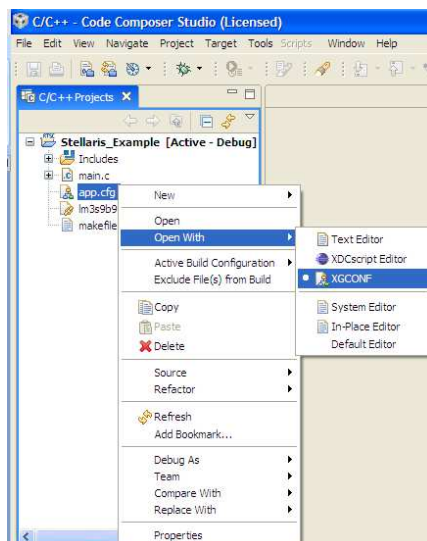


图 1. 打开 .cfg 文件

图 2显示了文件打开后出现的配置工具的系统概述标签页。

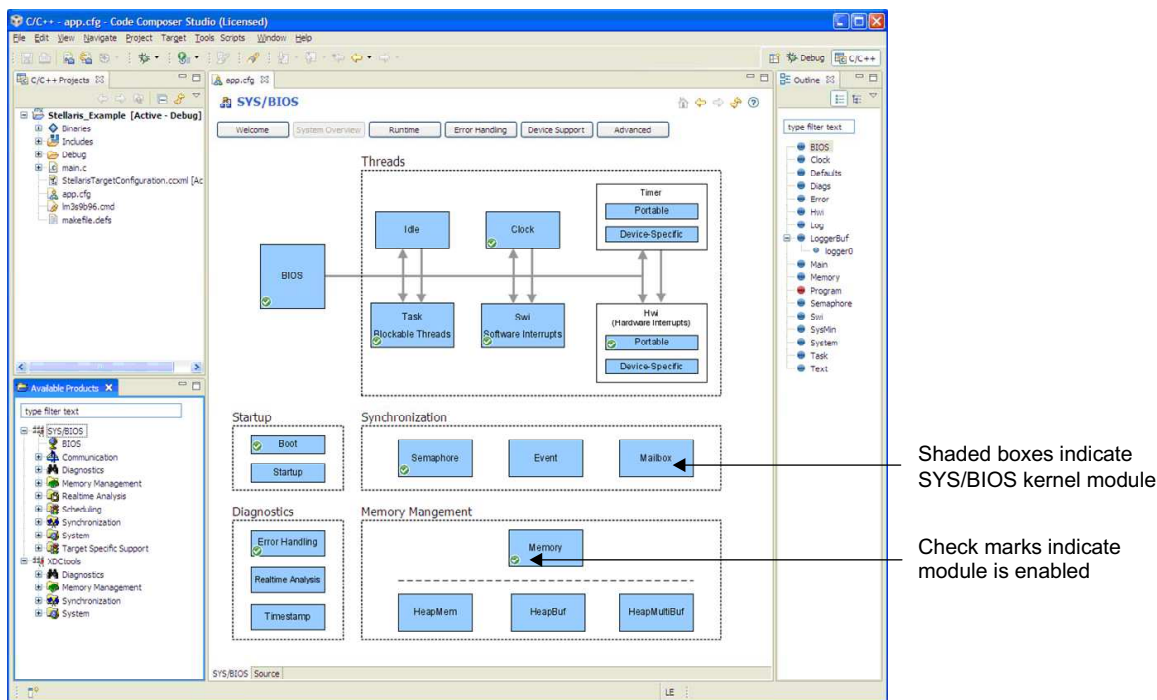


图 2. 配置工具概述

屏幕上的每个阴影块与 SYS/BIOS 内核中的一个模块相对应。阴影框左下角的选中标记表示此模块已在当前的配置中被启用。有两个方法来启用和禁用系统内模块的使用：

- 右键点击 GUI 内的阴影框
- 或者
- 右键点击位于屏幕左侧的可用产品列表内的模块

图 3 显示了在一个 SYS/BIOS 项目中添加和删除模块的两个方法。

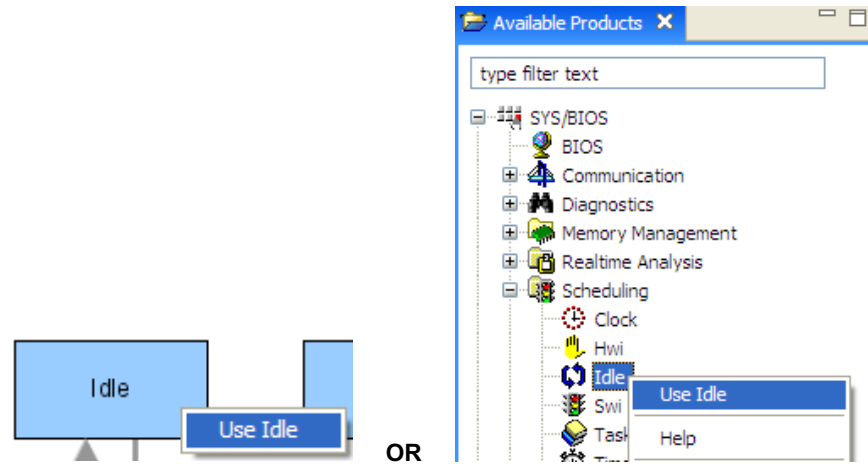


图 3. 添加和删除模块

可使用两个方法从一个程序中静态地删除已启用模块的示例或者将已启用模块的示例静态地添加到一个程序中

- 右键点击 GUI 内的阴影框
- 或者
- 右键点击大纲中的项目

图 4 显示了在程序内创建一个新任务的两个方法。请注意，通过使用对 SYS/BIOS 应用编程接口 (API) 的调用，可在运行时间内动态创建和配置模块示例。



图 4. 创建新的模块示例

双击阴影框或点击在大纲中的项目可打开一个具有针对模块的可用选项的窗口。

图 5 显示了当任务模块被选择时出现的选项窗口。这个窗口控制全局任务选项。针对单个任务的选项可通过点击大纲中的特定任务进行修改。

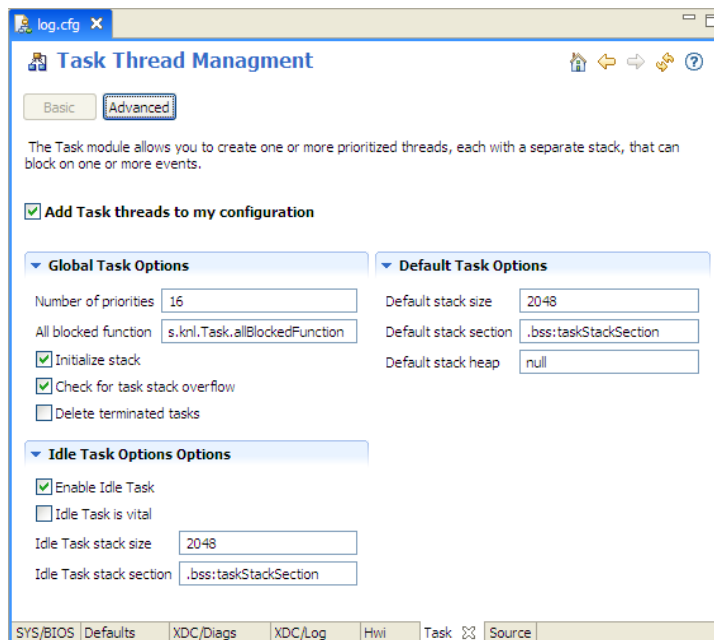


图 5. 编辑模块选项

除了图形化配置工具，一个基于文本的编辑器也可被用来配置 RTOS。图 5 中显示的窗口的底部是一个 Source（源）标签页。这个标签页显示配置文件的文本版本。当一个模块被选中时，相关的代码被标出，如图 6 所示。点击一个大纲中的项目将改变被标出的部分代码。

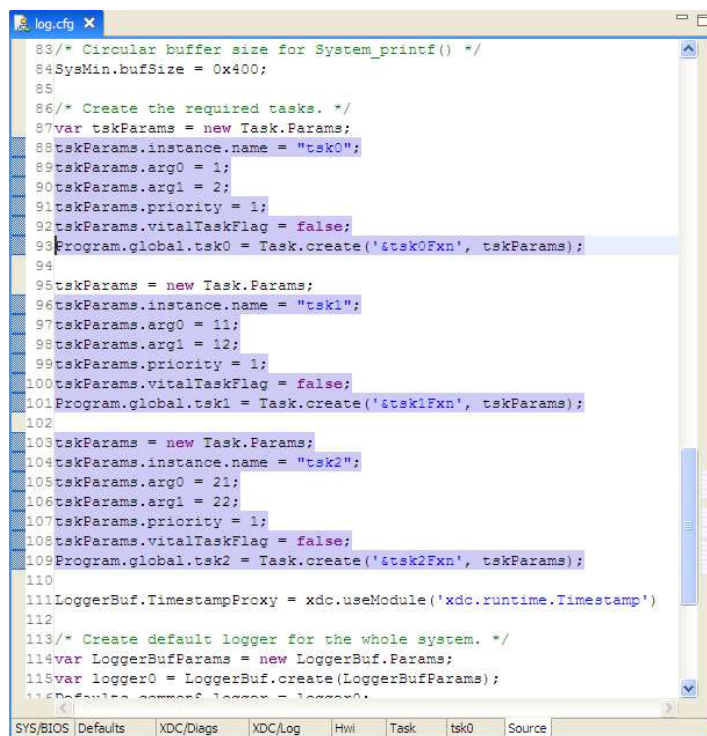


图 6. 基于文本的编辑器

通过右键点击项目中的 .cfg 文件并选择 Open with > XDCScript Editor 或者选择 Open With > Text Editor 可在 GUI 编辑器之外打开基于文本的编辑器。

可用参数的详细说明可在 Code Composer Studio 的 Help > Help Contents 菜单的 Help Contents 中的 SYS/BIOS 部分内的 API 参考中找到。

2.2 线程和优先级

在 SYS/BIOS 内部，术语线程是指一组指令，这组指令可在所需的寄存器已经被适当地初始化之后由 CPU 执行。SYS/BIOS 内提供四种类型的线程：

- 硬件中断 (HWI)
- 软件中断 (SWI)
- 任务
- 空闲

在 SYS/BIOS 系统内有两种类型的优先级：

- 内在优先级 - 由线程类型定义
- 外在优先级 - 由编程人员定义

图 7 中显示了针对四种主要线程类型的内在优先级。

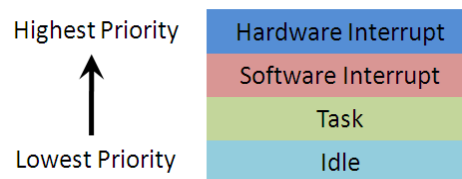


图 7. 内在线程优先级

在系统中，硬件中断具有最高的内在优先级，其次是软件中断、任务，最后是内在优先级最低的空闲任务。

可为单独的 HWI，SWI 和任务指定低于内在优先级的外在优先级。在 Stellaris 器件上，硬件中断优先级的最大数量为 8，而软件中断优先级的最大数量为 32，任务优先级的最大数量为 32。在一个基于 SYS/BIOS 的程序中，指定给 HWI 的优先级根据代码汇编运行的硬件而发生改变。在 Stellaris 器件上，最高优先级 HWI 具有一个 0 值，并且最低值取决于针对 HWI 所定义的优先级的数量。SWI 和任务接受 SYS/BIOS 缺省优先级，这样最低优先级有一个 0 值。针对 SWI 和任务的最高优先级取决于被执行的软件中断优先级以及任务优先级的数量。

占先、基于优先级的调度程序确保已就绪的最高优先级线程由 CPU 执行。一定数量的线程可在任一指定的时间内就绪运行，但是首先执行优先级最高的线程。为了在线程间切换，线程的运行环境必须被保存。这个运行环境包括程序计数器、堆栈和任何相关的寄存器值。针对每个线程类型的特定堆栈功能性包括在以下部分中。

2.3 硬件中断

不具有 RTOS 的程序在指令循环时（由一个高优先级中断处理例程 (ISR) 中断）通常具有一个无限低优先级。这个类型的系统可在前台和后台调度系统中实现两个优先级。这个配置并不易于扩展并且不能满足要求多个优先级的复杂系统的需要。一个基于 SYS/BIOS 的系统引入可实现这些多优先级和确定性调度的线程处理。

图 8 显示了一个典型程序和一个基于 SYS/BIOS RTOS 的程序间的关系。

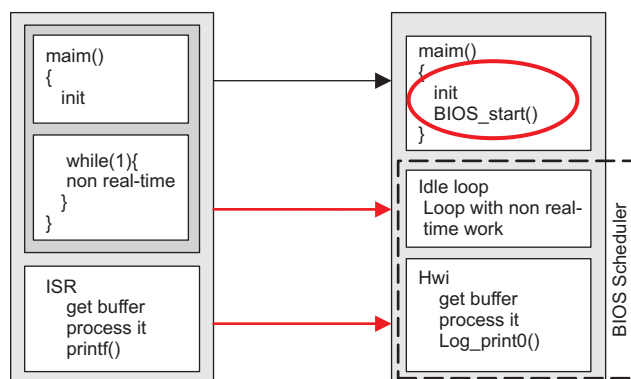


图 8. 前台和后台调度

空闲指令循环运行优先级最低并通常用于激活低功率模块、内置自检 (BIST) 和用户接口代码。这个指令循环被更高优先级线程所取代并在更高优先级代码执行完成时从指令循环被取代的那一点上恢复执行。硬件中断 (HWI) 取代了中断处理例程 (ISR) 的功能性。SYS/BIOS 调度程序在系统外围以确保在线程准备就绪时立即执行最高优先级的线程并保证在 HWI 执行完成时程序执行返回到空闲指令循环。BIOS_start() 函数在主程序将主 Init() 函数内所需的硬件和外设初始化之后启动调度程序。

除了 SYS/BIOS 程序，可在缺省配置中启用的另外一个选项是中断调度程序。中断调度程序可使硬件中断与其它线程一起正常运行，这是因为其熟悉 SYS/BIOS 调度程序。当更高优先级任务正在被执行时，低优先级线程被适当禁用。此调度程序还启用几个调试特性，使得用户能够看见中断何时发生。这个中断调度程序是一段代码，此代码对于所有中断通用，这样此调度程序能够减少代码的封装尺寸。中断调度程序包含三个要素：

- 中断矢量 - 调度程序在中断矢量表中的位置
- 调度表 - 包含针对硬件中断的参数
- 中断堆栈 - 当中断进行优先级竞争时保存运行环境

以下的部分概述了中断调度程序操作。括号()中的步骤可根据系统的配置进行优化。已被配置为零延迟的中断不再被调度程序禁用。

1. 全局禁用所有由调度程序管理的中断，但是为了实现关键部分保护，其中不包括非零延迟中断。
2. (禁用任务调度程序)
3. (禁用 SWI 调度程序)
4. 保存中断返回指针
5. (调用已配置的 HWI 开始钩子函数)
6. (全局启用调用程序管理的中断，但是，如果自动中断嵌套支持被启用的话，其中不包括非零延迟中断)
7. 调用 ISR 函数
8. (如果自动中断嵌套支持被启用的话，全局禁用调度程序管理的中断)
9. (调用已配置的 HWI 终止钩子函数)
10. (运行 SWI 调度程序，此调度程序运行 ISR 所发出的任一 SWI)
11. (运行任务调度程序，如果需要的话，此调度程序管理任务占先)
12. 从 ISR 返回

当在一个 SYS/BIOS 程序内部定义一个硬件中断时，请不要使用中断关键字，这是因为中断调度程序已经进行了运行环境的保存。

硬件中断的执行取决于特定系列器件。至于 Stellaris Cortex-M3 系列器件，硬件中断可被配置成使用已嵌套的矢量中断 (NVIC)。NVIC 使得应用能够利用内核中内置的最小中断延迟，但是此延迟并不是 SYS/BIOS 内核的缺省配置。SYS/BIOS 系统的零延迟中断特性绕过了 SYS/BIOS 中断调度程序，因此，防止 ISR 使用 SYS/BIOS 处理和 API。组装此种中断结构类型的额外信息请见[用于 Stellaris 器件的 SYS/BIOS 维基网页](#)。

与硬件中断相关的更多详细信息请参见 Code Composer Studio 内 Help > Help Contents 菜单中 Help Contents 的 SYS/BIOS 部分中的 SYS/BIOS API 参考。

2.4 软件中断

硬件中断通常用于处理时间敏感的资源。然而，一旦数据已经被采集，数据可被传递到限制不是很严格的线程进行进一步处理。在 SYS/BIOS 中，软件中断 (SWI) 通常被用于处理硬件中断数据。它们是优先级占先的并且一般具有 0-15 之间的优先级。SWI 通常从一个单一堆栈运行，此堆栈减少了整个应用的存储器尺寸，但是却防止 SWI 阻断信号量、使用一个睡眠函数将它们自身挂起，或者使用一个退出函数将它们自身终止。SWI 所需的堆栈尺寸与系统所需的优先级数量成正比。SWI 运行环境的保存和恢复由 SYS/BIOS 内核自动处理。图 9 显示了 SWI 是如何与 SYS/BIOS 架构相匹配的。

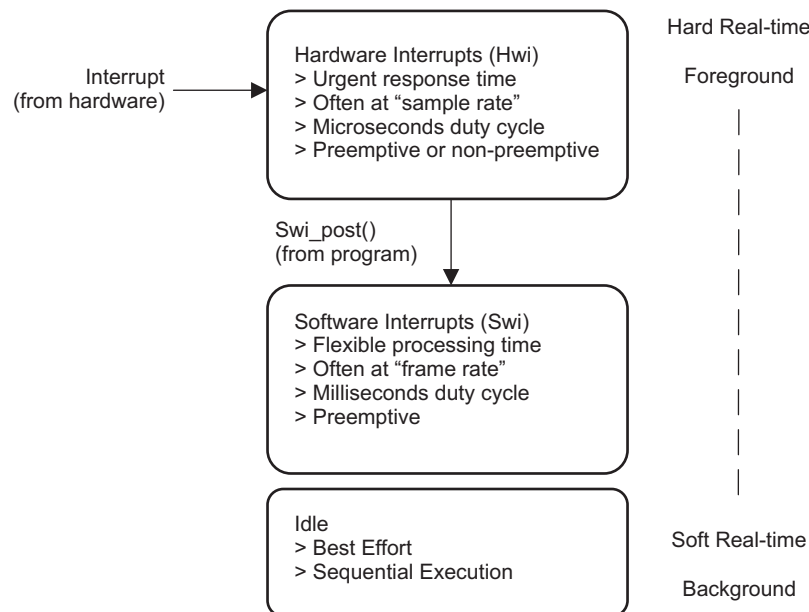


图 9. 硬件与软件中断间的关系

当 Swi_Post () 函数被调用时，通常为一个硬件中断，SWI 为运行做好准备。无论 SWI 在执行前被发出多少次，一个 SWI 只运行一次。

图 10 显示了一个硬件和软件中断被启用的系统是如何处理两个线程的。

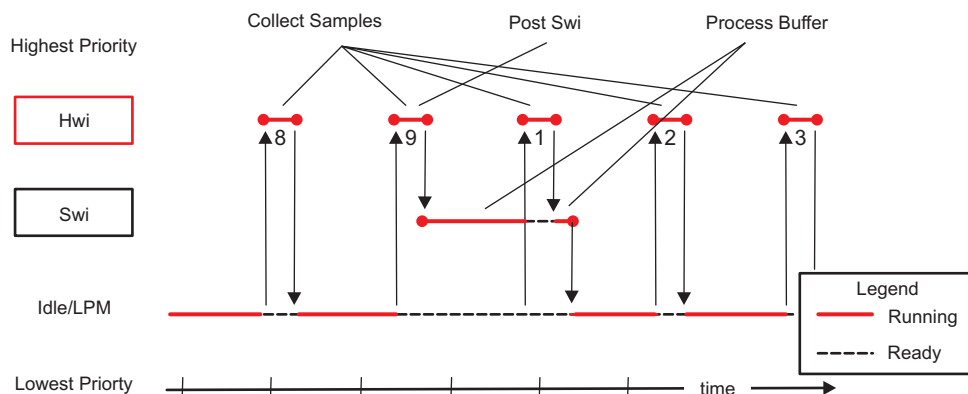


图 10. HWI 和 SWI 示例

HWI 被用来在一个周期时间帧内采集样本并将它们放置在一个缓冲器中。然后，SWI 被用来处理这些样本中的一个（例如，样本 9）。这个数据的处理时间要长于采样之间的时间。使用这一配置，HWI 能够在 SWI 继续处理之前样本的同时检索新数据。一旦 HWI 和 SWI 的处理都已完成，空闲线程重新获得系统的控制权。

具有同样优先级的 SWI 在先进先出 (FIFO) 类型的系统中被处理。图 11 显示了会发生此种操作的示例以及中断是如何被处理的。

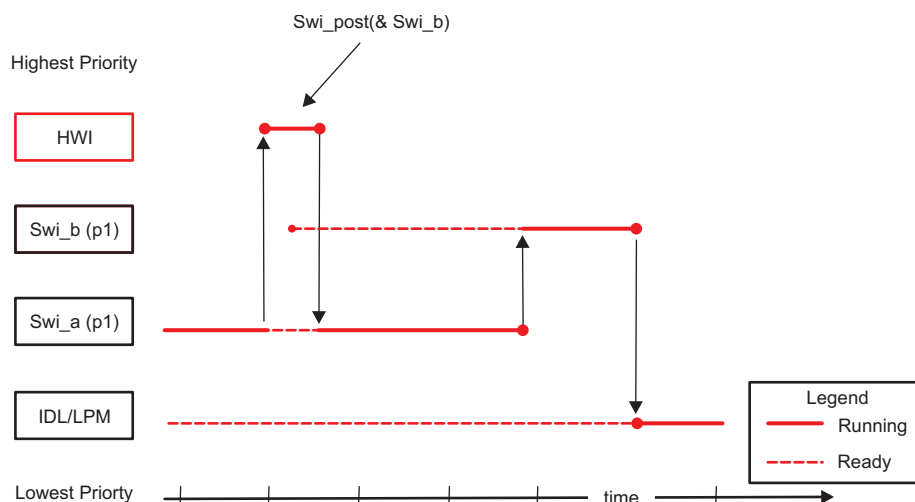


图 11. 同一优先级 SWI

Swi_a 和 Swi_b 被设定为同一个优先级 1。一个 HWI 取代 Swi_a 的执行并在其处理期间发出一个到 Swi_b 的调用。由于这些 SWI 具有相等的优先级，Swi_a 必须在 Swi_b 能够开始执行前完成其指令的执行。一旦两个 SWI 都已完成执行，程序控制返回到空闲线程。

虽然在执行前，无论 SWI 被发出的次数是多少，SWI 只执行一次，但是您也可以确定一个 SWI 被发出的次数。Swi_inc() 函数在 SWI 运行前返回 SWI 被发出的次数。您还可以在 SWI 已经被多次调用后调度一个 SWI。Swi_dec() 函数要求在 SWI 被调度前将一个明确次数发出给 SWI。这些函数和其它函数在 Code Composer Studio 内 Help > Help Contents 菜单中 Help Contents 的 SYS/BIOS 部分中的 SYS/BIOS API 参考中进行了详细解释。

与软件中断有关的更多详细信息，请参见 Code Composer Studio 内 Help > Help Contents 菜单中 Help Contents 的 SYS/BIOS 部分中的 软件中断运行时间 API 中的内容。

2.5 任务

第三个 SYS/BIOS 线程类型是任务。与硬件和软件中断不同，任务能够阻断和允许其它任务的执行。这要求每个任务示例在存储器中有其自身的堆栈，此堆栈与 SWI 和 HWI 所使用的通用堆栈模式相对应。因此，与其它线程类型相比，任务对于存储器更加敏感。根据它们的优先级，任务可占先或被取代，这可以进行动态改变。当任务必须等待一个事件或一个资源的发生而变为可用时，它们使用某些同步模块，诸如一个信号量来将它们自身阻断。

可使用以下两个方法中的一个来在 SYS/BIOS 中创建一个新任务：

- 在汇编时采用静态方法
- 在运行期间内采用动态方法

两个任务能够指向一个单一函数，只要此函数是可再次进入的。此函数的自变量可被用来确定是哪一个任务调用了函数。

一个任务的生命周期包含四个状态：

- **就绪：**一个任务已为运行做好准备
 - 如果任务在启动时被静态创建，那么它就进入这个状态
 - 如果任务是在运行期间由 `task_creat()` API 动态创建，那么任务进入这个状态
 - 就绪运行的最高优先级任务从就绪状态移动至运行状态
 - 如果一个处于就绪状态的任务调用 `task_delete()` 函数，此任务被移动到已终止状态
- **正在运行：**
 - 如果一个更高优先级的任务取代了这个任务，调度程序将此任务放回至就绪状态
 - 如果一个任务正在运行，然后由于此任务正在等待一个事件源而被一个 `semaphore_pend()` API 调用所阻断，其状态被更改为已阻断
 - 如果正在运行的任务调用 `task_yield()` 函数，此任务被移动至就绪状态
 - 如果此任务调用 `task_exit()` 函数，此任务被移动至已终止状态
- **已阻断：**一个任务正在等待一个资源或时间并被发出了一个 `semaphore_pend()` API 调用
 - 如果做出了一个 `semaphore_post` 调用，此任务被移动回就绪状态
- **已终止：**一个任务已经完成执行
 - 当一个任务处于已阻断状态时，不应将此任务终止，已确保程序资源在任务被终止前被适当释放。

图 12 显示了四个任务状态以及如何在这些状态间进行任务转换。

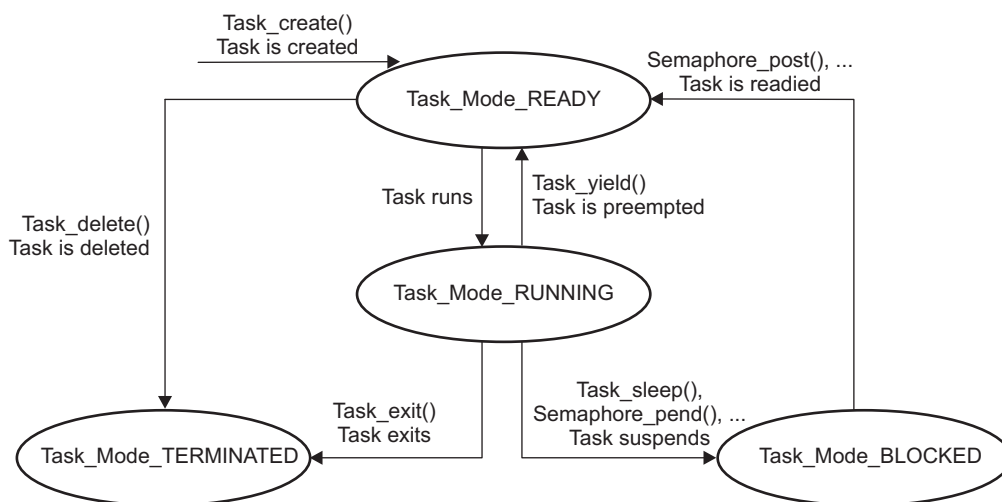


图 12. 任务状态图

与 SYS/BIOS 任务相关的更多信息，请见 Code Composer Studio 内 Help > Help Contents 菜单中 Help Contents 的 SYS/BIOS 部分中的 SYS/BIOS 用户手册中的内容。

2.6 同步模块

SYS/BIOS 内核为系统内的同步任务提供几个不同的模块。其中包括信号量、栅极、事件和邮箱。

信号量通常被用来保护由多个任务进行访问的硬件资源。信号量的使用可确保正在访问硬件资源的任务可在其它任务获得资源的控制权之前完成运行。信号量实用性的一个应用示例就是多个任务需要向一个 UART 接口写入输入数据时。当一个任务正在打印一条消息时中断此任务会在输出终端上生成一个混乱的消息。

栅极与信号量相类似，但是其通常用来保护与硬件资源相对应的共享软件部分。可使用多种类型的预先定义的栅极或创建多种用户定义的栅极。例如，将使用一个栅极来保护对全局变量的写入以确保在一个变量被允许访问前由一个任务将一个有效值分配给该变量。

事件是信号量的另外一个特定执行。它们要求在挂起线程从一个挂起调用返回前具备几个条件。请注意，每次只有一个单一任务能够在事件上挂起。在上面提到的 UART 示例情况下，第二个任务也许应该确保在程序开始将数据写入 UART 前将 UART 模块的写入缓冲器清零。此任务将建立一个事件来在信号量上挂起前检查 UART 写入缓冲器。

邮箱被用来在任务间传递数据缓冲器。可将事件与邮箱关联起来以实现额外同步。邮箱可被用来确保进入缓冲器的数据流不超过系统对这些缓冲器的处理能力。

考虑到本文档的用途，只详述了信号量。与其它同步模式相关相关的额外信息请参见 Code Composer Studio 内 Help > Help Contents 菜单中 Help Contents 的 SYS/BIOS 部分中的 SYS/BIOS 用户手册部分。

信号量被用来保护资源，此资源必须在将控制权切换至其它任务之前在一个任务之内完成所需的操作。在 SYS/BIOS 中有两类信号量。

- 二进制信号量
- 计数信号量

二进制信号量通常用于使用中或可用的资源。计数信号量使得一个资源在被限制访问前可由多个任务进行访问。一个零计数是针对一个信号量的最小可能值并表示此资源不可用。发出信号量增加其计数，而在信号量上挂起将减少其计数。

在一个计数为 1 的信号量 mySem 上调用 semaphore_post(mySem) 将导致：

- 在计数信号量上的 2 计数
- 在二进制信号量上的 1 计数

以下示例和图表显示了信号量的运行。

如果一个任务正在运行并且在计数为 2 的信号量上调用 Semaphore_pend() 函数（也就是说，减少信号的计数），任务如图 13 中所示继续运行。

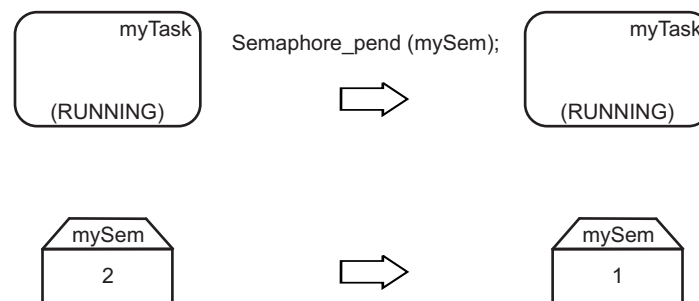


图 13. 任务在 **Semaphore_Pend()** 之后继续运行

然而，如果一个任务正在运行并且此任务在计数为 0 的信号量上调用 Semaphore_pend() 函数，任务如图 14 中所示发生阻断。

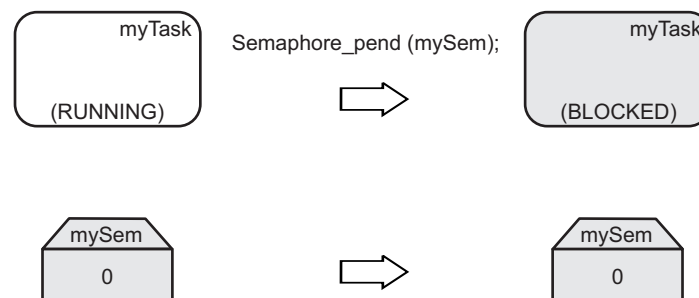


图 14. 用一个信号量来阻断任务

为了解除对任务的阻断，其它任务必须调用 semaphore_post() 函数来增加信号量的计数。在图 15 中的示例中，较低优先级的任务发出信号量以表示此任务不再需要对受限制资源进行访问。然后，更好优先级的任务从被阻断切换回运行状态，这是因为此任务正在等待这个资源变为可用。只要任务开始运行，任务在信号量上挂起以通知程序此任务正在使用由信号量控制的资源。

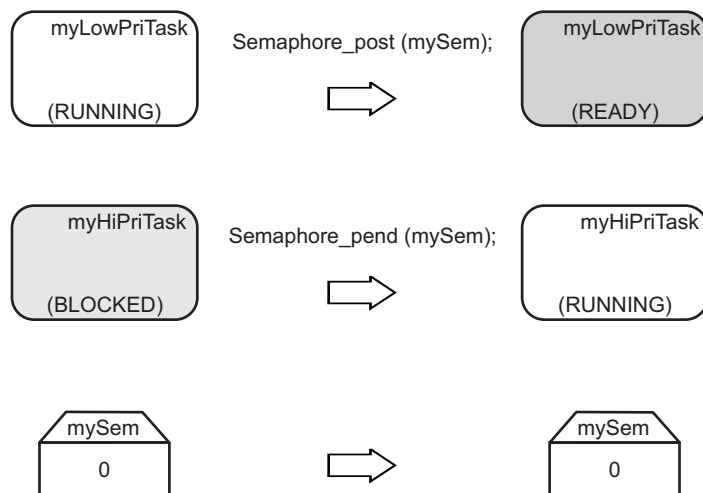


图 15. 使用信号量解除对任务的阻断

请注意，在 SYS/BIOS 内核中，信号量队列中的第一个任务（不一定是具有更高优先级的任务）是第一个被解除阻断的任务。这意味着当信号量被发出时，第一个被阻断的任务是第一个就绪的任务。此外，还请注意，在同一信号量上可阻断多个任务（如图 16 所示），在这里，高优先级和中优先级任务都在等待同一个信号量。由于中优先级任务是最先在信号量上挂起的任务，中优先级任务被允许在较高优先级任务之前执行。可使用一个被称为 `GateMutexPri` 的特定栅极类型来防止此类优先级倒置。

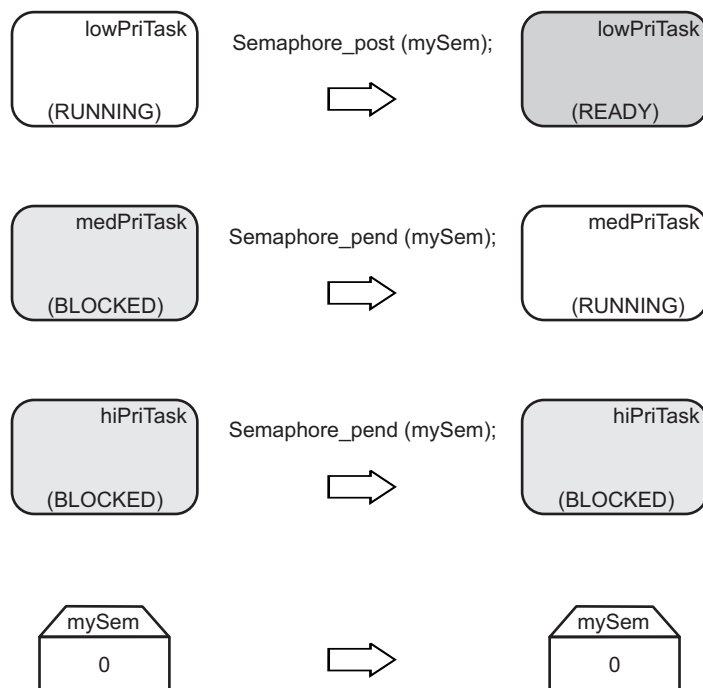


图 16. 当多个任务被阻断时，使用一个信号量来将一个任务解除阻断

与 SYS/BIOS 信号量有关的更多信息，请参见 Code Composer Studio 内 Help > Help Contents 菜单中 Help Contents 的 SYS/BIOS 部分中的 SYS/BIOS 用户手册中的内容。

2.7 定时器和时钟

可在 SYS/BIOS 内配置定时器和时钟来保持 TI 微控制器之间的可移植性。当使用 SYS/BIOS 定时器模块时，内核管理器件上的硬件定时器外设。定时器线程正在 HWI 线程的运行环境中运行。

SYS/BIOS 时钟模块位于定时器模块的顶层并管理 RTOS 时基。时钟模块可运行为单次定时器或者通过使用软件中断 (SWI) 的功能性和优先级进行周期运行。时钟运行在同一个 SWI 优先级上，这样它们不会相互取代。

在使用 Stellaris Cortex-M3 的情况下，通过将以下代码行添加到配置文件中，内核中的 SysTick 定时器可被用作 RTOS 时基：

```
var halTimer = xdc.useModule('ti.sysbios.hal.Timer');  
halTimer.TimerProxy = xdc.useModule('ti.sysbios.family.arm.m3.Timer');
```

与 SYS/BIOS 应用中的特定 Cortex-M3 定时器相关的更多信息，请参见[用于 Stellaris 器件的 SYS/BIOS 维基](#)网页。

除了定时器和时钟模块，一个时间戳模块被提供为 SYS/BIOS 架构的一个组件。这个模块对于要求准确定时测量的基准应用十分有用。

与 SYS/BIOS 定时器和时钟相关的更多信息，请参见 Code Composer Studio 内 Help > Help Contents 菜单中 Help Contents 的 SYS/BIOS 部分中的 *SYS/BIOS 用户手册* 中的内容。

2.8 调试工具

实时系统的调试过程需要额外的实时分析工具，这是因为传统调试工具所引入到系统中的延迟。SYS/BIOS 提供几种工具和特性，其中包括：

- 有效性
- 应用和 RTOS 内的日志记录
- 实时分析

有效性检查用于普通用户错误，例如使用一个无效自变量或者从一个不被支持的运行环境中调用一个 API。日志记录将应用和 RTOS 中的重要数据打印至系统中的用于调试时序问题的终端。实时分析将调试数据注入主机，而又不会停止处理器。

请注意 SYS/BIOS 调试工具的一个重要特性，日志和跟踪信息在主机计算机上被格式化，这样应用处理器能够继续执行代码而不受干扰，或受到的干扰最小。

为了访问 SYS/BIOS 实时分析 (RTA) 工具，请确保 CCS 处于调试模式中。然后，导航至 Tools > RTA 菜单选项来查看可用的 RTA 工具。图 17 显示了可用的 RTA 工具。

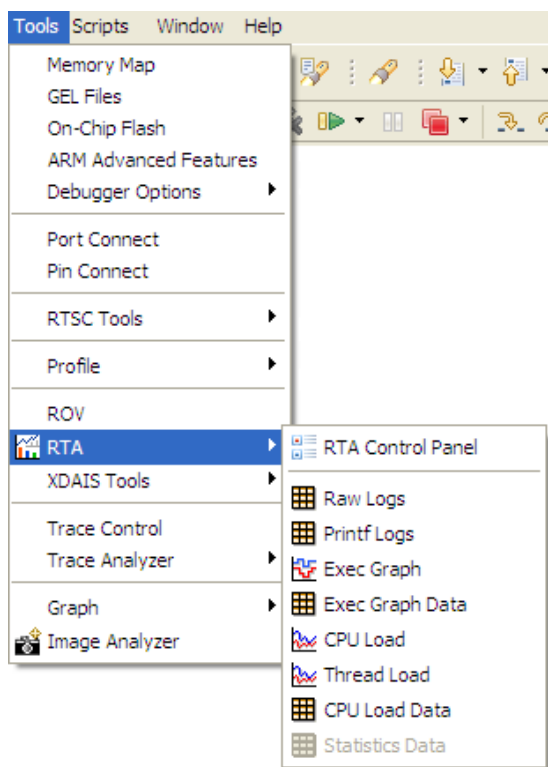


图 17. 实时分析工具

除了 RTA 工具，运行时间目标查看器 (ROV) 能够分析一个 SYS/BIOS 系统。这个工具包含一个程序中所使用的模块的列表并显示与所选模块相关的详细信息。例如，如果任务模块被选中，当程序被暂停或遇到一个断点时，每个任务的模式、自变量、堆栈尺寸和其它属性将被更新。

与 SYS/BIOS 调试工具相关的更多信息，请参见 Code Composer Studio 内 Help > Help Contents 菜单中 Help Contents 的 SYS/BIOS 部分中的 SYS/BIOS API 参考中的内容。

3 下载 SYS/BIOS

SYS/BIOS 被提供作为德州仪器的 Code Composer Studio (CCS) 集成开发环境 (IDE) 的一部分。

SYS/BIOS 也可作为独立组件下载，但是这个下载过程未在本文档中进行解释。要获得与独立组件相关的更多信息，请访问[SYS/BIOS 维基网页](#)。

从[德州仪器嵌入式处理器维基网页](#)内下载最新版本的 CCS。请注意 CCS 的代码尺寸受限版本并不包括 SYS/BIOS，所以您必须下载完全的 DVD 版本。

考虑到 本文档的用途，使用的 CCS 版本为 4.2.4.00033。可使用不同的 CCS 版本，但是它们的特性会有轻微的不同。

要下载并安装 Code Composer Studio:

1. 请访问http://processors.wiki.ti.com/index.php/Download_CCS
2. 要获得最新产品 CCSv4 DVD 镜像，请点击**Download**（下载）按钮

Download latest production CCSv4 DVD image



3. 点击**Download**按钮来下载 .zip 文件。

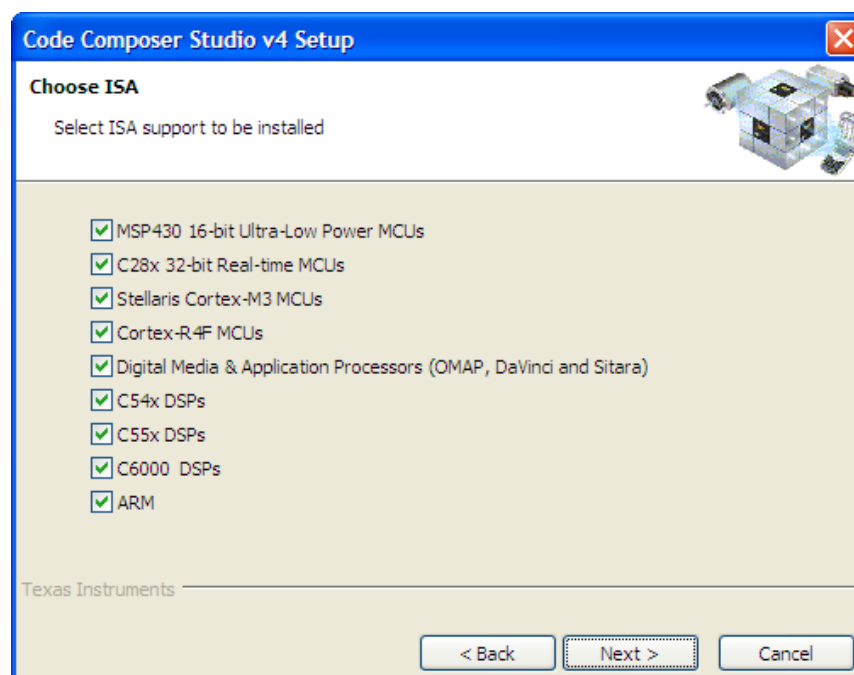
You have been approved to receive this Software.
Click "Download" to proceed.

In a few moments, you will also receive an email with the link to this file.

Download

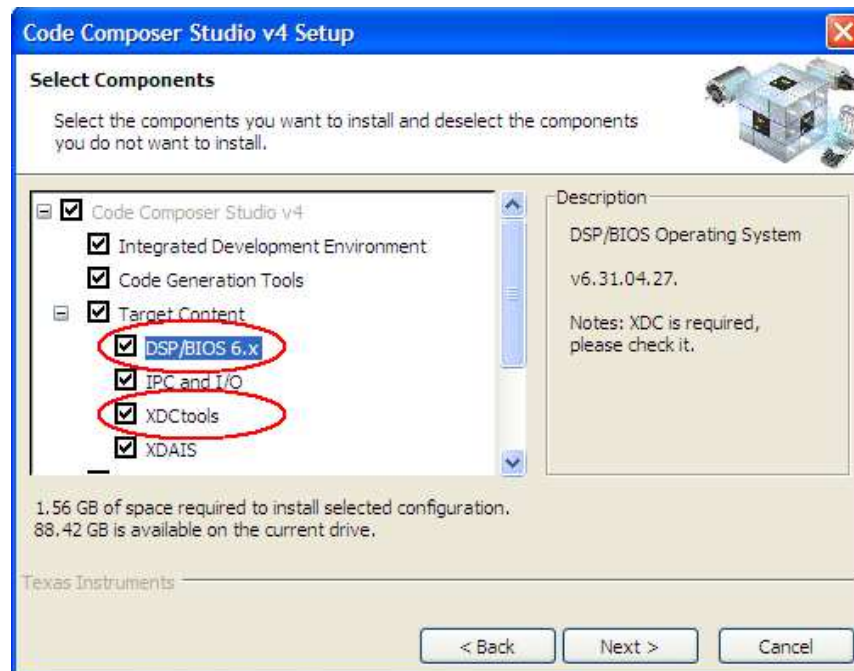
4. 保存 .zip 文件。
5. 提取 .zip 文件。
6. 运行 setup_CCS_x.x.x.x.exe。
7. 您可以使用缺省安装设置或者只安装 Stellaris 系列器件（Stellaris Cortex-M3 和 ARM）所需的组件。

请注意：这个屏幕中列出的 CCS 包还未安装，所以屏幕看起来会有所不同。



8. 点击**Next**（下一步）。

9. 请确保在下一个安装程序窗口中选中的 DSP/BIOS 6.x 和 XDCtools 框。



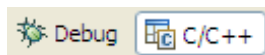
10. 点击**Next**，CCS 安装到所选择的目录。
11. 当安装程序完成时，点击**Finish**（完成）。

4 运行一个现有的 SYS/BIOS 项目

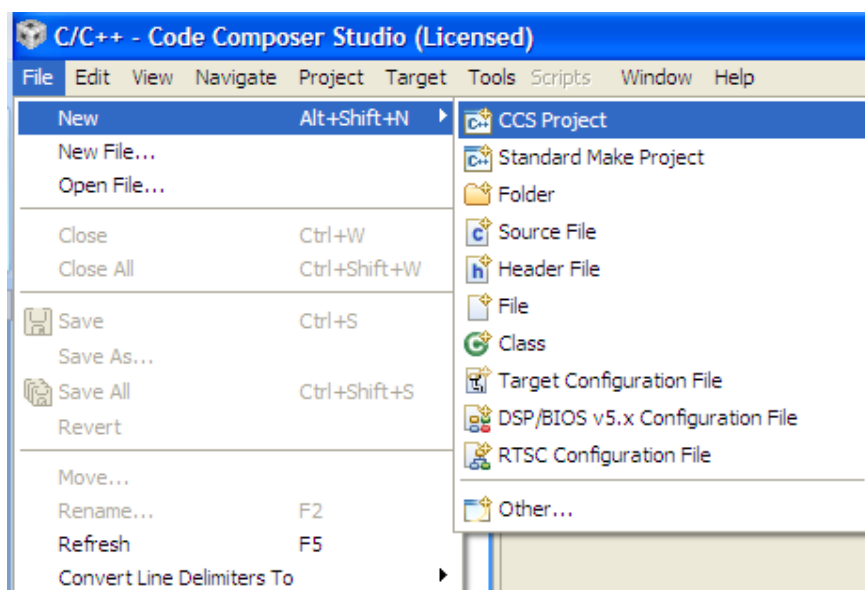
SYS/BIOS 软件包中随附了几个示例程序，这些示例程序展示了内核的特性和功能。以下的步骤列出了打开其中一个示例并在一个目标器件上运行此示例的过程。

考虑到示例项目的用途，使用了 Stellaris DK-LM3S9B96 开发板。然而，可替代任何一个 Stellaris Cortex-M3 器件并用作这个示例的平台。

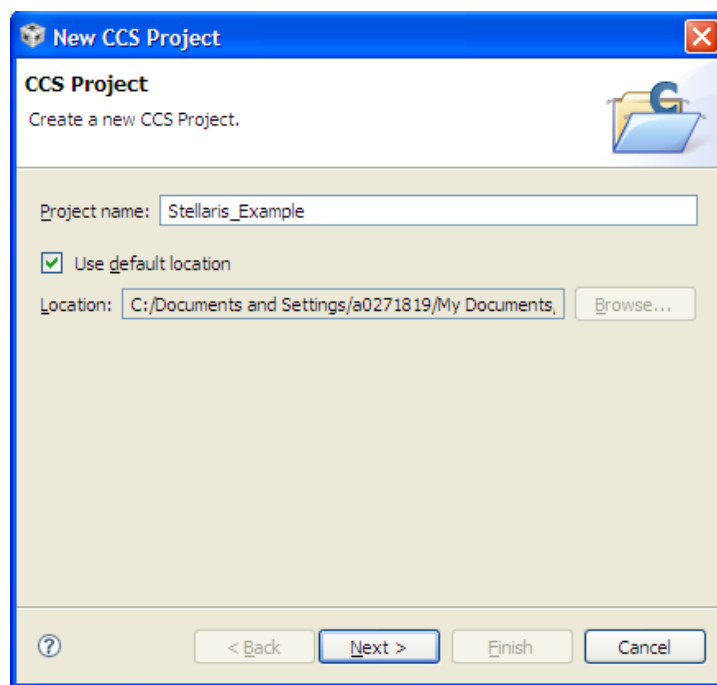
1. 从 Start > All Programs > Texas Instruments > Code Composer Studio v4.2.x > Code Composer Studio v4 中启动 CCS。
2. 请确保在 CCS 中选择了 C/C++。



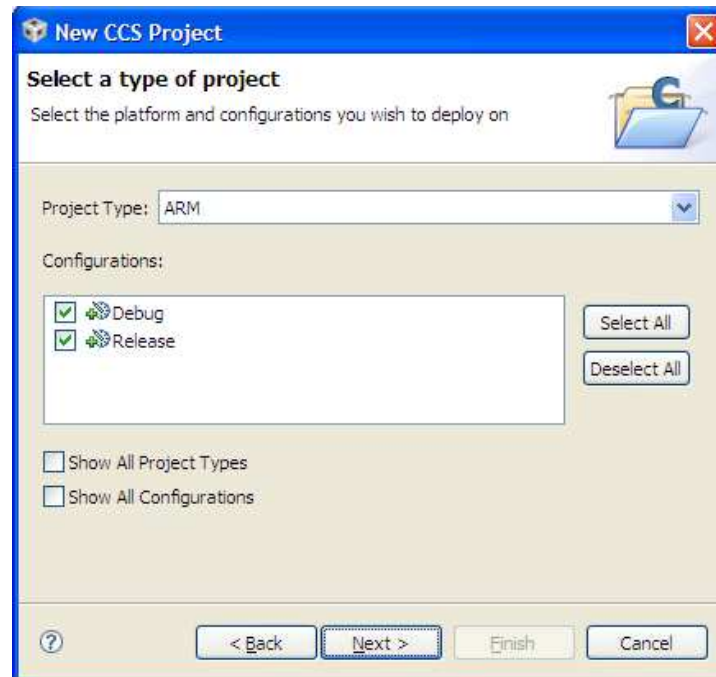
3. 选择 File > New > CCS Project。



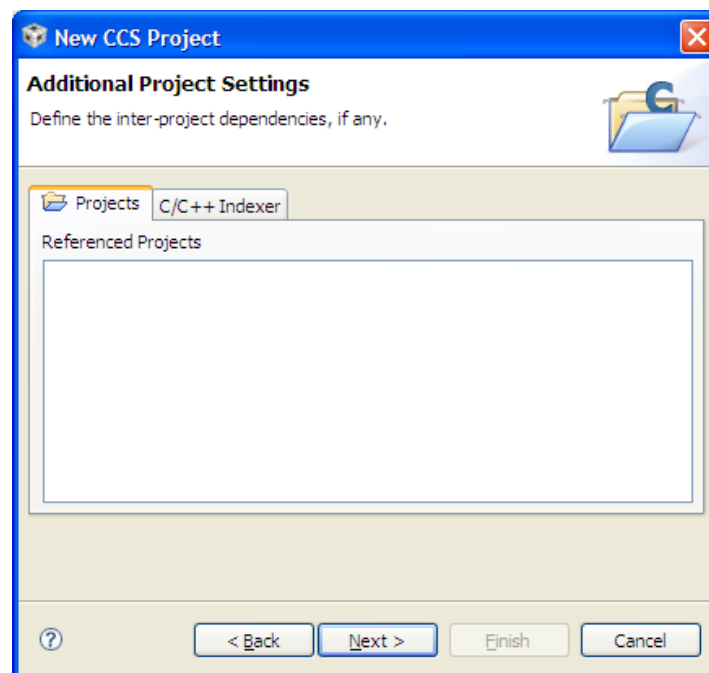
4. 在 Project name (项目名称) 中敲入项目的名称: 字段。



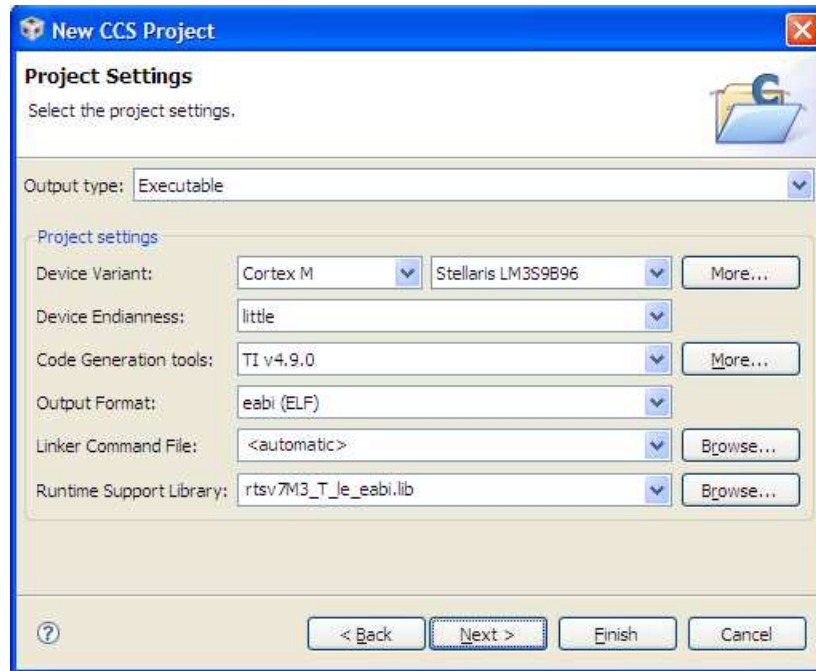
5. 点击**Next**。请确保在 Project Type（项目类型）中选择了 ARM 选项：下拉菜单：



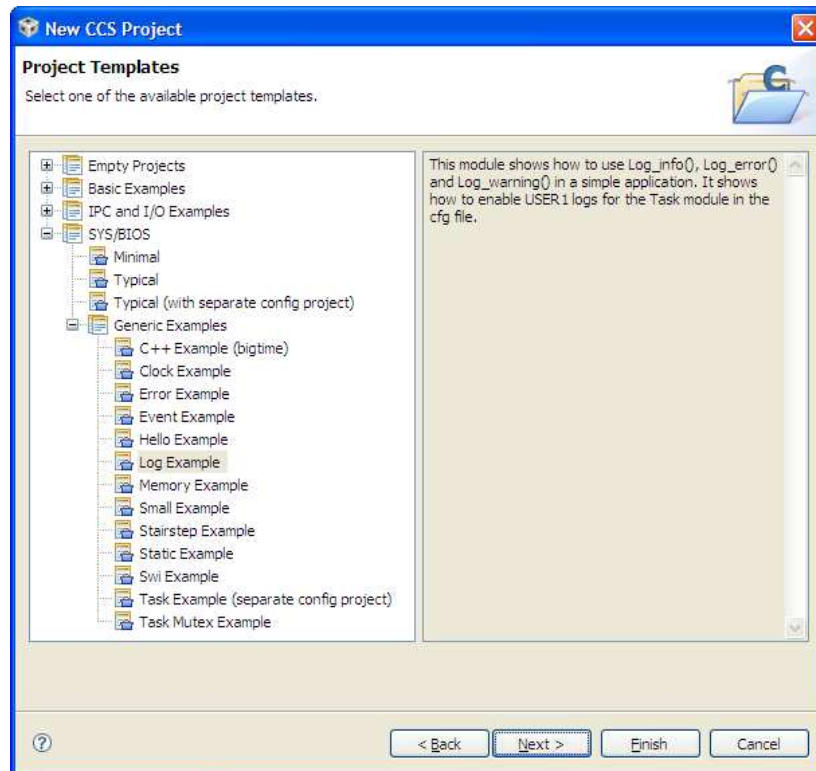
6. 点击**Next**。在这个情况下，将不会有任何参考项目，所以如果在 Referenced Project（参考项目）空间中出现任何项目框，请不要将其选中。



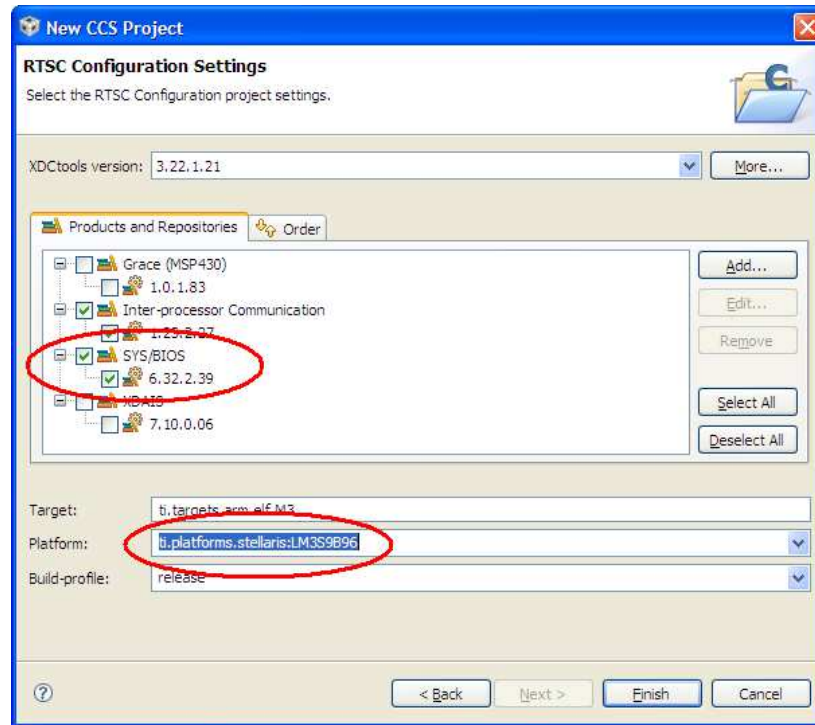
7. 点击**Next**。从左侧器件变量中选择 **Cortex M**：下拉菜单。将您正在使用的 **Stellaris** 器件从下拉菜单中选择至 **Cortex M** 选择的右侧。



8. 点击**Next**。在 **SYS/BIOS** 文件夹下，选择 **Generic Examples**（通用示例）。点击每个示例将显示右侧窗口中示例的说明。选择日志示例。



9. 点击**Next**。请确保在主窗口中 SYS/BIOS 框被选中。此外，从 Platform（平台）中选择适当的平台：下拉菜单。



10. 点击**Finish**。
11. 在项目被载入后，项目可被建立和调试。（请见5.3 节，建立和调试一个项目。）日志示例显示了原始日志和 **Printf** 日志实时分析 (RTA) 工具的特性。（请见2.8 节，调试工具。）

5 创建一个实例 SYS/BIOS 项目

使用提供的项目模板可轻松创建一个新的 SYS/BIOS 项目。有三个类型的模板：

- 最小型模板
- 典型模板
- 具有独立配置项目的典型模板

最小型模板用于只使用静态定义项目的应用。动态存储器分配被禁用以改进代码尺寸和系统性能。此模板使用包含几个 **printf()** 语句的单一任务函数以及一个到 **Task_sleep()** 函数的调用来打开。

针对大多数 SYS/BIOS 应用，典型型模板是一个常见的开始位置。与最小型模板不同，典型型模板可实现动态存储器分配，这使得任务可在运行时间被创建而非在汇编时间内被静态定义。事实上，为了定义单个缺省任务，示例项目与一个到 **Task_create()** 函数的调用一同打开。

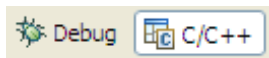
具有独立配置模板的典型型模板除了包含用于第一个项目的配置文件的第二项目，还创建一个典型项目。现在，多个项目可参考单个配置项目以获得它们各自的 SYS/BIOS 设置，从而减少了建立时间并使得几个开发人员可使用相同配置。

考虑到下面列出的示例项目的用途，使用了 **Stellaris DK-LM3S9B96** 开发板。然而，可替换 **Stellaris Cortex-M3** 器件中的任何一个并用作本示例的平台。

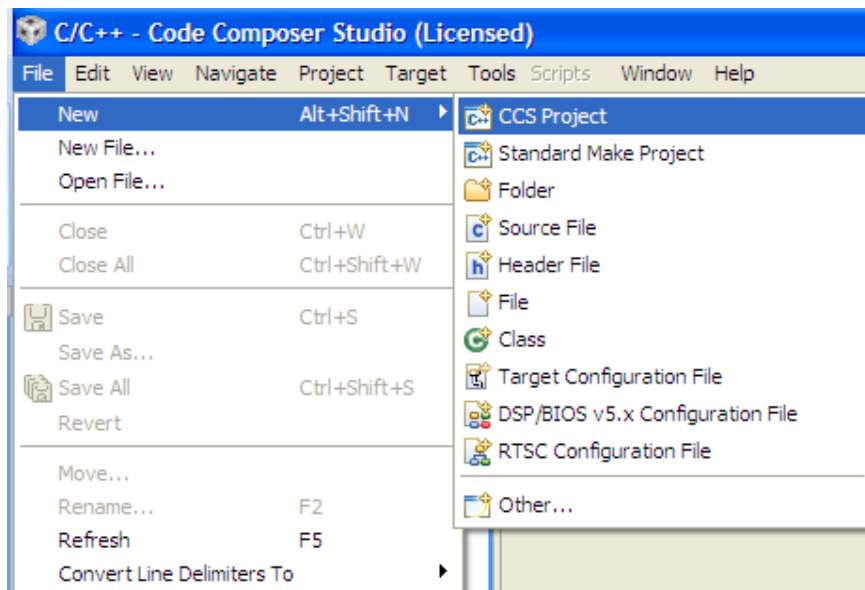
5.1 创建一个新项目

这一部分说明了如何将一个项目模板载入到工作区。

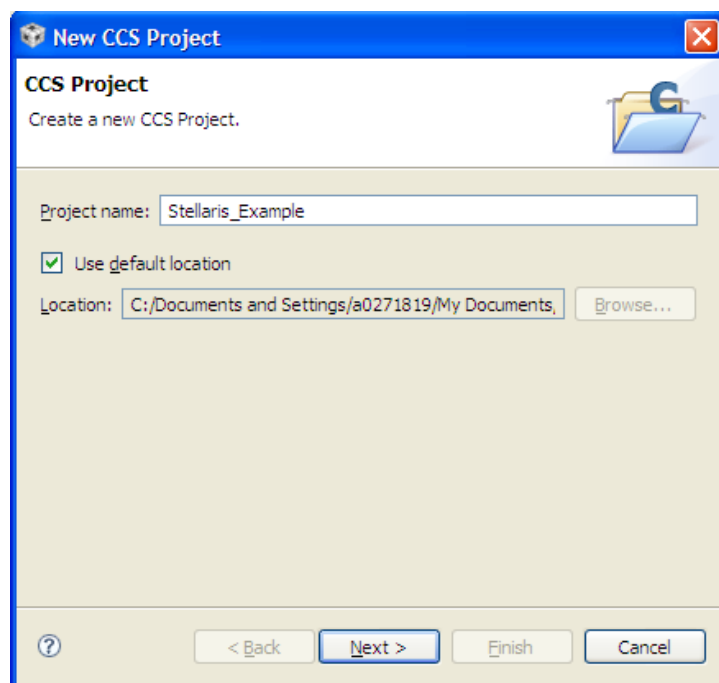
1. 选择 Start > All Programs > Texas Instruments > Code Composer Studio v4.2.x > Code Composer Studio v4 来启动 CCS。
2. 请确保在 CCS 中选择了 C/C++。



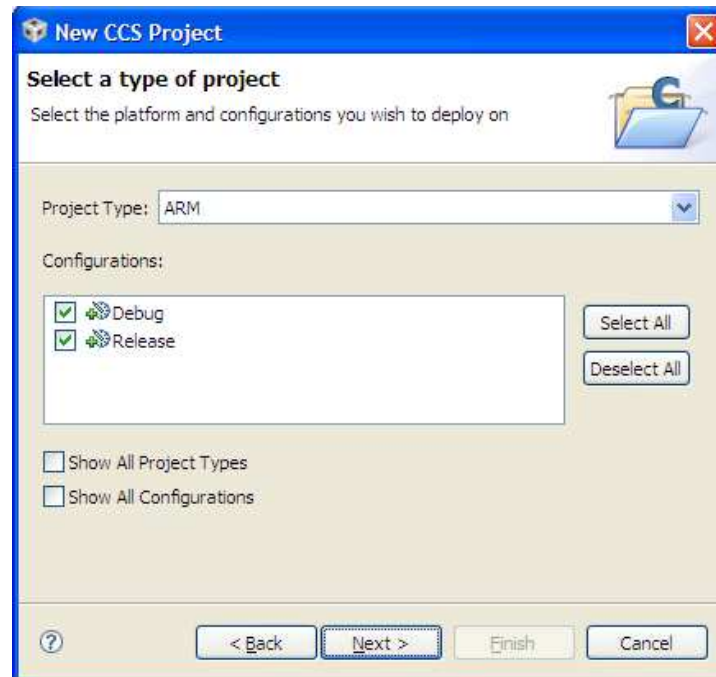
3. 选择 File > New > CCS Project。



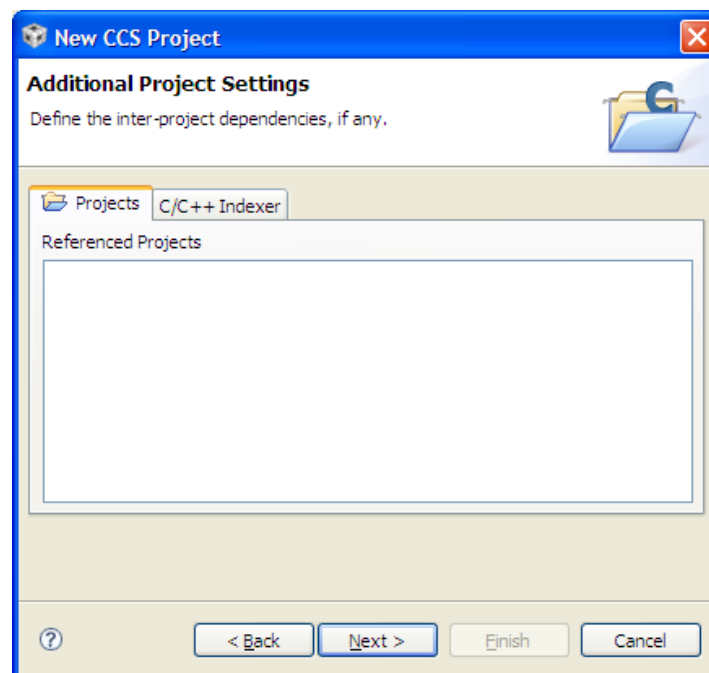
4. 在 Project name (项目名称) 中敲入项目的名称: 字段。



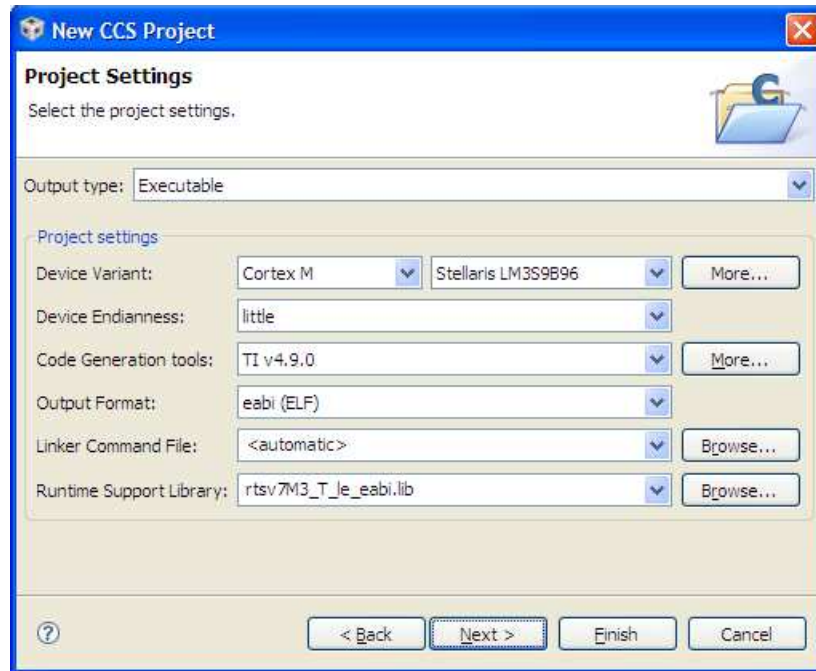
5. 点击**Next**。请确保在 Project Type（项目类型）中选择了 ARM 选项：下拉菜单：



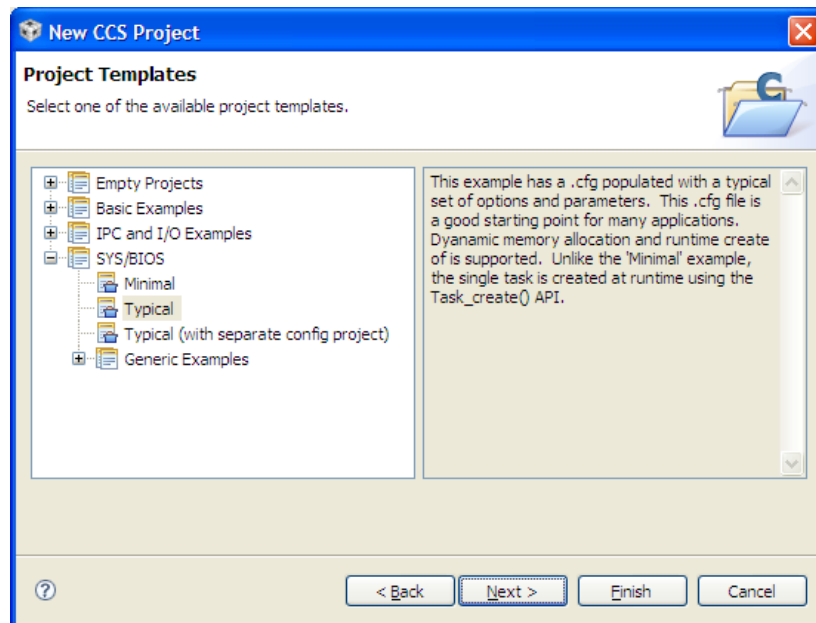
6. 点击**Next**。在这个情况下，将不会有任何参考项目，所以如果在 Referenced Project（参考项目）空间中出现任何项目框，请不要将其选中。



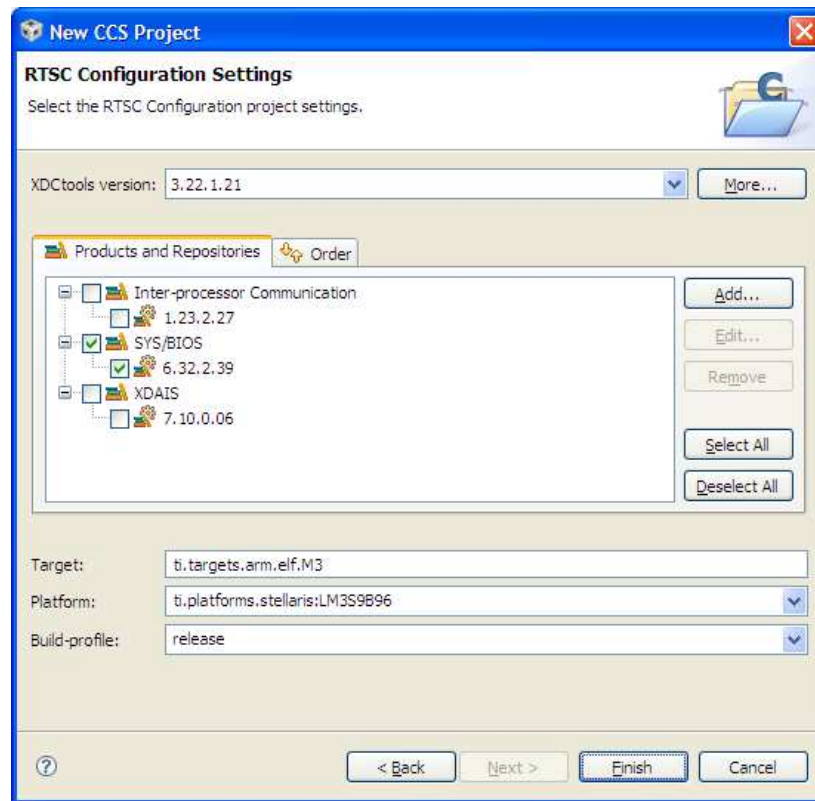
7. 点击**Next**。从左侧器件变量中选择 **Cortex M**：下拉菜单。将您正在使用的 **Stellaris** 器件从下拉菜单中选择至 **Cortex M** 选择的右侧。



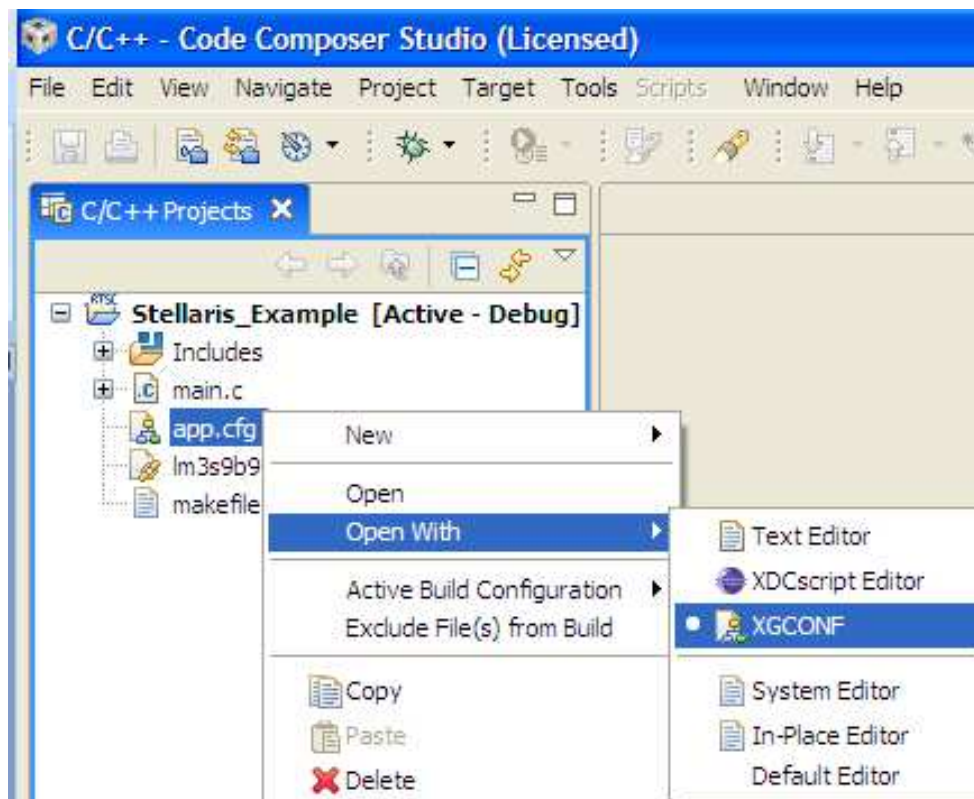
8. 点击**Next**。在 **SYS/BIOS** 文件夹内，选择您希望使用的新项目类型（最小型、典型型或具有独立配置项目的典型型）。这些选项之前在本部分中进行了解释（请见5节，创建一个实例 **SYS/BIOS** 项目）。将这个示例项目选择为典型型。



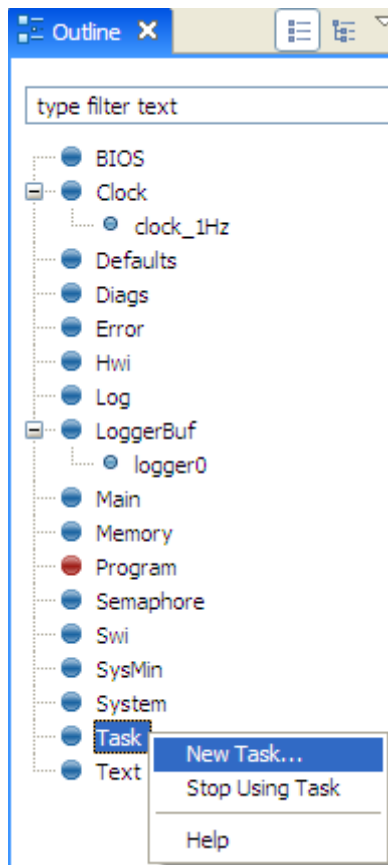
9. 点击**Next**。从 Platform 中选择适当的平台：下拉菜单。



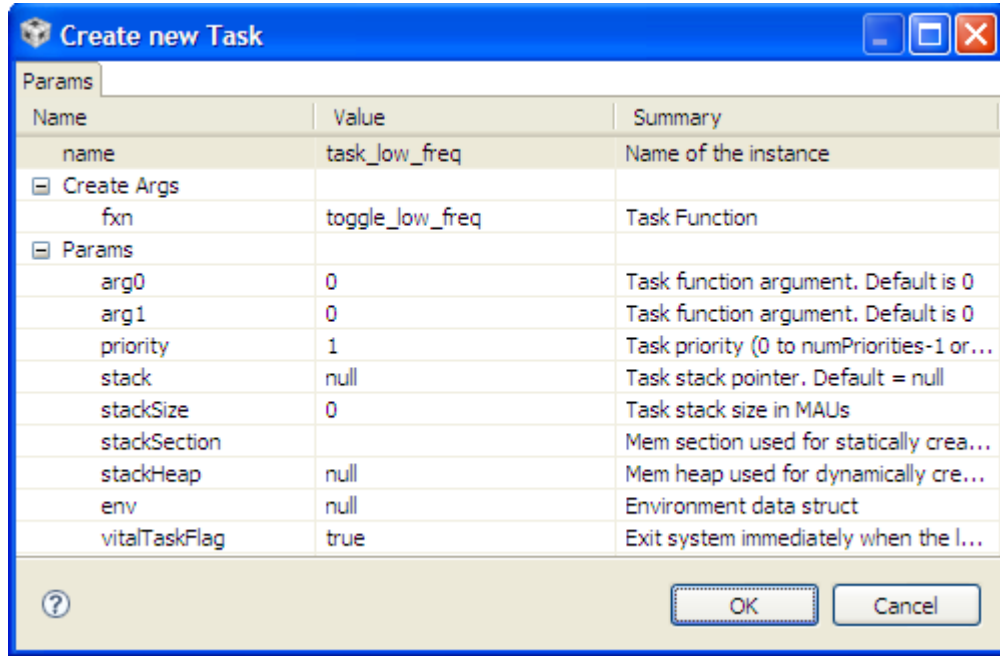
10. 点击**Finish**。
11. 由于我们正在使用 StellarisWare 库来访问 GPIO 外设，您必须按照5 节，创建一个实例 SYS/BIOS 项目中给出的步骤来将 driverlib 添加到项目中。
12. 要编辑程序的属性，请展开项目文件夹，右键点击 app.cfg 文件并选择 Open with > XGCONF。主 GUI 编辑器打开。从这一点开始，可修改 SYS/BIOS 内核的设置。与配置系统有关的信息，请见2.1 节，配置工具。



13. 通过右键点击 Task 并选择 New Task 来创建一个新项目。



14. 如所示的那样编辑新任务实例的域。 点击**OK**。

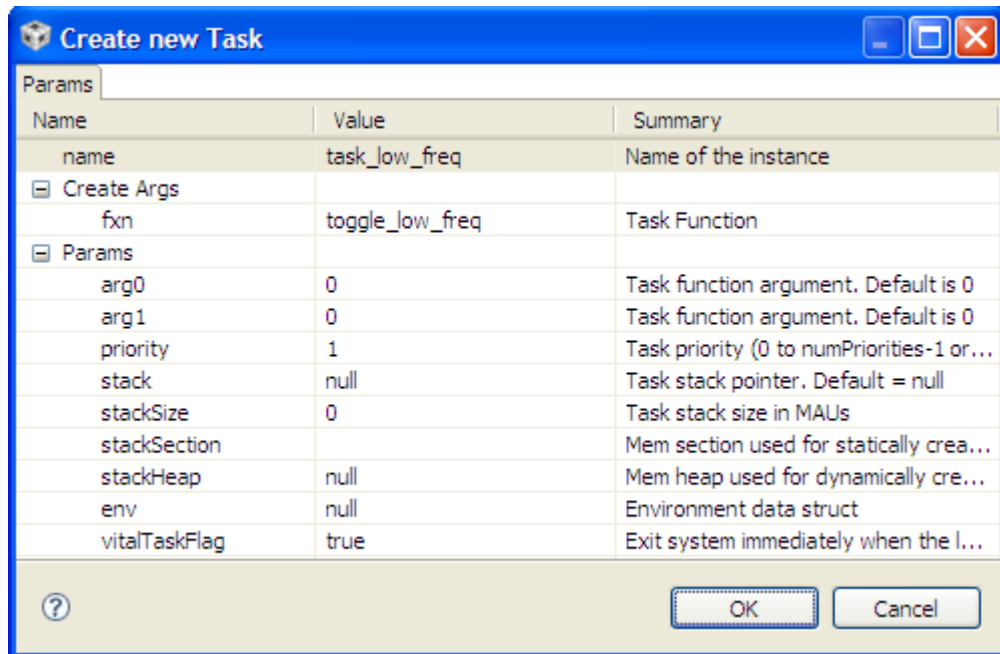


The dialog box titled "Create new Task" contains a table with the following data:

Name	Value	Summary
name	task_low_freq	Name of the instance
Create Args		
fxn	toggle_low_freq	Task Function
Params		
arg0	0	Task function argument. Default is 0
arg1	0	Task function argument. Default is 0
priority	1	Task priority (0 to numPriorities-1 or...
stack	null	Task stack pointer. Default = null
stackSize	0	Task stack size in MAUs
stackSection		Mem section used for statically crea...
stackHeap	null	Mem heap used for dynamically cre...
env	null	Environment data struct
vitalTaskFlag	true	Exit system immediately when the l...

At the bottom right, there are "OK" and "Cancel" buttons.

15. 用以下同样的方法来添加第二个任务实例。 点击**OK**。

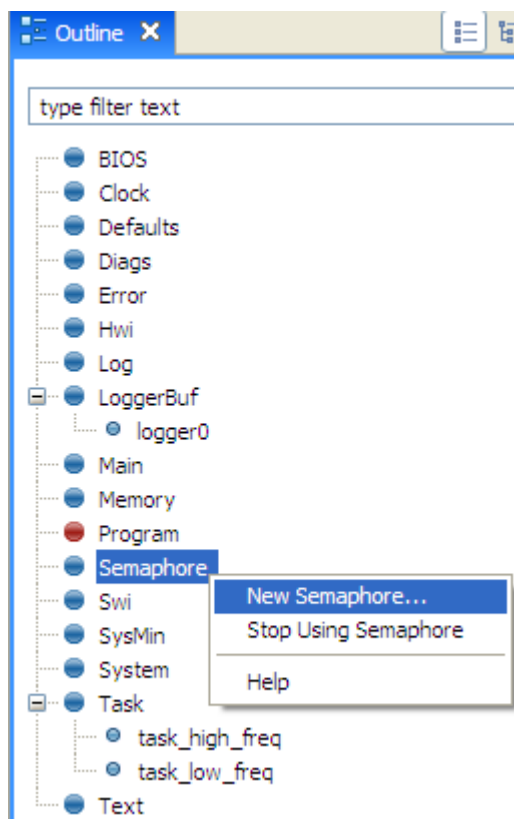


The dialog box titled "Create new Task" contains a table with the following data:

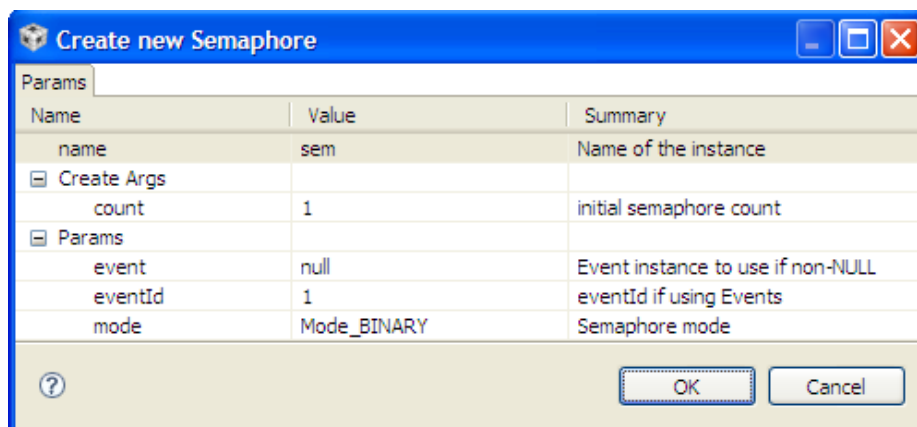
Name	Value	Summary
name	task_low_freq	Name of the instance
Create Args		
fxn	toggle_low_freq	Task Function
Params		
arg0	0	Task function argument. Default is 0
arg1	0	Task function argument. Default is 0
priority	1	Task priority (0 to numPriorities-1 or...
stack	null	Task stack pointer. Default = null
stackSize	0	Task stack size in MAUs
stackSection		Mem section used for statically crea...
stackHeap	null	Mem heap used for dynamically cre...
env	null	Environment data struct
vitalTaskFlag	true	Exit system immediately when the l...

At the bottom right, there are "OK" and "Cancel" buttons.

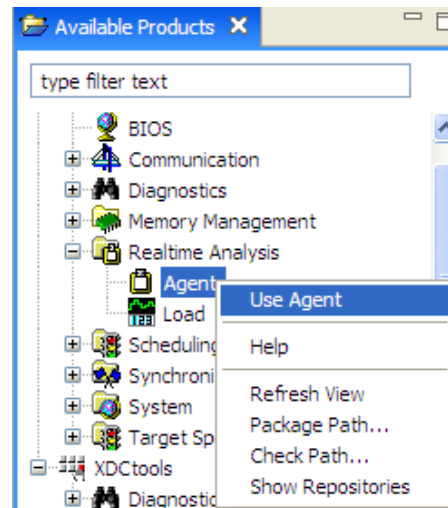
16. 通过右键点击 Semaphore 并选择 New Semaphore 来创建一个新的信号量。



17. 如所示的那样编辑新信号量示例的字段。 点击OK。



18. 添加代理模块来启用实时分析工具进行调试。



19. 将 main.c 文件修改为以下文本。

```
/*
 * ===== main.c =====
 */

#include <xdc/std.h>

#include <xdc/runtime/Error.h>
#include <xdc/runtime/System.h>
#include <xdc/runtime/Log.h>

#include <ti/sysbios/BIOS.h>
#include <ti/sysbios/knl/Task.h>
#include <ti/sysbios/knl/Semaphore.h>

#include "inc/hw_types.h"
#include "inc/hw_memmap.h"
#include "inc/omap3s9b96.h"
#include "driverlib/sysctl.h"
#include "driverlib/gpio.h"

extern Semaphore_Handle sem;

/*
 * ===== Task Functions =====
 */

Void toggle_low_freq()
{
    while(1){
        Log_info0("toggle_low_freq() enter");

        //Pend on the semaphore
        Log_info0("toggle_low_freq() pend sem");
        Semaphore_pend(sem, BIOS_WAIT_FOREVER);

        //Set the value of the GPIO pin low
        Log_info0("toggle_low_freq() pin low");
        GPIOpinWrite(GPIO_PORTA_BASE, GPIO_PIN_6, 0);

        //Sleep
        Log_info0("toggle_low_freq() sleep");
        Task_sleep(10);
    }
}
```

```

        //Post to the semaphore
        Log_info0("toggle_low_freq() post sem");
        Semaphore_post(sem);

        //Sleep
        Log_info0("toggle_low_freq() sleep");
        Task_sleep(10);

        Log_info0("toggle_low_freq() exit");
    }
}

Void toggle_high_freq()
{
    while(1){
        Log_info0("toggle_high_freq() enter");

        //Pend on the semaphore
        Log_info0("toggle_high_freq() pend sem");
        Semaphore_pend(sem, BIOS_WAIT_FOREVER);

        //Set the value of the GPIO pin low
        Log_info0("toggle_high_freq() pin low");
        GPIOPinWrite(GPIO_PORTA_BASE, GPIO_PIN_6, 0);

        //Sleep
        Log_info0("toggle_high_freq() sleep");
        Task_sleep(1);

        //Set the value of the GPIO pin high
        Log_info0("toggle_high_freq() pin high");
        GPIOPinWrite(GPIO_PORTA_BASE, GPIO_PIN_6, GPIO_PIN_6);

        //Post to the semaphore
        Log_info0("toggle_high_freq() post sem");
        Semaphore_post(sem);

        //Sleep
        Log_info0("toggle_high_freq() sleep");
        Task_sleep(1);

        Log_info0("toggle_high_freq() exit");
    }
}

/*
 * ===== main =====
 */

Void main()
{
    Log_info0("enter main()");

    //Enable and configure the GPIO peripheral
    SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOA);
    GPIOPinTypeGPIOOutput(GPIO_PORTA_BASE, GPIO_PIN_6);

    //Initialize GPIO pin PA6 to a value of 0
    GPIOPinWrite(GPIO_PORTA_BASE, GPIO_PIN_6, 0);

    //Start the BIOS scheduler
    BIOS_start(); /* enable interrupts and start SYS/BIOS */
}

```

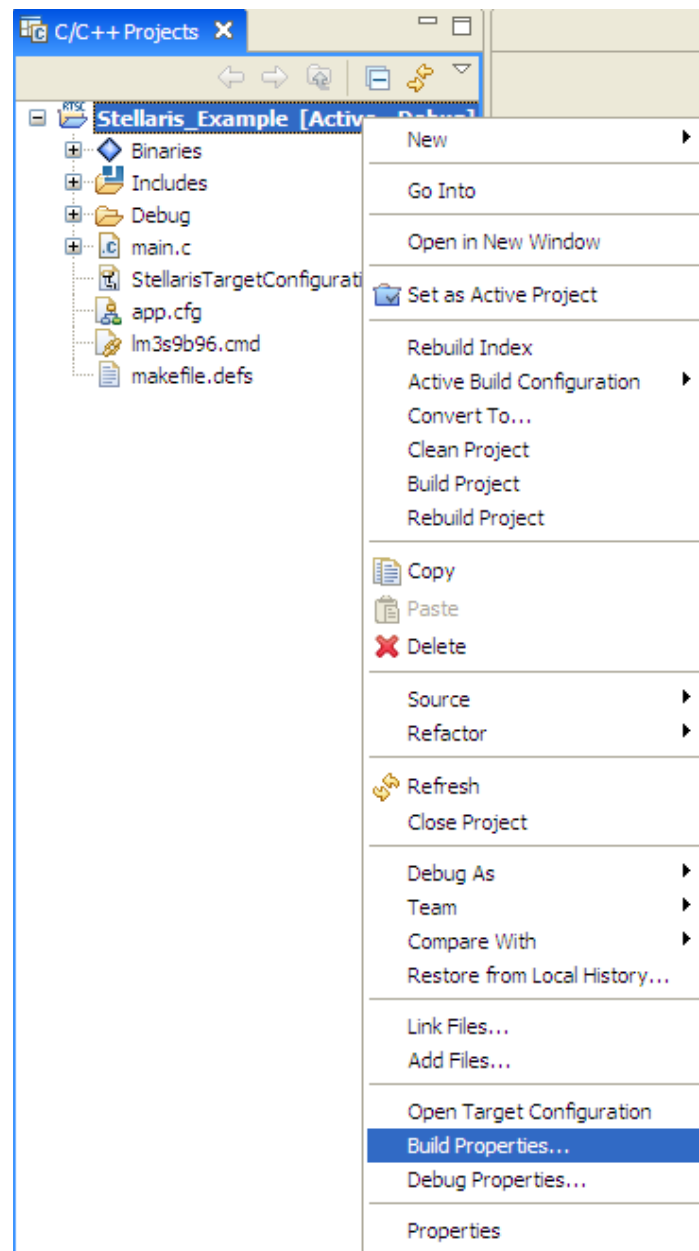
20. 在菜单中选择 File > Save All 来保存项目。

5.2 集成 StellarisWare 库

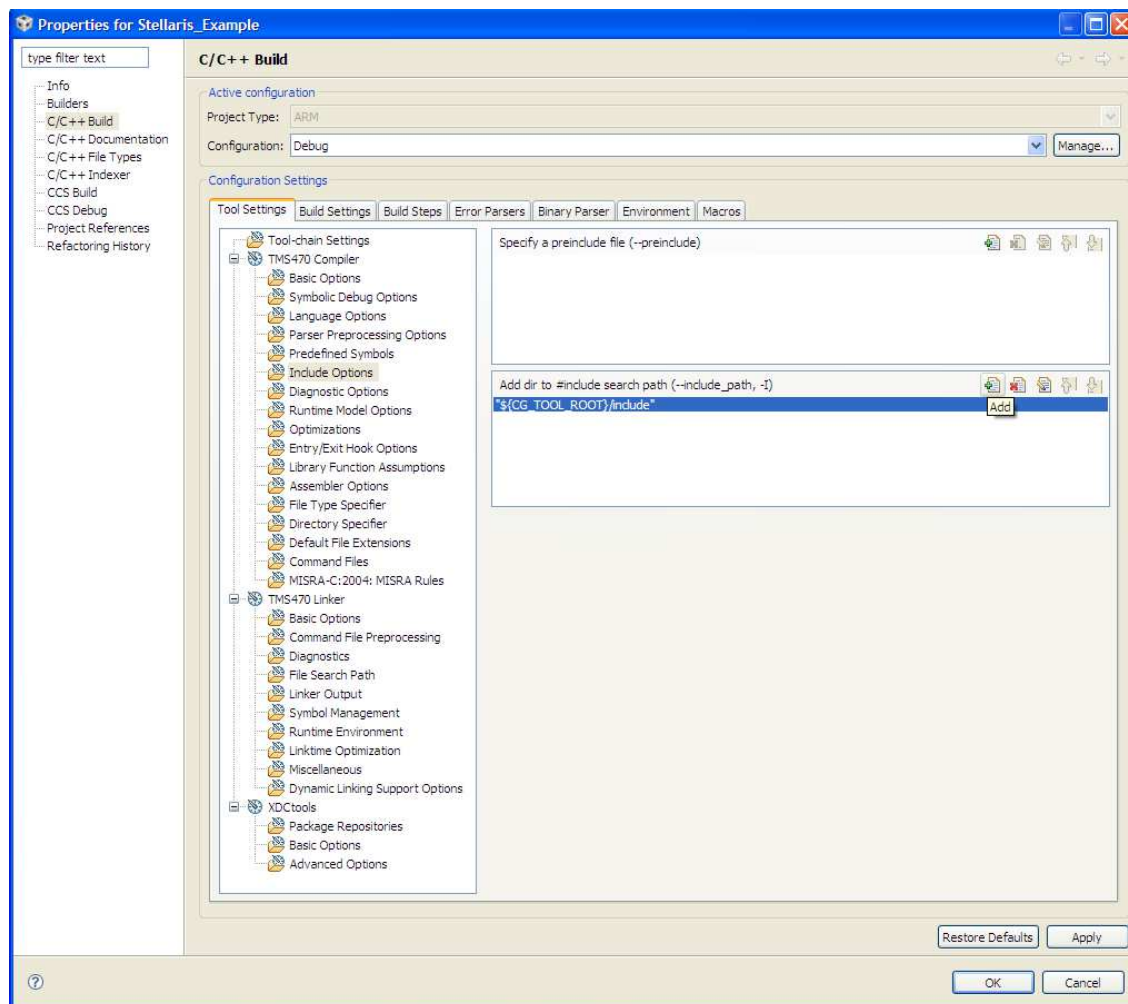
StellarisWare 是一款提供的与 Stellaris 微控制器一起使用的软件包，此软件包可实现位于大多数 Stellaris 器件只读存储器 (ROM) 中的 API 函数的使用。请注意，并不是所有 StellarisWare 函数都是固有线程安全的。换句话说，具有更高优先级的任务在第一个任务已经开始执行函数后调用一个函数有可能导致系统中无法预计的故障。为了改正这些情况类型，应该使用信号量或栅极来保护那些不是线程安全且会被不止一个运行环境调用的函数。

以下的步骤列出了将 driverlib API 库连接到一个 SYS/BIOS 项目的过程。应该重复此过程以包含 usblib (USB) 和 glib (图形) 库。如果您的项目不包括到 StellarisWare 库的调用，您可以跳过本部分并前往[5.3 节](#)，建立和调试一个项目。

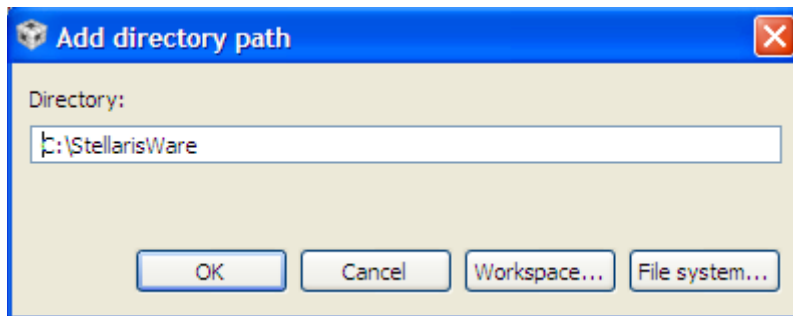
1. 右键点击您的项目的名称并选择 Build Properties (建立属性)。



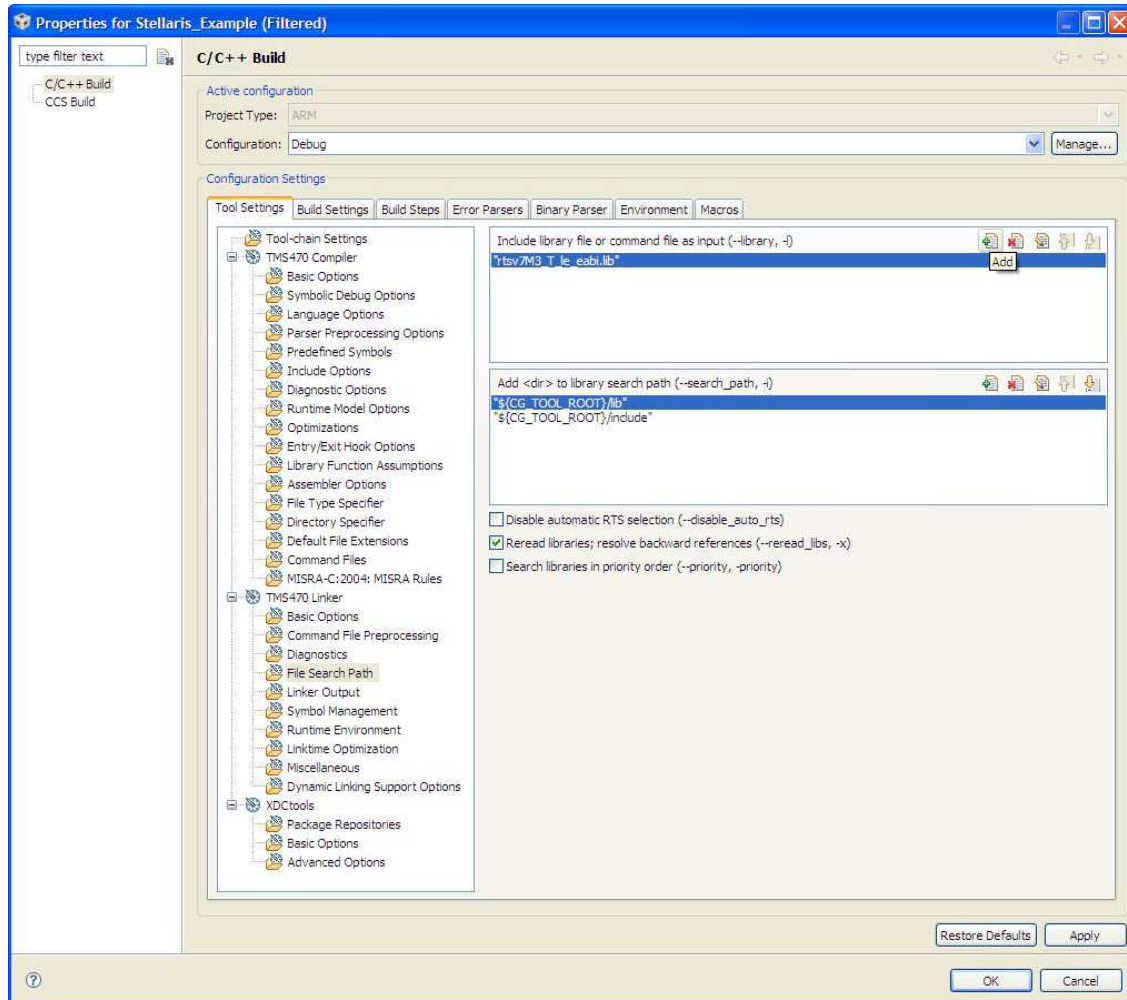
2. 点击 **Tool Setting**（工具设置）标签页内文件夹树中的 **TIMS470** 编译器文件夹下的 **Include Options**（包含选项）文件夹。点击出现在屏幕右侧的下部选择框内的 **Add**（添加）按钮。



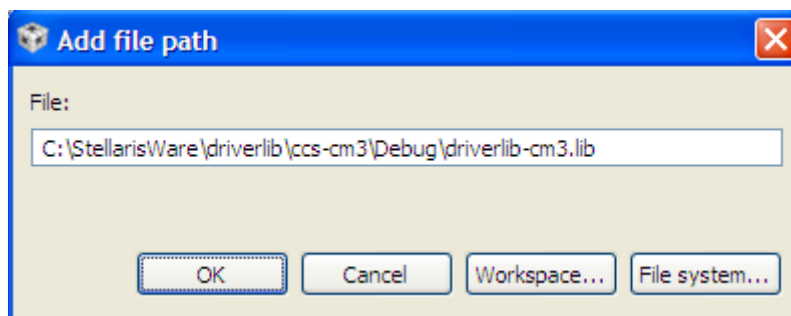
3. 在 **Directory**（目录）中敲入 **C:\StellarisWare**。Add Directory 路径对话框内的字段。点击 **OK**。



4. 点击 **Tool Setting** 标签页内文件夹树中的 **TMS470 连接器** 文件夹下的 **File Search Path**（文件查找路径）文件夹。点击出现在屏幕右侧的上部选择框内的 **Add** 按钮。



5. 在 **File** 中敲入 `C:\StellarisWare\driverlib\ccs-cm3\Debug\driverlib-cm3.lib`：Add file 路径对话框中的字段。



6. 在 **Properties** 窗口中点击 **OK**。

7. 在 main.c 文件中，将下列包含语句添加到代码的顶部：

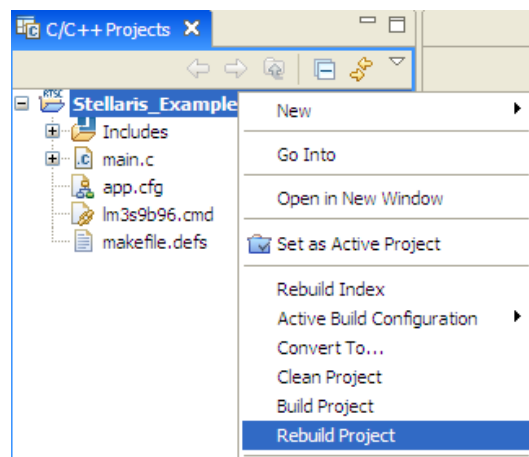
```
#include "inc/hw_types.h"
#include "driverlib/sysctl.h"
```

```
5 #include <xdc/std.h>
6
7 #include <xdc/runtime/Error.h>
8 #include <xdc/runtime/System.h>
9
10 #include <ti/sysbios/BIOS.h>
11
12 #include <ti/sysbios/knl/Task.h>
13
14 #include "inc/hw_types.h"
15 #include "driverlib/sysctl.h"
```

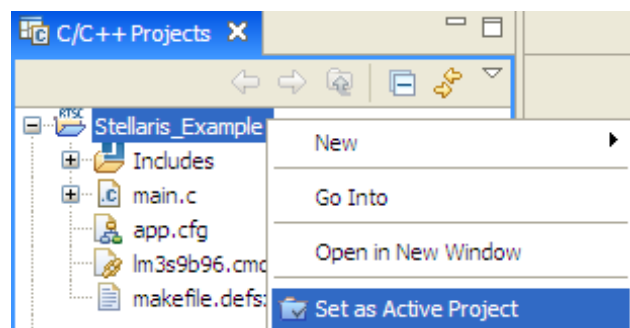
5.3 建立和调试一个项目

本部分列出了建立一个项目并在调试器中启动此项目的步骤。这个部分还强调了几个关键 SYS/BIOS 诊断工具。

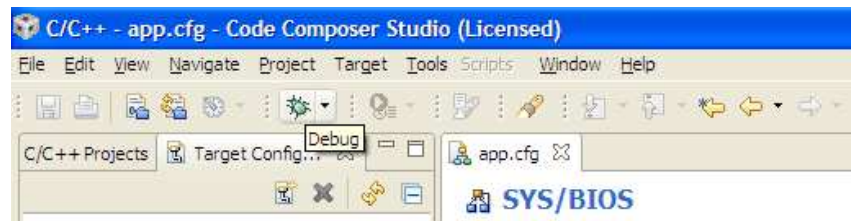
1. 通过右键点击项目并选择 **Rebuild Project**（重建项目）来建立项目。



2. 将 LM3S9B96 开发套件连接到 PC 上。
3. 将项目设定为 **Active Project**（激活的项目）。



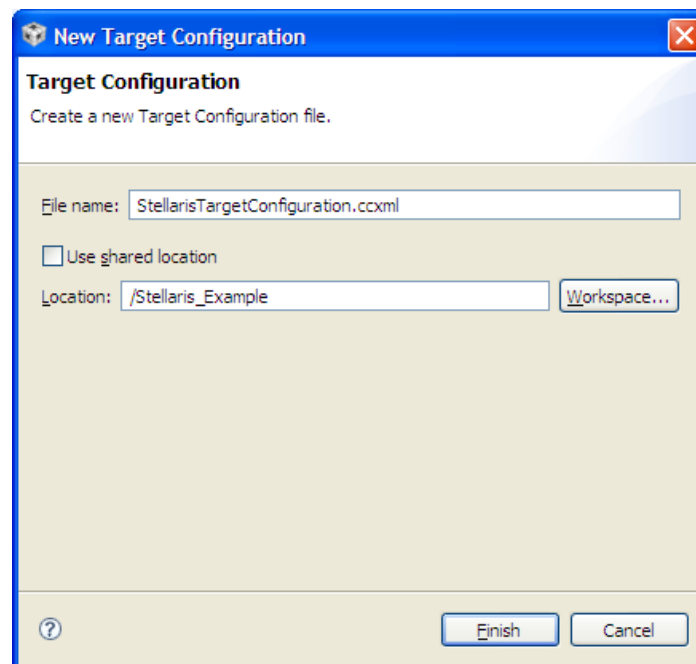
4. 点击绿色的**Debug**按钮来将代码载入器件并启动调试器。



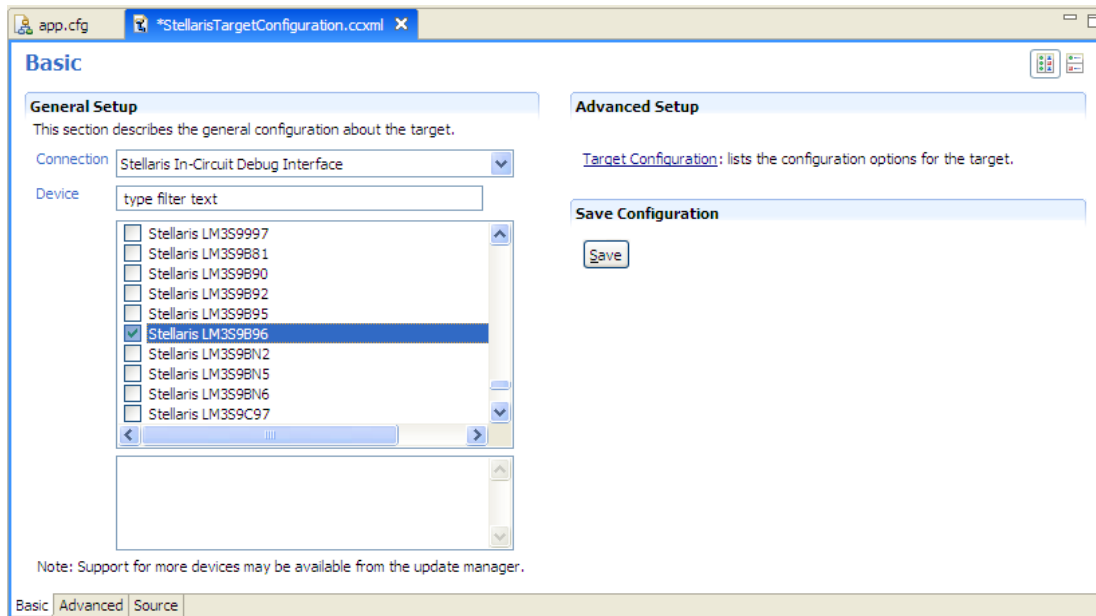
5. 当 Open Target Configuration Dialog（打开目标配置对话框）出现时，点击**Yes**。



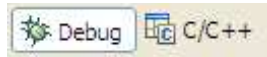
6. 在 File name 内为配置文件敲入一个名称：文本字段。



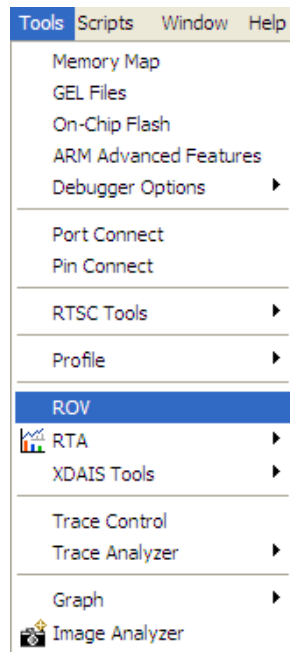
- 在 **Connection**（连接）下拉菜单中选择 **Stellaris In-Circuit Debug Interface**（Stellaris 电路内调试接口）。从 **Device** 窗口中选择您的器件。



- 点击 **Save**。
- 如果 CCS 调试透视图没有自动启用，点击屏幕右上角的 **Debug** 来改变此透视图。



- 点击 **Tools > ROV** 来打开 **Runtime Object Viewer**。当目标被暂停时，这个工具提供与 **SYS/BIOS** 模块相关的信息。



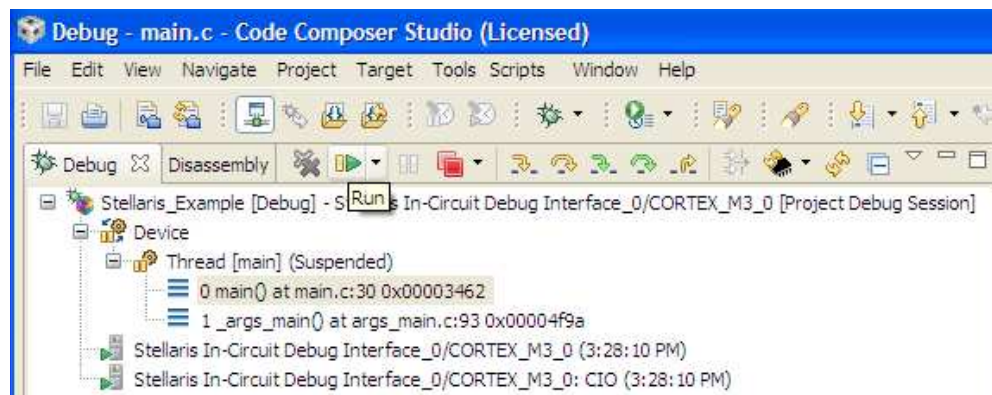
11. 如所示的那样在每个任务中设定一个断点。

```

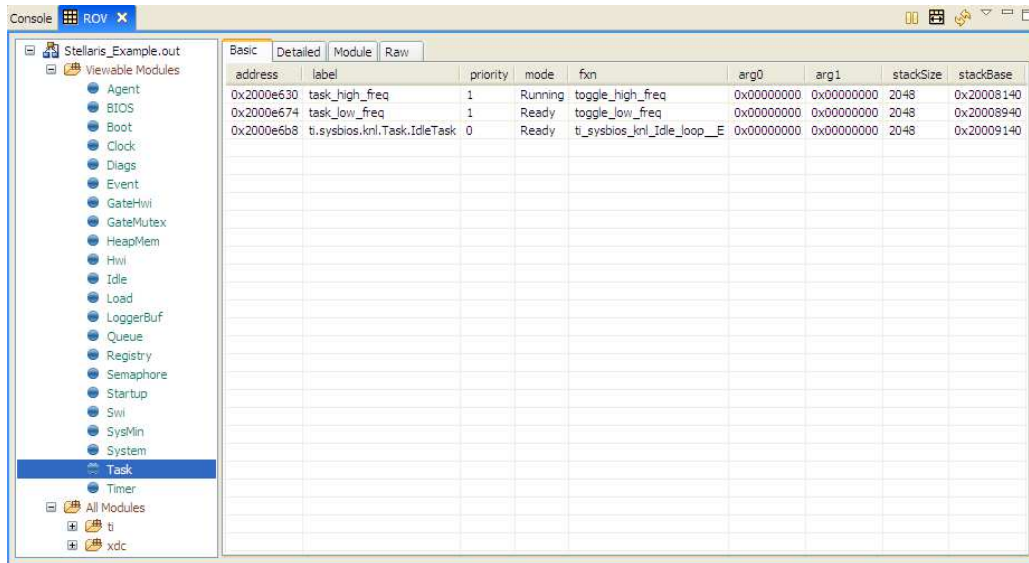
27 Void toggle_low_freq()
28 {
29     while(1){
30         Log_info0("toggle_low_freq() enter");
31
32         //Pend on the semaphore
33         Log_info0("toggle_low_freq() pend sem");
34         Semaphore_pend(sem, BIOS_WAIT_FOREVER);
35
36         //Set the value of the GPIO pin low
37         Log_info0("toggle_low_freq() pin low");
38         GPIO_PORTA_DATA_R &= ~(GPIO_PIN_6);
39
40         //Sleep
41         Log_info0("toggle_low_freq() sleep");
42         Task_sleep(10);
43
44         //Post to the semaphore
45         Log_info0("toggle_low_freq() post sem");
46         Semaphore_post(sem);
47
48         //Sleep
49         Log_info0("toggle_low_freq() sleep");
50         Task_sleep(10);
51
52         Log_info0("toggle_low_freq() exit");
53     }
54 }
55
56 Void toggle_high_freq()
57 {
58     while(1){
59         Log_info0("toggle_high_freq() enter");
60

```

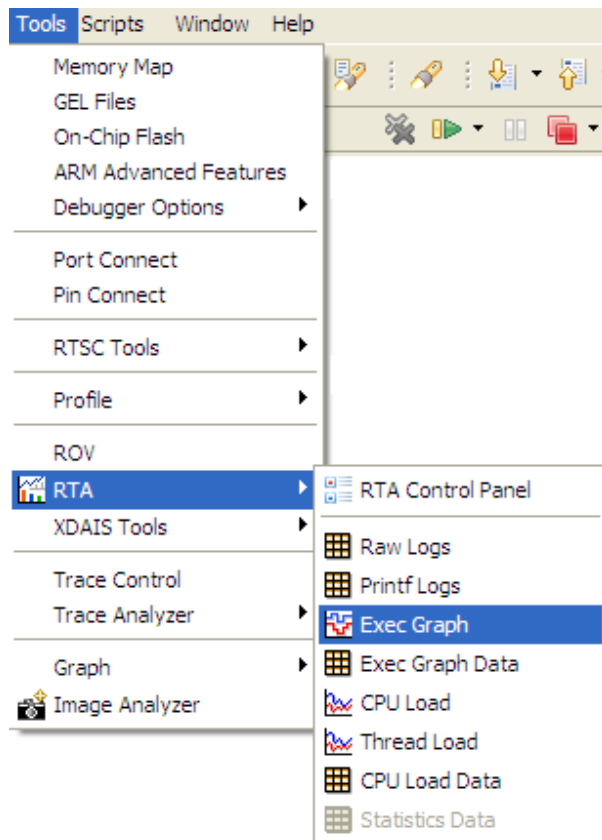
12. 运行调试程序。



13. 当目标在第一个断点上被暂停时，打开 ROV 工具并观察任务细节。每个任务的状态与每个任务的其它细节一同显示。



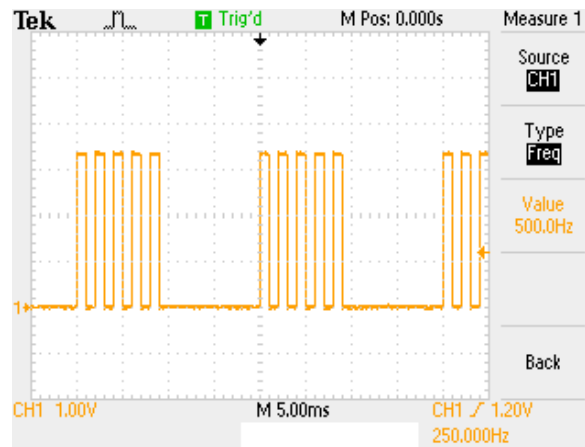
14. 寻找 ROV 工具内的其它可用选项。
15. 前往 **Tools > RTA > Exec Graph** 来打开实时分析工具中的一个。RTA 工具提供包括日志、执行图和载入数据在内的与系统执行相关的诊断信息。



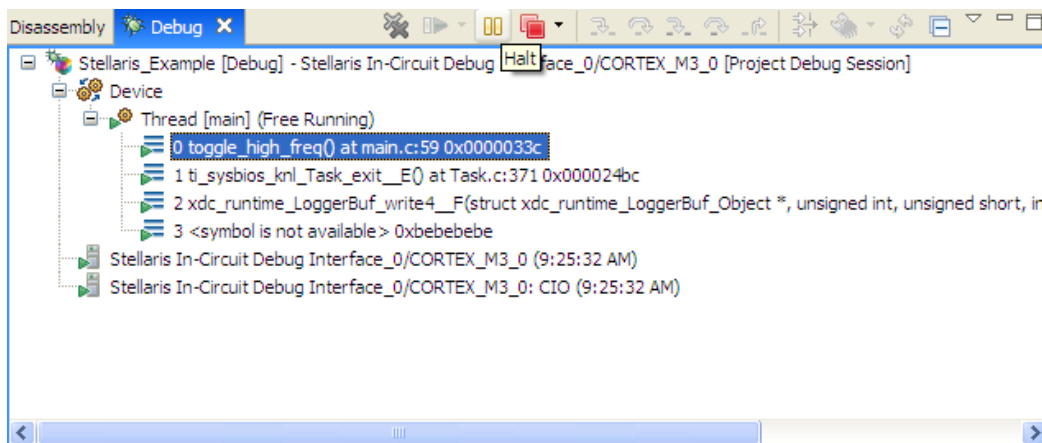
16. 为了绕过针对 ROV 工具而放置在代码内的断点，从 Run 按钮旁的下拉选项中选择 Free Run（自由运行）。



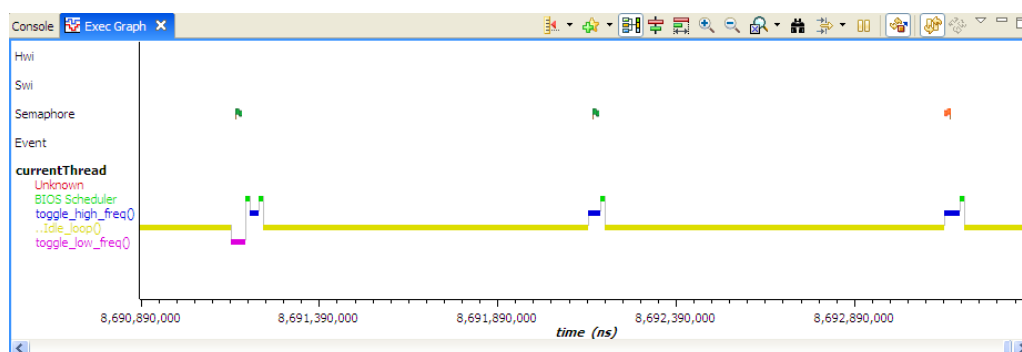
17. 在这一点上，以下的输出应该出现在开发板的 GPIO 引脚 PA6 上。高频振荡由 toggle_high_freq 任务创建，而较低频率方波包迹由 toggle_low_freq 任务创建。



18. 暂停目标。



19. 在 Exec Graph 窗口中观察输出。



20. 探究 RTA Exec Graph Tool 内的可用选项。

要获得与 SYS/BIOS 调试工具相关的更多信息，请参见 Code Composer Studio 内 Help > Help Contents 菜单中 Help Contents 的 SYS/BIOS 部分中的 SYS/BIOS 用户手册中的内容。

6 结论

SYS/BIOS 内核为开发需要精确定时和工具的应用提供了基础。SYS/BIOS 软件包的调试特性对不同执行阶段上的程序执行状态有深入了解。图形配置工具可实现对内核设置以及模块的快速且便捷的操作。创建一个在 Stellaris Cortex-M3 器件上运行的项目是一个简单直接的过程，所提供的 SYS/BIOS 项目模板的使用使得此过程更加容易。总的来说，SYS/BIOS 为设计实时应用提供一个稳健耐用的、可移植结构。

7 参考书目

下列相关文档可从 Stellaris 网站<http://www.ti.com/stellaris>内获得。

- Stellaris LM3S9B96 微控制器数据表（文献编号 [SPMS182](#)）
- StellarisWare 驱动程序库。可从<http://www.ti.com/tool/sw-drl>内下载。
- Stellaris® 外设驱动程序库用户指南，版号 SW-DRL-UG（文献编号 [SPMU019](#)）。
- StellarisWare 软件。可从<http://www.ti.com/tool/sw-lm3s>内下载。

德州仪器嵌入式处理器维基网页 (processors.wiki.ti.com) 包含下列 SYS/BIOS 资源：

- [SYS/BIOS 概述](#)
- [SYS/BIOS 入门指南](#)
- [SYS/BIOS 在线培训](#)
- [用于 Stellaris 器件的 SYS/BIOS](#)
- [SYS/BIOS 入门研讨会](#)

在互联网上提供额外的资源：

- [SYS/BIOS E2E 论坛](#)
- [Stellaris 尺寸基准](#)
- [Stellaris 时序基准](#)

重要声明

德州仪器(TI) 及其下属子公司有权根据 JESD46 最新标准, 对所提供的产品和服务进行更正、修改、增强、改进或其它更改, 并有权根据 JESD48 最新标准中止提供任何产品和服务。客户在下订单前应获取最新的相关信息, 并验证这些信息是否完整且是最新的。所有产品的销售都遵循在订单确认时所提供的TI 销售条款与条件。

TI 保证其所销售的组件的性能符合产品销售时 TI 半导体产品销售条件与条款的适用规范。仅在 TI 保证的范围内, 且 TI 认为 有必要时才会使用测试或其它质量控制技术。除非适用法律做出了硬性规定, 否则没有必要对每种组件的所有参数进行测试。

TI 对应用帮助或客户产品设计不承担任何义务。客户应对其使用 TI 组件的产品和应用自行负责。为尽量减小与客户产品和应用相关的风险, 客户应提供充分的设计与操作安全措施。

TI 不对任何 TI 专利权、版权、屏蔽作品权或其它与使用了 TI 组件或服务的组合设备、机器或流程相关的 TI 知识产权中授予 的直接或隐含权限作出任何保证或解释。TI 所发布的与第三方产品或服务有关的信息, 不能构成从 TI 获得使用这些产品或服务 的许可、授权、或认可。使用此类信息可能需要获得第三方的专利权或其它知识产权方面的许可, 或是 TI 的专利权或其它 知识产权方面的许可。

对于 TI 的产品手册或数据表中 TI 信息的重要部分, 仅在没有对内容进行任何篡改且带有相关授权、条件、限制和声明的情况 下才允许进行复制。TI 对此类篡改过的文件不承担任何责任或义务。复制第三方的信息可能需要服从额外的限制条件。

在转售 TI 组件或服务时, 如果对该组件或服务参数的陈述与 TI 标明的参数相比存在差异或虚假成分, 则会失去相关 TI 组件 或服务的所有明示或暗示授权, 且这是不正当的、欺诈性商业行为。TI 对任何此类虚假陈述均不承担任何责任或义务。

客户认可并同意, 尽管任何应用相关信息或支持仍可能由 TI 提供, 但他们将独力负责满足与其产品及其应用中使用的 TI 产品 相关的所有法律、法规和安全相关要求。客户声明并同意, 他们具备制定与实施安全措施所需的全部专业技术和知识, 可预见 故障的危险后果、监测故障及其后果、降低有可能造成人身伤害的故障的发生机率并采取适当的补救措施。客户将全额赔偿因 在此类安全关键应用中使用任何 TI 组件而对 TI 及其代理造成的任何损失。

在某些场合中, 为了推进安全相关应用有可能对 TI 组件进行特别的促销。TI 的目标是利用此类组件帮助客户设计和创立其特 有的可满足适用的功能安全性标准和要求的终端产品解决方案。尽管如此, 此类组件仍然服从这些条款。

TI 组件未获得用于 FDA Class III (或类似的生命攸关医疗设备) 的授权许可, 除非各方授权官员已经达成了专门管控此类使 用的特别协议。

只有那些 TI 特别注明属于军用等级或“增强型塑料”的 TI 组件才是设计或专门用于军事/航空应用或环境的。购买者认可并同 意, 对并非指定面向军事或航空航天用途的 TI 组件进行军事或航空航天方面的应用, 其风险由客户单独承担, 并且由客户独 力负责满足与此类使用相关的所有法律和法规要求。

TI 已明确指定符合 ISO/TS16949 要求的产品, 这些产品主要用于汽车。在任何情况下, 因使用非指定产品而无法达到 ISO/TS16949 要 求, TI 不承担任何责任。

	产品		应用
数字音频	www.ti.com.cn/audio	通信与电信	www.ti.com.cn/telecom
放大器和线性器件	www.ti.com.cn/amplifiers	计算机及周边	www.ti.com.cn/computer
数据转换器	www.ti.com.cn/dataconverters	消费电子	www.ti.com.cn/consumer-apps
DLP® 产品	www.dlp.com	能源	www.ti.com.cn/energy
DSP - 数字信号处理器	www.ti.com.cn/dsp	工业应用	www.ti.com.cn/industrial
时钟和计时器	www.ti.com.cn/clockandtimers	医疗电子	www.ti.com.cn/medical
接口	www.ti.com.cn/interface	安防应用	www.ti.com.cn/security
逻辑	www.ti.com.cn/logic	汽车电子	www.ti.com.cn/automotive
电源管理	www.ti.com.cn/power	视频和影像	www.ti.com.cn/video
微控制器 (MCU)	www.ti.com.cn/microcontrollers		
RFID 系统	www.ti.com.cn/rfidsys		
OMAP应用处理器	www.ti.com.cn/omap		
无线连通性	www.ti.com.cn/wirelessconnectivity	德州仪器在线技术支持社区	www.deyisupport.com

邮寄地址: 上海市浦东新区世纪大道 1568 号, 中建大厦 32 楼 邮政编码: 200122
Copyright © 2013 德州仪器 半导体技术(上海)有限公司