CC3220 学习笔记---SimpleLinkMCUSDK 用户指南

SimpleLink?MCUSDK 介绍

SimpleLink?MCUSoftwareDevelopmentKit（SDK 软件开发工具包）是一套软件开发工具，使工程师可以针对德州仪器公司（TI）的一系列微控制器开发应用程序。它是一个功能强大的软件工具包，通过对基本软件组件进行包装，易用的示例以及易用的软件开发包，提供了连贯一致的软件体验。

如果你还没有安装 SDK 及执行安装初始步骤，请参考【QuickStartGuide】位于安装目录【\ti\simplelink_cc32xx_sdk_1_30_01_03\docs\simplelink_mcu_sdk\Quick_Start_Guide.html】。

文档与支持

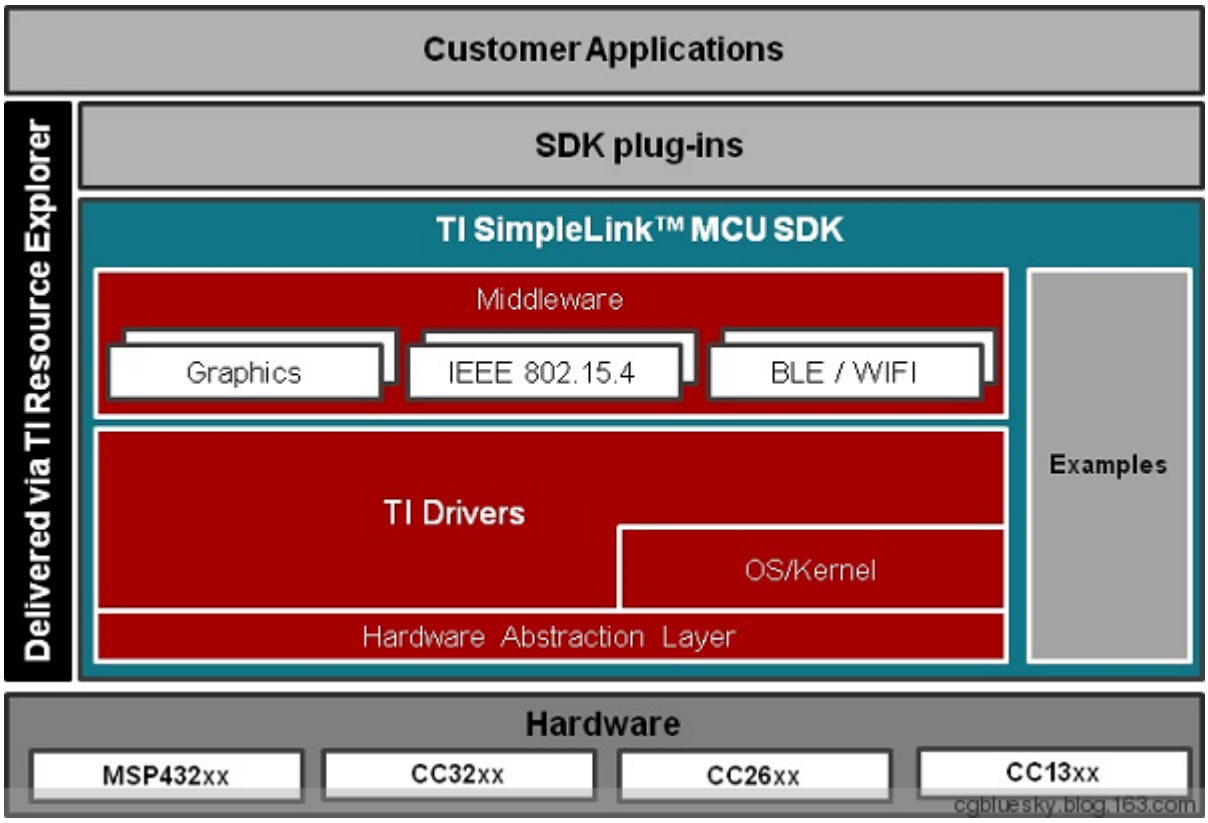
链接至 SDK 文档，其组件在【DocumentationOverview】中提供，位于安装目录【ti\simplelink_cc32xx_sdk_1_30_01_03\docs\Documentation_Overview.html】

SimpleLinkAcademy 提供了循序渐进的讨论及视频教程来介绍 SimpleLinkSDK 的组件。

TIE2ECommunity 支持此 SDK。

SDK 组件

SDK 组件一起用于构建应用程序，SDK 组件相互之间的关联如下图所示：



从架构图底部开始，组件如下：

- **Hardware**：TISimpleLink 系列：
- MSP432：围绕 ARMCortexM4 内核构建的低功耗 MCU，为物联网传感器结点优化。
- CC32xx：基于 ARMCortexM4 的 MCU，整合了 WiFi 和安全功能。
- CC26xx/CC13xx：基于 ARMCortexM3 的低功耗无线 MCU，用于高性能 RF 应用程序。

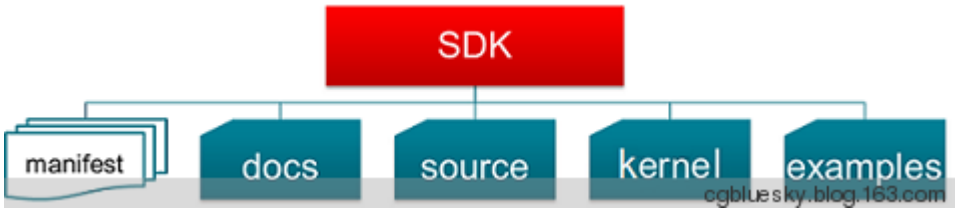
  它们支持一系列无线标准。
- **HardwareAbstractionLayer(HAL 硬件抽象层)**：抽象写硬件寄存器的 C 函数。这是驱

  动和操作系统内核用来访问硬件功能的层。

- **OS/Kernel（操作系统/内核）：** 内核提供定时器和任务调试等服务。所有 TISimpleLinkSDK 都支持 TI-RTOS 内核。一些 SDK 还支持其它 RTOS 内核（如 FreeRTOS）。
- DriverPortingLayer(DPL 驱动程序移植层)抽象了驱动接口。驱动程序使用 RTOS 功能如时钟、中断、互斥量及信号量。通过抽象这些功能，应用程序使用 TI 驱动时无需依赖于 TI-RTOS，而可使用 FreeRTOS 内核代替。
- POSIX 层抽象了应用程序使用的 RTOS 内核的功能性。POSIX 层使得示例及用户应用程序很容易地移植到不同的内核。使用此层是可选项（参考：“ChoosingWhethertoUsePOSIX”章节）。
- **TIDriverAPI**：所有 TISimpleLink 设备中具有相同硬件驱动程序设备的接口。虽然在不同芯片上 UART 的硬件实现有所不同，但 TIDriverAPI 使用同样的方法访问它们的公共功能。
- **Middleware（中间件）：** 在驱动程序的基础上添加功能，例如通信栈和图形库。
- **Examples（示例）：** SDK 提供了广泛的例子。它们的目的是让开发者更容易开始编写应用程序。每个示例都有自己的文档和项目文件。示例提供了使用 RTOS 的内核的方法，也提供了不使用 RTOS 的方法。

目录结构

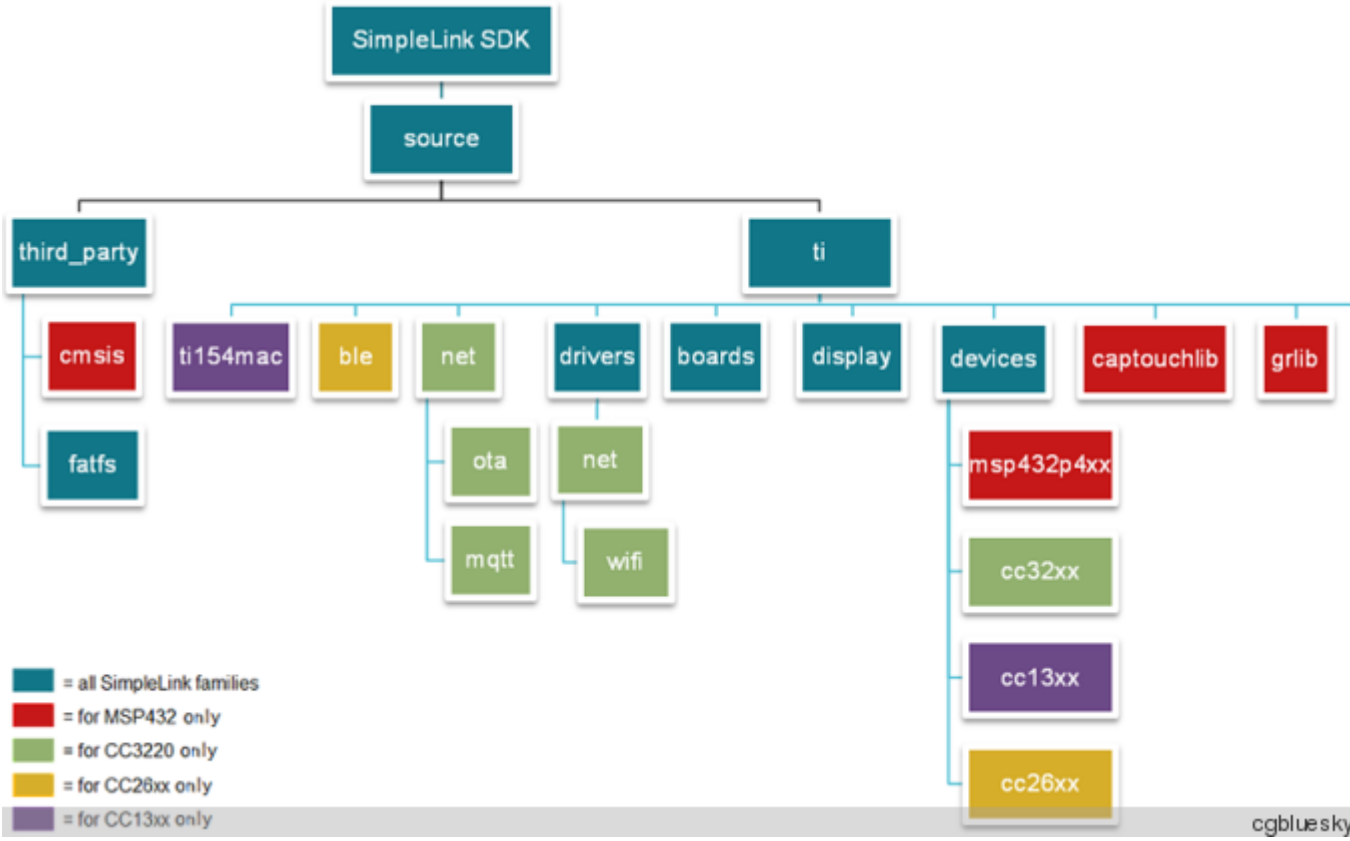
SDK 安装在/ti 或默认的 c:\ti。这也是所有与 SDK 一起安装的插件的安装位置。这是 SDK 安装的顶层目录结构：



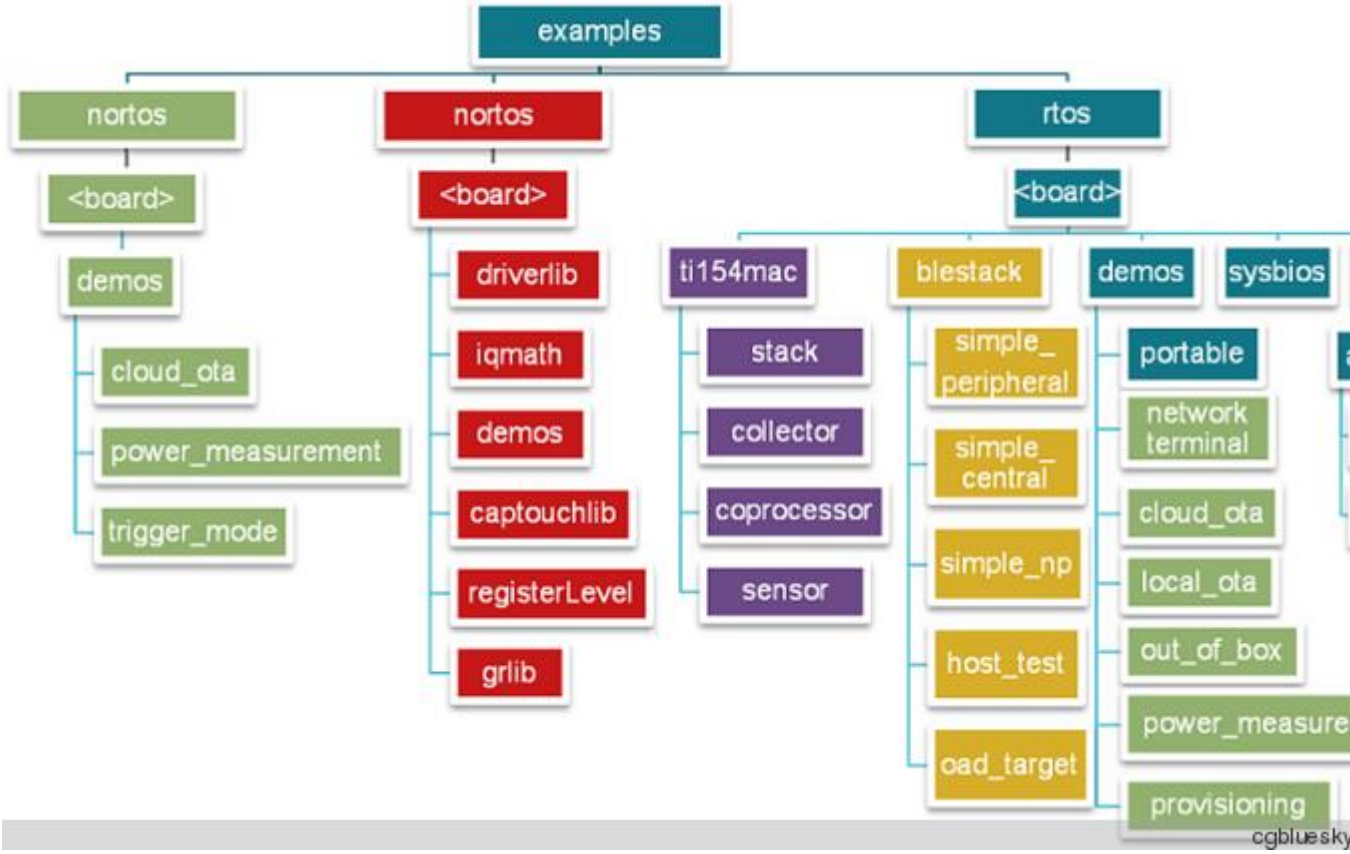
- “doc”目录包含了所有各种 SDK 组件的文档文件。【file:///C:/ti/simplelink_cc32xx_sdk_1_30_01_03/docs/Documentation_Overview.html】里有各文档的链接。
- “examples”目录包含了针对每个所支持开发板的示例应用程序。
- “kernel”目录包含了使用 TI-RTOS 内核以及 FreeRTOS 内核所需的文件。内核实现了 DPL 及 POSIX 层，使得 SDK 应用程序可以在不同内核间切换。
- “source”目录包含源码、库及编译应用程序时所使用的链接文件。这些包括驱动、主板、设备、中间件以及其它文件。例如，所有 TI 设备源码都在此提供。参考下图获取更多此目录树内容。

由于 TISimpleLink 系列的不同 SDK 所使用的目录结构相同，有可能在不同设备间移动应用程序代码无需明显改动文件结构或包含路径。

下图显示了“source”目录的内容。注意左下角方块，显示了 SDK 中的特定目标所代表的颜色。



下图详细显示了“examples”目录所包含的内容，并使用了同样的颜色。



未包含的工具和实用程序

SDK 并不包含 IDE 或代码生成器，如编译器及连接器。你应该有一个可以支持的代码生成工包。合适的工具如 TI 代码生成工具 CodeComposerStudio，IARWorkbeacn 及它的编译器和 GNU 编译器集合（gcc）。

SDK 未安装 FreeRTOS。你右通过 FreeRTOS 网站下载 FreeRTOS 源文件。SDK 已经安装了 DriverPortingLayer(DPL)以及 POSIX 抽象使得 FreeRTOS 可与其它 SDK 组件一起使用。

SDK 已经安装 XDCtools，它包含了 TI-RTOS 内核的配置工具。XDCtools 组件下面的工具可用于 TI-RTOS 及基内核的工具和模块。当你安装了 SDK，XDCtools 已经被装在了与 SDK 安装目录同级的平等目录中。

一些版本的 SDK 还提供了特定功能的工具。如用于蓝牙低功耗栈 PC 端接口的 BTool。

理解一个 SimpleLinkMUCSDK 应用程序

SimpleLinkMUCSDK 提供了一个叫 demos\portable 的例子，作为使用 SDK 创建你自己的应用程序的介绍。此示例使用了 UART、I2C 驱动及低功耗状态。它展示了多任务 I/O 驱动。此代码使用了 POSIX，所以它很容易使用 TI-RTOS 内核及 FreeRTOS 内核运行。

应用程序有如下功能：

- 在阻塞的读写模式中初始化 UART 驱动。
- consoleThread 线程提供了一个简单的控制台。当 UART 关闭，驱动可进入低功耗状态。
- temperatureThread 线程通过 I2C 驱动从外设读取温度。

portable 示例位于

【<SDK_INSTALL_DIR>\examples\rtos\<board>\demos\portable】目录。它可用于所有支持的开发板、RTOS 内核、工具链及 IDE。

导入、编译并运行示例

如 QuickStartGuide（位于 file:///C:/ti/simplelink_cc32xx_sdk_1_30_01_03/docs/simplelink_mcu_sdk/Quick_Start_Guide.html）所描述的那样导入示例。分别介绍了在 CodeComposerStudio(CCS)IDE、IAREmbeddedWorkbench、GCC 编译器的 makefiles 以及其它支持的开始环境中的使用情况。按照指南中的“ExecuteyourFirstApplication”这一节进行操作。

在 QuickStartGuide 中的相同章节所述编译“portable”。

在你导入的示例文件中有一个 README.html 文件。阅读此文件，按照介绍中的“ExampleUsage”节运行示例。

对于非 CC3220 开发板，还需要 SensorsBoosterPack，它包含了一个 TMP007 温度传感器。

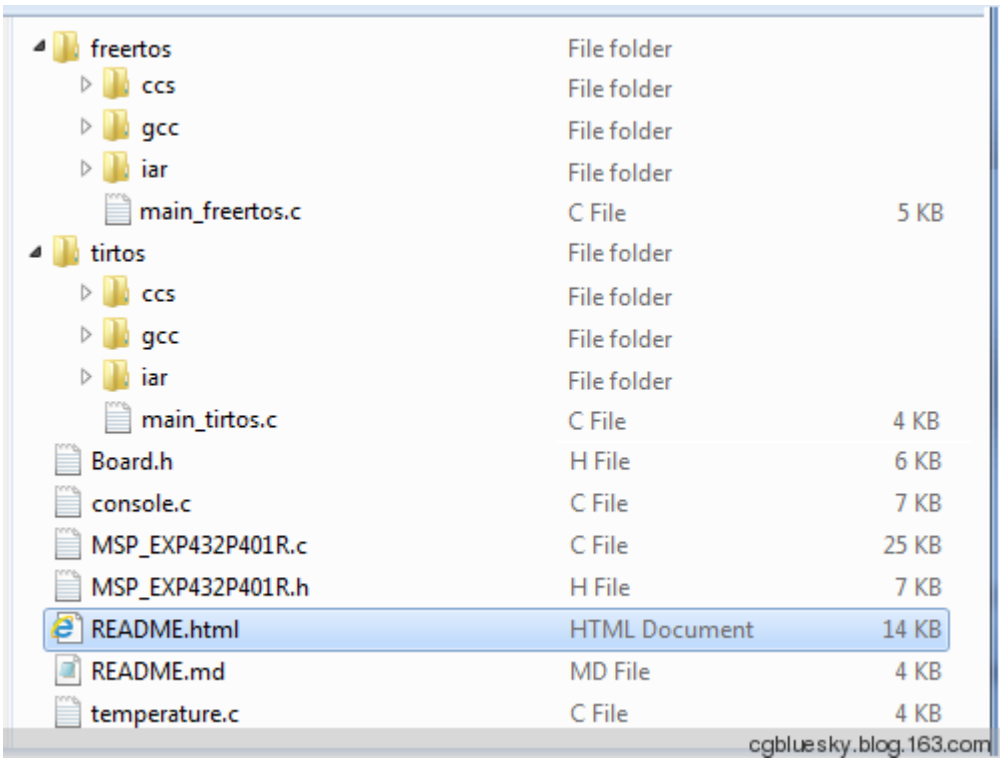
对于使用 CCS 的 CC3220 开发板，让设备“FreeRun”而不是“Run”。

理解示例

此节审视“protable”示例代码，以理解它是如何创建的。

要理解示例是如何支持多个 IDE 及 RTOS 内核的，最好的方法是读 SDK 安装的示例文件。导入一个示例及删除文件的过程用于支持其它 IDE 及 RTOS 内核。

进入到你的【<SDK_INSTALL_DIR>\examples\rtos\<board>\demos\portable】目录。



此示例包含以下源文件：

- main_tirtos.c 和 main_freertos.c：这些文件包含了 main()函数。它们位于子目录，并使用 POSIX（pthread）进行驱动程序的初始化和创建运行 consoleThread 及 temperatureThread 的线程。
- console.c：此文件包含由 consoleThread 线程所运行的函数，以及当主板 BUTTON1 按下时所产生的 GPIO 中断回调函数。

- temperature.c：此文件包含由 temperatureThread 线程所运行的函数。
- <board>.c 和<board>.h（其中<board>是主板的名称）：这些文件设置特定主板项，如各种 TI 驱动的属性结构。例如，对于 UART 驱动，时钟源和引脚的使用进行配置。
- Board.h：将特定主板常量映射至可被应用程序使用的主板无关常量，从而使得应用程序代码具有可移植性。

以下章节更为详尽地描述了应用程序的这些部分。在你阅读这些章节时最好打开代码文件，以便更好理解应用程序是如何使用 SDK 的。

main_tirtos.c 和 main_freertos.c

两个版本的 main()函数功能大多相同。因为 POSIX 的 Pthreads 用于代替内核 API 的直接调用，运行 TI-RTOS 内核及 FreeRTOS 内核只有轻微不同。

Headerfiles

两个版本的文件都包含以下头文件：

- stdint.h
- pthread.h
- ti/drivers/GPIO.h（这是一个 TI-RTOS 包路径，位于<SDK_INSTALL_DIR/source）
- Board.h

FreeRTOS 版本文件还包含 FreeRTOS.h 及 task.h。

TI-RTOS 版本文件还包含 ti/sysbios/BIOS.h，此例中，位于 <SDK_INSTALL_DIR>/kernel/tirtos/packages/ti/sysbios。

main()函数

1.main()函数开始于 Board_initGeneral()的调用，此函数定义于 Board.h 文件，指向 <board>.c 文件中的一个特定板函数。

2.接下来在 pAttrs 结构体设置各种线程的属性。

- 使用 PTHREAD_CREATE_DETACHED 状态调用 pthread_attr_setdetachstate()，使得创建的线程处于分离状态，因为此应用程序所使用的线程永远无需连接其它线程。
- 调用 pthread_attr_setschedparam()将线程调度优先级设置为 1（最小优先）。
- 调用 pthread_attr_setstacksize()将线程所使用的栈尺寸设置为 THREADSTACKSIZE，它在相同文件前面定义。

3.接下来 main()函数两次调用 pthread_create()。

- 第一次创建了一个将要运行 consoleThread()函数的线程。此线程运行于默认优先级（1，最小值）。（参考 console.c 文件获知此线程所执行的动作）
- 第二次创建了一个将要运行 temperatureThread()函数的线程。此线程的优先级高于 consoleThread（优先级 2）。这样，如果两个线程都准备运行，temperatureThread 则优先运行。当 temperature 线程阻塞或从 I2C 等待数据时，console 线程得以运行。（参考 temperature.c 文件获知此线程所执行的动作）

例如：

destyle="margin:0px;padding:0px;border:none;font-size:12px;font-family:Consolas,'LiberationMono',Courier,monospace;border-radius:3px;background:transparent;">priParam.sched_priority=2;pthread_attr_setschedparam(&pAttrs,&priParam);retc=pthread_create(&thread,&pAttrs,temperatureThread,NULL);if(retc!=0){/*pthread_create()failed*/while(1);}de>

参考 wiki 主题 SYS/BIOSPOSIXThread(pthread)Support 获取有关 SDK 支持哪些 POSIXAPI 的更详细信息。

如果你的应用程序直接选择 TI-RTOS 内核而不是 POSIX，则直接调用 Task_create()API 而不是 pthread_create()API：

destyle="margin:0px;padding:0px;border:none;font-size:12px;font-family:Consolas,'LiberationMono',Courier,monospace;border-radius:3px;background:transparent;">Task_Params_init(&taskParams);taskParams.priority=2;taskParams.stackSize=768;temperatureTaskHandle=Task_create(temperatureThread,&taskParams,&eb);de>

参考 TI-RTOSKernelUser’ sGuide 获取更多有关 TI-RTOS 内核 API 的信息。

1.main()函数接下来创建一个互斥体，将用于 consoleThread()及 temperatureThread()更新温度时护它的安全。

2.接下来调用 GPIO_init()来初始化 GPIO 驱动。此项工作在 main()中完成是因为线程将使用 GPIO 驱动。

3.此时，两个版本的 main()函数分开。

- TI-RTOS 版本内核简单调用 BIOS_start()以开始调度。BIOS_start()不会返回。
- FreeRTOS 版本调用 vTaskStartScheduler()以开始 FreeRTOS 调度。

  vTaskStartScheduler()不会返回。

console.c

此文件对于所有开发板、RTOS 内核及开发环境是相同的。记得查看 README.html 文件获取有关如何打开一个串口会话来运行示例控制台的内容。

consoleThread()函数在 RTOS 开始调度时运行。首先它会执行一些配置任务，接下来进入 while 循环，直到应用程序停止。

在函数的配置部分，consoleThread()函数执行以下动作：

1. 使能 CC3220 开发板的电源管理，为了方便调试，CC3220 默认情况下是关掉电源管理的。
2. 配置 Board_GPIO_BUTTON1 在回应按键时运行 gpioButtonFxn()以唤醒主板。
3. 调用 POSIX 函数 sem_init()以初始化一个匿名信号量。按钮按下将提交信号量，之后的 while 循环则等待信号量。
4. 初始化 UART 通信参数。

   在 while 循环中，consoleThread()函数执行以下动作：

1. 如果 uartEnabled 为 false（当控制台关闭时发生），等待信号量，直到按钮按下。

2. 通过配置参数调用 UART_open()在阻塞读写模式下为控制台打开一个 UART。
3. 运行 simpleConsole()函数，它使用 UART_write()和 UART_read()来管理控制台的用户接口，此用户接口接收以下命令：

   destyle="margin:0px;padding:0px;border:none;font-size:12px;font-family:Consolas,'LiberationMono',Courier,monospace;border-radius:3px;background:transparent;">ValidCommands----------h:helpq:quitandshutdownUARTc:clearthescreent:displaycurrenttemperaturede>

1. 如果使用“t”命令，由 temperatureThread 所提供的外部变量 temperatureC 和 temperatureF 的值都存储于本地。一个互斥量用于保护这些操作。没有互斥量，拥有更高优先级的 temperature 线程可能会中断 console 线程对这些变量的读取。这会导致 console 线程所打印的摄氏温度值与华氏温度值不匹配。
2. 静态 itoa()函数用于将整数温度值转换为字符串，UART_write()用于将值输出到控制台。
3. 如果使用“q”命令，simpleConsole()函数返回并调用 UART_close()。UART 关闭后，电源管理自动进入低功耗状态。参考 TI-RTOSPowerManagementUser’ sGuide 获取更多有关电源状态的信息。

temperature.c

此文件对于所有开发板、RTOS 内核及开发环境是相同的。

temperatureThread()函数在 RTOS 开始调度时运行。首先它会执行一些配置任务，然后进入 while 循环直到应用程序关闭。线程在 while 循环结尾处阻塞等待 semTimer 的信号

量。定时器每隔 1 秒提交一次 semTimer 信号量以让任务解除阻塞。sleep()函数可以使用，但会发生一个很小的偏移量。

在函数的配置部分，temperatureThread()函数执行以下动作：

1. 初始化 I2C 驱动并打开驱动使用。
2. 初始化 I2C_transaction 结构体，它将用于 I2C_transfer()。
3. 初始化定时器和 semTimer 信号量。

在 while 循环中，temperatureThread()函数执行这些动作：

1. 如果 I2C 传输成功，则互斥量锁止以让随后的线程操作安全。
2. 从接收到的数据提取温度。（对于 C3200 开发板，默认读取板载 TMP006 传感器的 TMP_DIE_TEMP 寄存器。对于所有其它开发板，示例读取 SensorsBoosterPack 上的 TMP007 传感器的 TMMP_OBJ_TEMP 寄存器。）
3. 温度在摄氏温度下被精确化，并转化为华氏温度。两个值都被存储并输出。
4. 释放互斥锁。
5. 如果温度超过 30 度，将使用 GPIO_write()写一个警告信息，否则，所有之前的警告信息被清空。
6. while 循环阻塞于 semTimer 信号量，此信号量由线程创建的定时器每秒提交一次，这为设备提供了足够的时间在电源管理的控制下转换为低功耗状态。

   注意：MSP432 使用看门狗以实现更好地能量节省，详情请参考 wiki 主题 TI-RTOSMSP432Timer。

<board>.c 和<board>.h

（<board>是你的开发板的名称）这些文件负责建立开发板的主板特定项。例如，GPIO 部分配置 GPIO 输入、输出引脚和 LED。它为输入引脚创建了一组回调函数并设置为 NULL。对于每个驱动，它都声明了一个配置结构体并将结构体内属性字段设置为默认值。

<board>.c 文件还配置 TI 驱动使用的结构体。这些结构体对于大多数 TI 设备相同。例如，<board>.c 中定义的一个叫<Driver>_Config 的配置数据结构体。每个驱动的配置包含一个指向函数表的指针，一个指向对象的指针，和一个指向一组硬件属性设置的指针。有关驱动配置结构体的更详细信息，参考 detailedreferenceinformation，位于【file:///C:/ti/simplelink_cc32xx_sdk_1_30_01_03/docs/tidrivers/doxygen/html/index.html】。

Board.h

此文件将主板特定常量和函数映射至主板无关常及函数。主板无关名称将被应用程序使用，从而应用程序代码变得可移植。

CCS、GCC 和 IA 文件

对于每个支持的 RTOS，当你在开发环境导入示例时，它包含了为此示例建立项目的文件。其中包含 ComposerStudio(CCS)、IAREmbeddedWorkbench 和 theGNUCompilerCollection(GCC)的代码文件。

更多相关资料访问博客:
http://cgbluesky.lofter.com/