UM1722
用户手册

在具有 RTOS 的 STM32Cube 上开发应用

前言

STMCube™ 计划源自意法半导体，旨在通过减少开发的工作量、时间与成本，使开发者受益。 STM32Cube 涵盖 STM32 产品系列。

STM32Cube 1.x 版包括：

- 图形软件配置工具 STM32CubeMX，可通过图形化的生成初始化 C 代码。
- 针对每个系列提供综合的嵌入式软件平台 （即 STM32CubeF4 用于 STM32F4 系列）
  - STM32 抽象层嵌入式软件 STM32Cube HAL，确保在 STM32 各个产品之间实现最大限度的可移植性
  - 一套一致的中间件，比如 RTOS、 USB、 TCP/IP、图形
  - 所有嵌入式软件实用工具均配备一套完整的示例。

实时操作系统是为在嵌入式 / 实时应用中使用而优化的操作系统。它们的主要目标是确保及时、确定性地响应事件。使用实时操作系统，应用可写为一组独立的线程，线程间使用消息队列和信号量通信。

本用户手册的目标读者为在 STM32 微控制器上使用 STM32Cube 固件的开发者。它完整描述了如何使用具有实时操作系统 （RTOS）的 STM32Cube 固件组件；本用户手册还提供了一组示例说明，它们基于 FreeRTOS，使用 CMSIS-OS 封装层提供的通用 API。

在 STM32Cube 固件中，通过 ARM 提供的通用 CMSIS-OS 封装层，将 FreeRTOS 用作实时操作系统。使用 FreeRTOS 的样例和应用可直接移植到其它任何 RTOS 而不需要修改高层 API，在此情况下仅需更改 CMSIS-OS 封装。

请参考软件包的发布说明，以了解与 STM32Cube™ 共同使用的 FreeRTOS 和 CMSIS-RTOS 固件组件版本。

本文档适用于所有 STM32 器件；然而为了简洁起见，以 STM32F4xx 器件和 STM32CubeF4 作为参考平台。若需了解更多在 STM32 设备上样例实现的信息，请参考相关 STM32Cube 固件包中提供的自述文件。

目录

表格索引

图片索引

1 Free RTOS

1.1 概述

FreeRTOS 是 RTOS 的一种，尺寸非常小，可运行于微控制器上，但其使用并不限于微控制器应用。

微控制器是尺寸小、资源受限的处理器，它在单个芯片上包含了处理器本身、用于保存要执行的程序的只读存储器 （ROM 或 Flash）、所执行程序需要的随机存取存储器 （RAM）。一般情况下，程序直接从只读存储器执行。

微控制器用于深度嵌入式应用（对于那些应用，您永远不会看到处理器本身或运行的软件），它们一般有非常明确、专门的工作。尺寸的限制以及专用的终端应用等性质，令其很少能使用完整的 RTOS 实现 - 或者说不可能使用完整的 RTOS 实现。因此，FreeRTOS 仅为内核提供了实时调度功能、任务间通信、时序和同步原语。这意味着更准确地说，它是一个实时内核，或实时执行器。命令控制台界面、网络栈等额外的功能可作为附加组件。

FreeRTOS 为可调整的实时示例生成器内核，专为小型嵌入式系统设计。其特点包括

- Free RTOS 示例生成器内核 —— 优先式、合作式及混合式配置选项。
- 官方支持 27 种架构 （ARM7 和 ARM Cortex M3 每个算一种架构）。
- FreeRTOS-MPU 支持 Cortex M3 存储器保护单元 （MPU）。
- 设计目标为小尺寸、简单和易用。一般来说，示例生成器内核二进制映像大约为 4K 到 9K 字节。
- 代码结构极易移植，主要用 C 编写。
- 支持任务和协同例程。
- 可通过队列、二进制信号量、计数信号量、递归信号量、互斥量在任务间、任务与中断间通信和同步。
- 互斥量有优先级继承。
- 支持高效的软件定时器。
- 强大的执行跟踪功能。
- 栈溢出检测选项。
- 预配置的示例应用，用于选定的单板电脑，可直接使用，加快学习曲线。
- 免费论坛支持，或可选择商业支持和授权。
- 可创建的任务数无软件限制。
- 可使用的优先级数无软件限制。
- 优先级指定无限制 - 可为多个任务指定同一优先级。
- 免费的开发工具可用于很多支持的架构。
- 免费的嵌入式软件源代码。
- 免版税。
- 可从标准的 Windows 主机交叉开发。

FreeRTOS 的 heap2 方案用于内存分配管理，此方案使用最佳适用算法释放之前分配的块。然而，它不会将相邻的自由块合并为一个大块。可用的 RAM 总量通过定义 *configTOTAL_HEAP_SIZE* 设置 - 定义于 *FreeRTOSConfig.h* 中。

1.2 授权

FreeRTOS 源代码使用修正的 GNU 通用公开许可来授权。该修正使用了除外形式。 GNU 通用公开许可全文如下：

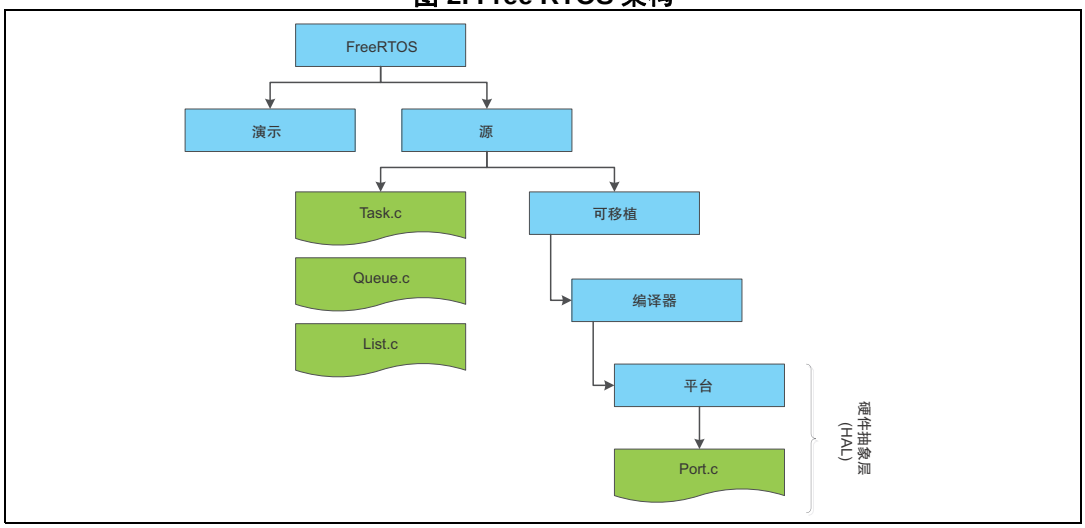
图 1. FreeRTOS 许可

```
The FreeRTOS.org source code is licensed by the modified GNU General Public
License (GPL) text provided below.  The FreeRTOS download also includes
demo application source code, some of which is provided by third parties
AND IS LICENSED SEPARATELY FROM FREERTOS.ORG.

For the avoidance of any doubt refer to the comment included at the top
of each source and header file for license and copyright information.

This is a list of files for which Real Time Engineers Ltd are not the
copyright owner and are NOT COVERED BY THE GPL.


1) Various header files provided by silicon manufacturers and tool vendors
   that define processor specific memory addresses and utility macros.
   Permission has been granted by the various copyright holders for these
   files to be included in the FreeRTOS download.  Users must ensure license
   conditions are adhered to for any use other than compilation of the
   FreeRTOS demo applications.

2) The uIP TCP/IP stack the copyright of which is held by Adam Dunkels.
   Users must ensure the open source license conditions stated at the top
   of each uIP source file is understood and adhered to.

3) The lwIP TCP/IP stack the copyright of which is held by the Swedish
   Institute of Computer Science.  Users must ensure the open source license
   conditions stated at the top  of each lwIP source file is understood and
   adhered to.

4) Various peripheral driver source files and binaries provided by silicon
   manufacturers and tool vendors.  Permission has been granted by the
   various copyright holders for these files to be included in the FreeRTOS
   download.  Users must ensure license conditions are adhered to for any
   use other than compilation of the FreeRTOS demo applications.

5) The files contained within FreeRTOS\Demo\WizNET_DEMO_TERN_186\tern_code,
   which are slightly modified versions of code provided by and copyright to
   Tern Inc.

Errors and omissions should be reported to Richard Barry, contact details for
whom can be obtained from http://www.FreeRTOS.org.


The GPL license text follows.

A special exception to the GPL is included to allow you to distribute a
combined work that includes FreeRTOS without being obliged to provide
the source code for any proprietary components.  See the licensing section
of http://www.FreeRTOS.org for full details.  The exception text is also
included at the bottom of this file.
```

1.3 Free RTOS 源代码组织

下载的 FreeRTOS 包括每个处理器移植及每个示例应用的源代码。将所有移植放置在一处下载会极大简化发布，但文件数太多。然而，目录结构非常简单，而且 FreeRTOS 实时内核仅包含在 4 个文件中 （若需软件定时器或协同例程功能，则需更多文件）。

图 2. Free RTOS 架构



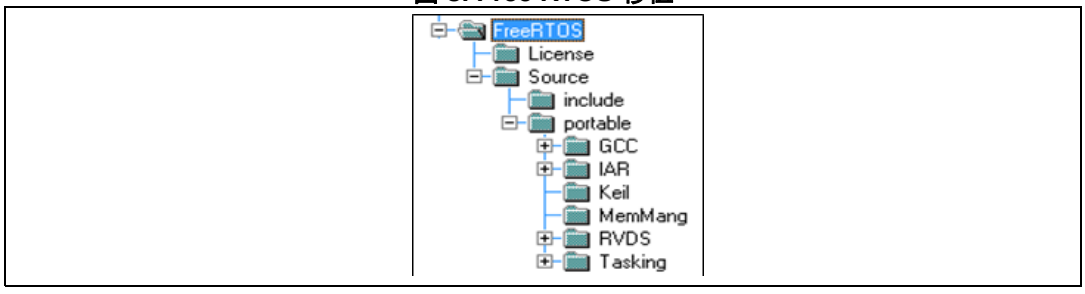
RTOS内核代码包含在三个文件中，名为tasks.c、queue.c和list.c，它们位于FreeRTOS/Source目录中。该目录还包含两个可选文件，名为 timers.c 和 croutine.c，它们实现了软件定时器和协同例程功能。每个所支持的处理器架构都需要一小部分专门针对该架构的 RTOS 代码。这就是 RTOS 移植层，它位于 FreeRTOS/Source/Portable/[compiler]/[architecture] 子目录中，其中 [compiler] 和 [architecture] 分别是创建移植所使用的编译器，以及移植所运行的架构。

样例堆分配方案也位于移植层中。不同的样例 heap_x.c 文件位于 FreeRTOS/Source/portable/MemMang 目录中。

1.4 将 FreeRTOS 移植到 STM32

FreeRTOS 支持下列 ST 处理器系列：STM32 （Cortex-M0、 Cortex-M3 和 Cortex-M4F）、STR7 （ARM7）和 STR9 （ARM9） ，可与下列工具共同使用：IAR、 Atollic TrueStudio、GCC、 Keil、 Rowley CrossWorks。

图 3. Free RTOS 移植



1.5 FreeRTOS API

表 1. Free RTOS API

API 类别	API
任务创建	– xTaskCreate – vTaskDelete
任务控制	– vTaskDelay – vTaskDelayUntil – uxTaskPriorityGet – vTaskPrioritySet – vTaskSuspend – vTaskResume – xTaskResumeFromISR – vTaskSetApplicationTag – xTaskCallApplicationTaskHook
任务工具	– xTaskGetCurrentTaskHandle – xTaskGetSchedulerState – uxTaskGetNumberOfTasks – vTaskList – vTaskStartTrace – ulTaskEndTrace – vTaskGetRunTimeStats
内核控制	– vTaskStartScheduler – vTaskEndScheduler – vTaskSuspendAll – xTaskResumeAll
队列管理	– xQueueCreate – xQueueSend – xQueueReceive – xQueuePeek – xQueueSendFromISR – xQueueSendToBackFromISR – xQueueSendToFrontFromISR – xQueueReceiveFromISR – vQueueAddToRegistry – vQueueUnregisterQueue
信号量	– vSemaphoreCreateBinary – vSemaphoreCreateCounting – xSemaphoreCreateMutex – xSemaphoreTake – xSemaphoreGive – xSemaphoreGiveFromISR

1.6 FreeRTOS 存储器管理

FreeRTOS 源代码下载 （V2.5.0 及更高）包含了四个样例 RAM 分配方案。各种示例应用可视情况使用。下面的小节说明了可用的方案及使用条件，并着重说明了可演示其用法的示例程序。

每个方案都包含在一个单独的源文件中（分别是heap_1.c、heap_2.c、heap_3.c和heap_4.c），它们可位于 Source/Portable/MemMang 目录中。若需要可添加其他方案。

方案 1 - heap_1.c

这是所有方案里最简单的。当内存分配后，它不允许释放内存，但除了这点，它适合于大量的应用。

该算法仅在请求 RAM 时，将一个数组分为更小的块。数组总大小通过定义 configTOTAL_HEAP_SIZE 设置 - 定义于 FreeRTOSConfig.h 中。此方案特点为：

若您的应用永远不会删除任务或队列 （永远不会调用 vTaskDelete () 或 vQueueDelete ()），则可使用。

- 它是确定性的 （总是用相同时间返回一个块）。
- 它被 PIC、AVR 及 8051 演示应用使用 - 因为它们在调用 vTaskStartScheduler() 后，不会动态创建或删除任务。

heap_1.c 适用于很多在内核启动前即创建了所有任务和队列的小实时系统。

方案 2 - heap_2.c

此方案使用了最佳适用算法，与方案 1 不同，它允许释放之前分配的块。然而，它不会将相邻的自由块合并为一个大块。

同样，可用的RAM总量通过定义configTOTAL_HEAP_SIZE设置 - 定义于FreeRTOSConfig.h中。

此方案特点为：

- 即使应用反复调用 vTaskCreate ()/vTaskDelete () 或 vQueueCreate ()/vQueueDelete ()（导致多次调用 pvPortMalloc() 和 vPortFree()），仍可使用此方案。
- 若分配和释放的内存为随机大小 —— 仅在每个被删除的任务都有不同的栈深度，或被删除的队列有不同的长度时才会如此 —— 则不应使用此方案。
- 若您的应用创建队列或任务块的顺序不可预测，则可能导致内存碎片问题。可能所有应用都不会这样，但是应对此了解。
- 它不是确定性的 - 但它也不是效率特别低的。

heap_2.c 适合于很多必须动态创建任务的小实时系统。

方案 3 - heap_3.c

它仅是标准 malloc() 和 free() 函数的封装。它可确保线程安全。此方案特点为：

- 需要链接器建立堆，编译器库提供 malloc() 和 free() 的实现。
- 它不是确定性的。
- 可能会大幅增加内核代码量。
- 它被 PC （x86 单板电脑）演示应用所使用。

方案 4 - heap_4.c

此方案使用了首先适用算法，与方案 2 不同，它不会将相邻的自由内存块合并为一个大块（它不包含合并算法）。

可用的堆空间总量通过 configTOTAL_HEAP_SIZE 设置 - 定义于 FreeRTOSConfig.h 中。

xPortGetFreeHeapSize() API 函数返回还未分配的堆空间总量（令 configTOTAL_HEAP_SIZE 设置优化），但它不会提供未分配内存如何分片为更小块的信息。

此实现的特点为：

- 即使应用反复删除任务、队列、信号量、互斥量等等，仍可使用此实现。
- 相比于 heap_2 实现，它不容易产生分片为多个小块的堆空间 - 即使在分配和释放的内存大小随机时也是如此。
- 它不是确定性的 - 但它比多数标准 C 库的 malloc 实现的效率高得多。

对于需要在应用代码中直接使用移植层内存分配方案 （而不是通过调用 API 函数，间接调用 pvPortMalloc() and vPortFree()）的应用而言， heap_4.c 尤其有用。

1.7 FreeRTOS 低功耗

通常，降低 FreeRTOS 所在微控制器功耗的方法是使用空闲任务钩子函数，将微控制器置为低功耗状态。此简单方法可实现的功率节省有限，因为需要周期性离开再进入低功耗状态以处理时间片中断。此外，若时间片中断的频率太高，则在每个时间片进入再离开低功耗状态所消耗的能量和时间会太多，超过最轻功耗节省模式可能带来的功率节省增益。

FreeRTOS 的无时间片空闲模式停止了空闲期间 （无应用线程可执行期间）的周期性时间片中断，当时间片中断重新开始时，再对 RTOS 时间片的数值做纠正调整。

停止时间片中断使微控制器能一直保持在功率节省状态，直到中断发生，或 RTOS 内核需要将线程转为就绪状态时。

1.8 FreeRTOS 配置

可配置的参数数量很多，可令 FreeRTOS 内核完全贴合您的特定应用要求。这些项都位于一个名为 FreeRTOSConfig.h 的文件中。 FreeRTOS 源代码下载中包含的每个演示应用都有它自己的 FreeRTOSConfig.h 文件。下面是一个典型的例子

图 4. FreeRTOS 配置

```
/* Ensure stdint is only used by the compiler, and not the assembler. */
#if defined(__ICCARM__) || defined(__CC_ARM) || defined(__GNUC__)
 #include <stdint.h>
 extern uint32_t SystemCoreClock;
#endif

#define configUSE_PREEMPTION                    1
#define configUSE_IDLE_HOOK                     0
#define configUSE_TICK_HOOK                     0
#define configCPU_CLOCK_HZ                      ( SystemCoreClock )
#define configTICK_RATE_HZ                      ( ( portTickType ) 1000 )
#define configMAX_PRIORITIES                    ( ( unsigned portBASE_TYPE ) 7 )
#define configMINIMAL_STACK_SIZE                ( ( unsigned short ) 128 )
#define configTOTAL_HEAP_SIZE                   ( ( size_t ) ( 15 * 1024 ) )
#define configMAX_TASK_NAME_LEN                 ( 16 )
#define configUSE_TRACE_FACILITY                1
#define configUSE_16_BIT_TICKS                  0
#define configIDLE_SHOULD_YIELD                 1
#define configUSE_MUTEXES                       1
#define configQUEUE_REGISTRY_SIZE               8
#define configCHECK_FOR_STACK_OVERFLOW          0
#define configUSE_RECURSIVE_MUTEXES             1
#define configUSE_MALLOC_FAILED_HOOK            0
#define configUSE_APPLICATION_TASK_TAG          0
#define configUSE_COUNTING_SEMAPHORES           1
/* Cortex-M specific definitions. */
#ifdef __NVIC_PRIO_BITS
       /* __BVIC_PRIO_BITS will be specified when CMSIS is being used. */
       #define configPRIO_BITS                __NVIC_PRIO_BITS
#else
       #define configPRIO_BITS                4        /* 15 priority levels */
#endif

/* The lowest interrupt priority that can be used in a call to a "set priority"
function. */
#define configLIBRARY_LOWEST_INTERRUPT_PRIORITY     0xf

/* The highest interrupt priority that can be used by any interrupt service
routine that makes calls to interrupt safe FreeRTOS API functions */
#define configLIBRARY_MAX_SYSCALL_INTERRUPT_PRIORITY     5

/* Interrupt priorities used by the kernel port layer itself.  These are generic
to all Cortex-M ports, and do not rely on any particular library functions. */
#define configKERNEL_INTERRUPT_PRIORITY             (
configLIBRARY_LOWEST_INTERRUPT_PRIORITY << (8 - configPRIO_BITS) )
#define configMAX_SYSCALL_INTERRUPT_PRIORITY  (
configLIBRARY_MAX_SYSCALL_INTERRUPT_PRIORITY << (8 - configPRIO_BITS) )

/* Definitions that map the FreeRTOS port interrupt handlers to their CMSIS
standard names. */
#define vPortSVCHandler SVC_Handler
#define xPortPendSVHandler PendSV_Handler

/* IMPORTANT: This define MUST be commented when used with STM32Cube firmware,
              to prevent overwriting SysTick_Handler defined within STM32Cube HAL
*/
/* #define xPortSysTickHandler SysTick_Handler */
```

注： *当使用 FreeRTOS 时，为防止重复定义，必须从 stm32f4xx_it.c/.h 中删除 SVC_Handler 和 PendSV_Handler*

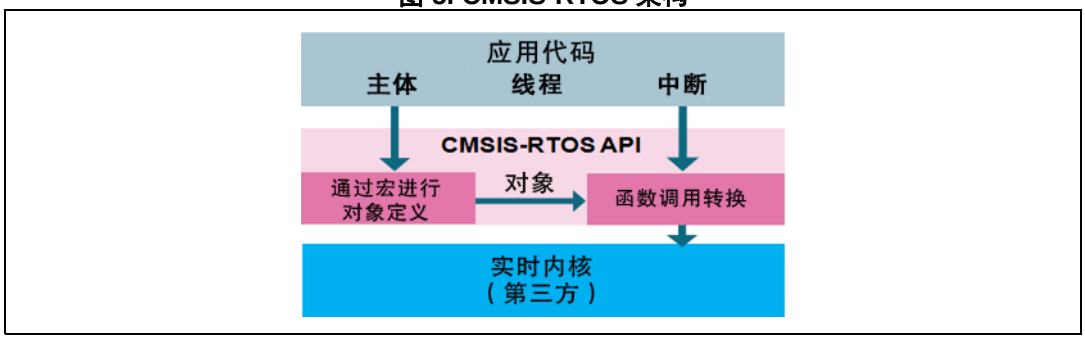
2 CMSIS-RTOS 模块

2.1 概述

CMSIS-RTOS 是实时操作系统的通用 API。它提供了标准化的编程接口，可移植到很多 RTOS，使软件模板、中间件、库及其它组件能工作于支持的 RTOS 系统。

此模块由 cmsis_os.c/h 文件表示，位于“Middlewares\Third_Party\FreeRTOS\CMSIS_RTOS”下。

图 5. CMSIS-RTOS 架构



典型的 CMSIS-RTOS API 实现与已有实时内核连接。 CMSIS-RTOS API 提供了下列属性和功能：

- 函数名、标识符、参数都容易理解。函数功能强大灵活，减少了暴露给用户的函数数目。
- 线程管理可定义、创建、控制线程。
- 中断服务程序（ISR）可调用很多 CMSIS-RTOS 函数。当无法从 ISR 调用 CMSIS-RTOS 函数时，函数会拒绝该调用。
- 有三种不同的线程事件类型，可支持多线程和 / 或 ISR 间的通信：
  - 信号：该标志可用于向线程发出特定状态的信号。可在 ISR 中修改信号，或从其它线程设置。
  - 消息：为 32 位的值，可被发送到线程或 ISR。消息缓存于队列中。消息类型和队列大小定义于描述符中。
  - 邮件：为固定大小的内存块，可被发送到线程或 ISR。邮件缓存于队列中，提供有内存分配。邮件类型和队列大小定义于描述符中。
- 包括了互斥量管理和信号量管理。
- CPU 时间可用
- 下列功能调度：
  - 超时参数包含在很多 CMSIS-RTOS 函数中，以避免系统死锁。当指定了超时参数时，系统会等待，直到有资源可用或事件发生。当等待时，可调度其它线程。

- osDelay 函数会将线程置于 WAITING 状态一段特定时间。
- 通用的 osWait 函数会等待指定到线程的事件。
- osThreadYield 提供了合作式的线程切换，将执行传递给相同优先级的另一个线程。

CMSIS-RTOS API 设计为可选择地包括多处理器系统和 / 或通过 Cortex-M 存储器保护单元（MPU）进行访问保护。

在一些 RTOS 实现中，线程可能在不同的处理器上执行，因此邮件和消息队列可存在于共享的存储器资源中。

CMSIS-RTOS API 鼓励软件业界发展现有的 RTOS 实现。使用宏定义与访问内核对象。这可实现差异化。RTOS 实现可针对 Cortex-M 处理器，在各方面差异化并优化。例如，可选的特性可为：

- 通用 Wait 函数，即，支持时间周期。
- 支持 Cortex-M 存储器保护单元 （MPU）。
- 零复制邮件队列。
- 支持多处理器系统。
- 支持 DMA 控制器。
- 确定性上下文切换。
- 循环调度上下文切换。
- 死锁避免，例如反转优先级。
- 通过使用 Cortex-M3/M4 指令 LDEX 和 STEX，实现零中断延迟。

2.2 CMSIS-RTOS API

下表提供了所有 CMSIS-RTOS API 的简单概述：

表 2. CMSIS-RTOS API

模块	API	说明
内核信息和控制	osKernelInitialize	初始化 RTOS 内核
	osKernelStart	启动 RTOS 内核
	osKernelRunning	查询 RTOS 内核是否在运行
	osKernelSys Tick (*)	得到 RTOS 内核系统定时器计数器
	osKernelSys TickFrequency (*)	RTOS 内核系统定时器频率，单位 Hz
	osKernelSys TickMicroSec (*)	将微秒值转换为 RTOS 内核系统定时器值

表 2. CMSIS-RTOS API （续）

模块	API	说明
线程管理： 定义、创建和控制线程函数	osThreadCreate	开始执行线程函数。
	osThreadTerminate	停止执行线程函数。
	osThreadYield	将执行传递至下一个就绪的线程函数。
	osThreadGetId	得到线程标识符，以引用此线程。
	osThreadSetPriority	更改线程函数的执行优先级。
	osThreadGetPriority	得到线程函数当前的执行优先级。
通用 Wait 函数： 等待一段时间或未指定的事件。	osDelay	等待指定的时间。
	osWait (*)	等待信号、消息、邮件类型的任何事件。
定时器管理 (*)： 创建并控制定时器和定时器回调函数。	osTimerCreate	定义定时器回调函数的属性
	osTimerStart	以一个时间值开始或重新开始定时器。
信号管理：控制或等待信号标志。	osSignalSet	设置线程的信号标志。
	osSignalClear	复位线程的信号标志。
	osSignalClear	挂起执行，直到特定的信号标志被设置。
互斥量管理 (*)： 使用互斥量同步线程的执行。	osMutexCreate	定义并初始化互斥量
	osMutexWait	得到互斥量，或等待其变为可用。
	osMutexRelease	释放互斥量
	osMutexDelete	删除互斥量
信号量管理 (*)： 控制对共享资源的访问。	osSemaphoreCreate	定义并初始化信号量。
	osSemaphoreWait	得到信号量令牌，或等待其变为可用。
	osSemaphoreRelease	释放信号量令牌。
	osSemaphoreDelete	删除信号量
内存池管理 (*)： 定义并管理固定大小的内存池。	osPoolCreate	定义并初始化固定大小的内存池。
	osPoolAlloc	分配内存块。
	osPoolCAlloc	分配内存块并将此块置零。
	osPoolFree	将内存块返回至内存池。

表 2. CMSIS-RTOS API （续）

模块	API	说明
消息队列管理 (*)：控制、发送、接收或等待消息。	osMessageCreate	定义并初始化消息队列。
	osMessageCreate	将消息放在消息队列中。
	osMessageCreate	得到消息，或挂起线程执行直到消息到达
邮件队列管理 (*)：控制、发送、接收或等待邮件。	osMailCreate	定义并初始化具有固定大小内存块的邮件队列
	osMailAlloc	分配内存块
	osMailCAlloc	分配内存块并将此块置零
	osMailPut	将内存块放在邮件队列中
	osMailGet	获取邮件，或挂起线程直到邮件到达。
	osMailFree	将内存块返回至邮件队列。

标为 (*) 的模块或 API 是可选的。

3 FreeRTOS 应用

STM32CubeF4 FreeRTOS 软件包有若干使用栈 API 集的应用。

这些应用分为两类

表 3. Free RTOS 应用类别

类别	应用
入门级 （基础）	**线程创建示例**
	线程间信号量示例
	ISR 信号量示例
	互斥量示例
	队列示例
特性	**定时器示例**
	低功耗示例

3.1 线程创建示例

使用 RTOS 的实时应用可组织为一组独立线程。每个线程在其自身的上下文内执行，与系统内的其它线程或 RTOS 调度器自身不会发生同时依赖性。在任一时间点，应用内只会执行一个线程，由 RTOS 调度器负责决定应执行哪个线程。

本例的目的是解释怎样基于 FreeRTOS API，使用 CMSIS-RTOS 创建线程。

该例实现了两个线程，使用同一优先级运行，周期循环执行。下面详细说明了每个线程的执行。

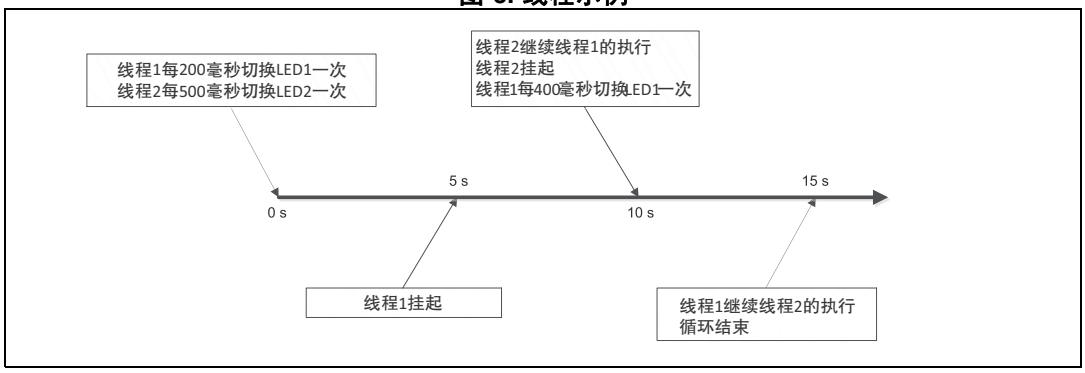
线程 1：此线程在 5 秒内，每 200 毫秒切换 LED1 一次，然后挂起，5 秒之后，线程 2 继续线程 1 的执行，在接下来的 5 秒，每 400 毫秒切换 LED1 一次。

线程创建说明：

```
/* 线程 1 定义 */
osThreadDef(LED1, LED_Thread1, osPriorityNormal, 0,
configMINIMAL_STACK_SIZE);

/* 启动线程 1 */
LEDThread1Handle = osThreadCreate (osThread(LED1), NULL);
```

线程 2：此线程在 10 秒内，每 500 毫秒切换 LED2 一次，然后挂起自己。线程 1 将在 5 秒后继续线程 2 的执行。

图 6. 线程示例



使用此例：

- 生成应用代码并编程至 STM32 闪存
- 运行样例，检查 LED 是否如*图 6* 中描述的一样切换。

3.2 信号量示例

信号量可用于互斥及同步的目的。

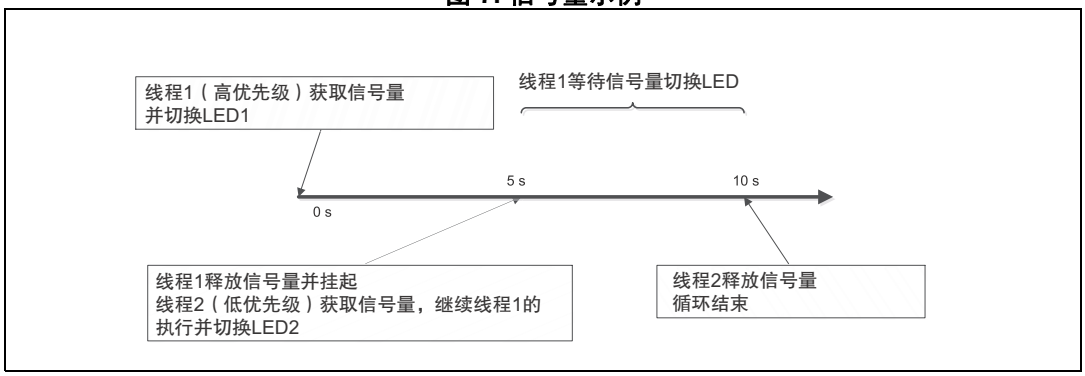
3.2.1 线程间信号量

本例的目的是解释怎样基于 FreeRTOS API，通过 CMSIS-RTOS 使用信号量。

本例实现了具有不同优先级的两个线程，它们共享一个信号量以切换 LED，下面是本例执行的更详细信息。

1. 具有更高优先级的线程 1 得到信号量，切换 LED1 5 秒
2. 线程 1 释放信号量，挂起自己。
3. 现在，低优先级的线程可以执行了，它得到信号量，继续线程 2 的执行。
4. 因为线程 1 具有更高优先级，所以它尝试得到信号量，但因为信号量已经被低优先级线程得到，因此线程 1 阻塞，
5. 线程 2 切换 LED2 5 秒，然后释放信号量，开始新循环。

图 7. 信号量示例



信号量创建说明：

```
/* 定义信号量 */
osSemaphoreDef(SEM);

/* 创建二进制信号量 */
osSemaphoreId osSemaphore = osSemaphoreCreate(osSemaphore(SEM), 1);
```

使用此例：

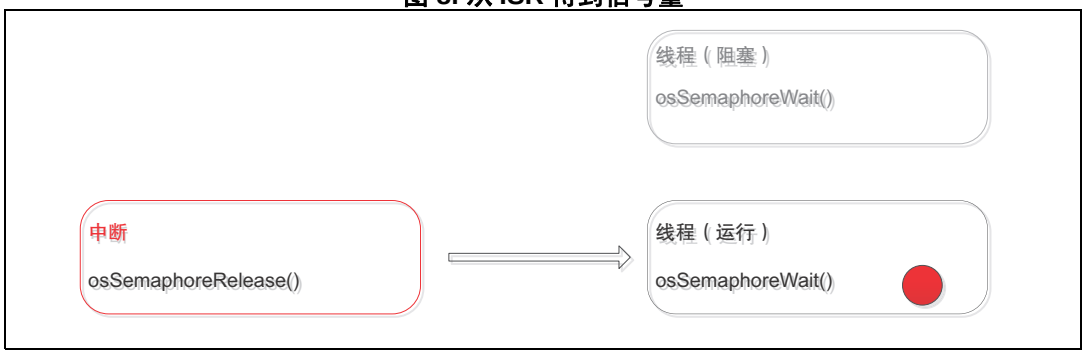
1. 生成应用代码并编程至 STM32 闪存
2. 运行样例，检查 LED 是否如*图 7* 中描述的一样切换。

3.2.2 从 ISR 得到信号量

本例演示了怎样使用来自中断的信号量。

它包含了一个基本线程，无限等待信号量切换 LED。当用户按评估板的 KEY 按钮后，STM32 会生成中断，信号量被释放。

图 8. 从 ISR 得到信号量



使用此例

1. 生成应用代码并编程至 STM32 闪存
2. 运行样例，检查当按评估板的 **KEY** 按钮时， LED1 是否会切换。

3.3 互斥量示例

互斥量是包含了优先级继承机制的二进制信号量。二进制信号量更适合于实现同步 （在任务之间，或在任务和中断之间），而互斥量更适合于实现简单的互斥。

本例创建了三个具有不同优先级的线程，它们访问同一个互斥量。

1. 高优先级线程首先执行，抢占互斥量，然后短时间睡眠，让更低的优先级线程执行。
2. 中优先级线程通过执行阻塞 “wait”，尝试访问互斥量。当互斥量已经被高优先级线程得到时，此线程阻塞。直到高优先级线程释放互斥量，它才解除阻塞，实际上直到高优先级线程挂起自己，它才会运行。
3. 低优先级线程一直紧凑循环，尝试使用非阻塞调用得到互斥量。因为它是最低优先级的线程，所以直到高、中优先级线程挂起，它才会成功得到互斥量。
4. 高优先级线程在挂起自己之前归还互斥量。
5. 中优先级线程得到互斥量，它所做的也仅是在挂起自己之前归还互斥量。此刻，高、中优先级线程都已挂起。
6. 低优先级线程得到互斥量，它首先在归还互斥量之前继续两个挂起的线程，因此低优先级线程暂时继承了最高线程优先级。

互斥量创建说明：

```
/* 定义互斥量 */
osMutexDef(osMutex);

/* 创建互斥量 */
osMutexId osMutex = osMutexCreate(osMutex(osMutex));
```

使用此例：

1. 生成应用代码并编程至 STM32 闪存
2. 当运行于调试模式时，请将下述变量添加到调试器的实时监测：HighPriorityThreadCycles、MediumPriorityThreadCycles 和 LowPriorityThreadCycles；这三个变量必须保持相等。 LED1、 LED2 和 LED4 应无限切换， LED3 仅在错误时打开

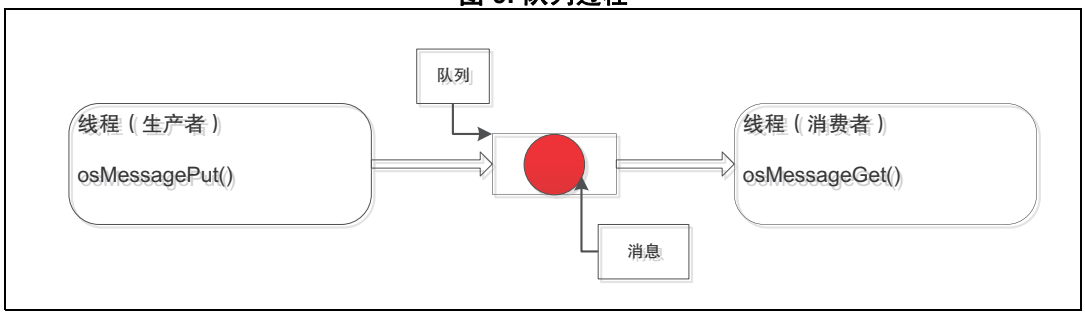
3.4 队列示例

队列是任务间通信的主要形式。可使用队列在任务间、在中断与任务间传递消息。在多数时候，它们作为线程安全的 FIFO（先进先出）缓冲使用，新数据发送到队尾，有时也可发送到队头。

本例创建了两个线程，它们向队列发送 / 从队列接收递增的数。一个线程作为生产者，另一个线程作为消费者。

消费者的优先级比生产者高，设置为读队列时阻塞。队列空间仅能容纳一个对象，一旦生产者向队列发布了一条消息，消费者将解除阻塞，抢占生产者运行，删除该对象。

图 9. 队列过程



队列创建说明：

```
/* 定义队列，“QUEUE_SIZE”项为 2 个字节 */
osMessageQDef(osqueue, QUEUE_SIZE, uint16_t);

/* 创建队列 */
osMessageQId osQueue = osMessageCreate (osMessageQ(osqueue), NULL);
```

使用此例：

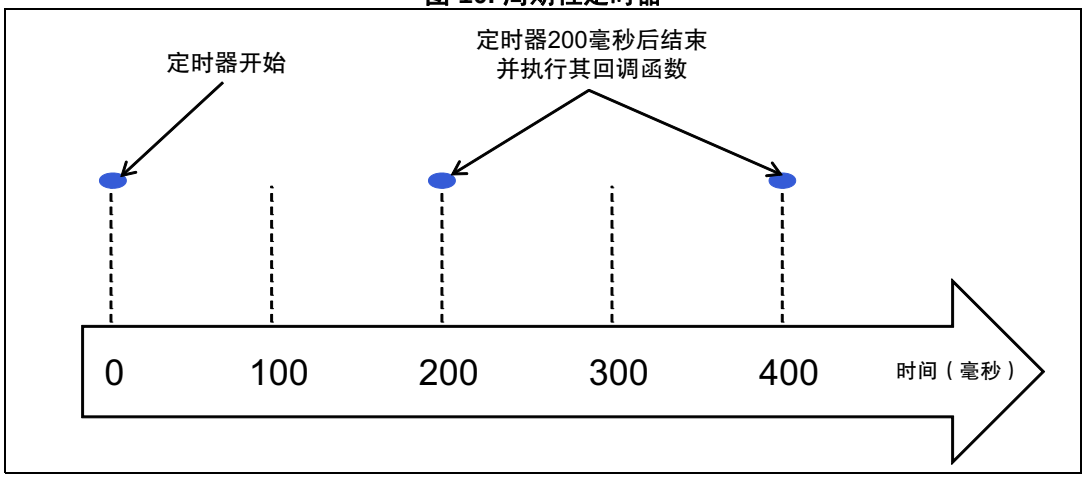
1. 生成应用代码并编程至 STM32 闪存
2. 运行该例，检查 LED1 是否在收到每条正确消息时切换，否则 LED3 切换。

3.5 定时器示例

定时器可令函数在未来某设定时间执行。由定时器执行的函数称为定时器的回调函数。从定时器启动到回调函数执行之间的时间称为定时器的周期。简单地说，当定时器周期结束时，执行回调函数。

本例演示了怎样基于 FreeRTOS API 使用 CMSIS RTOS API 定时器，创建的周期性定时器每隔 200 毫秒调用一次回调函数，切换评估板的 LED1。

图 10. 周期性定时器



周期性定时器创建说明：

```
/* 定义一个定时器， "osTimerCallback" 为其回调过程 */
osTimerDef(LEDTimer, osTimerCallback);

/* 创建定时器 */
osTimerId osTimer = osTimerCreate (osTimer(LEDTimer), osTimerPeriodic,
NULL);
```

使用此例：

1. 生成应用代码并编程至 STM32 闪存
2. 运行此列，检查 LED1 是否每 200 毫秒 （定时器周期结束）切换一次

注： *若要使用 FreeRTOS 软件定时器，请将 "timers.c" 添加到您的项目工作空间。*

3.6 低功耗示例

本例演示了如何在低功耗模式下使用 STM32 设备运行 FreeRTOS （若需 FreeRTOS 低功耗模式的更多信息，请参考*第 1.7 章节*）。

在 FreeRTOSConfig.h 中，将 configUSE_TICKLESS_IDLE 定义为 1，即可启用内置的无时间片空闲功能 （低功耗）

在本例中创建了两个线程和一个队列，它们具有下列功能：

- 第一个线程 "RxThread" 阻塞于队列，等待数据，每次收到数据时切换 LED（打开然后关闭），然后再次返回到阻塞于队列的状态。
- 第二个线程 "TxThread" 重复进入阻塞状态 500ms。当离开阻塞状态时，"TxThread" 通过队列向 "RxThread" 发送一条消息 （导致 "RxThread" 离开阻塞状态，切换 LED）。

当这两个线程阻塞时，内核停止时间片中断，将 STM32 置于低功耗（睡眠）模式以降低功耗。

表 4 显示了在所述的样例情况下， STM32F4 设备上测得的功耗。

表 4. 功耗比较

硬件平台	运行时模式	睡眠模式
STM324xG-EVAL	62.4 mA	14.2 mA
STM324x9I-EVAL	80.5 mA	20.8 mA

4 结论

本用户手册解释了如何在 STM32Cube HAL 驱动内集成 FreeRTOS 中间件组件。

本文说明了一组样例，以帮助用户基于 FreeRTOS 操作系统使用 CMSIS-RTOS API 开发应用。

5 FAQ

怎样将 FreeRTOS 移植到不同的 Cortex-M 内核？

若需将 FreeRTOS 移植到正确的 Cortex-M 产品，您必须从正确的目录导入 “port.c”。例如，若微控制器是带有 IAR 工具的 Cortex-M0 内核，则您必须从 “FreeRTOS\Source\portable\IAR\ARM_CM0” 获取 port.c。

FreeRTOS 使用多少 ROM/RAM？

这取决于您的编译器、架构，以及 RTOS 内核配置。一般来说， RTOS 内核本身需要大约 5 到 10 K 字节 ROM 空间。

如果创建的线程或队列数增加， RAM 使用量就会上升。

怎样设置 CPU 时钟？

CPU 时钟由 FreeRTOSConfig.h 中的 configCPU_CLOCK_HZ 定义，在 STM32CubeF4 固件内它由 SystemCoreClock 提供，表示 HCLK 时钟 （AHB 总线），当通过调用 SystemClock_Config() 函数配置 RCC 时钟时会设置此值。

怎样设置中断优先级？

任何使用 RTOS API 函数的中断服务程序，其优先级必须手动设置为大于等于 *FreeRTOSConfig.h* 文件中 *configMAX_SYSCALL_INTERRUPT_PRIORITY* 的设置值。

这确保了中断的逻辑优先级小于等于 *configMAX_SYSCALL_INTERRUPT_PRIORITY 设置*。

怎样使用非 SysTick 时钟生成时间片中断？

用户可选择性地自己提供时间片中断源，方法是使用非 SysTick 的定时器生成中断：

- 提供 vPortSetupTimerInterrupt() 的实现，它会以 configTICK_RATE_HZ FreeRTOSConfig.h 常量指定的频率生成中断。
- 将 xPortSysTickHandler() 安装为定时器中断的处理程序，确保 xPortSysTickHandler() 在 FreeRTOSConfig.h 中未映射至 SysTick_Handler()，且在 port.c 中未重命名为 SysTick_Handler()。

怎样启用无时间片空闲模式？

FreeRTOS 无时间片模式 （低功耗）通过进入睡眠模式并停止周期性的时间片中断来降低 MCU 功耗。在 FreeRTOSConfig.h 中，将 configUSE_TICKLESS_IDLE 定义为 1，即可启用此功能

当使用非 SysTick 定时器生成时间片中断时，也可启用无时间片空闲模式。用户必须添加下列动作至上一个问题所述内容：

- 在 FreeRTOSConfig.h 中，将 configUSE_TICKLESS_IDLE 设为 2。
- 按 FreeRTOS 网站的文档页面说明，定义 portSUPPRESS_TICKS_AND_SLEEP()。

6 修订历史

表 5. 文档修订历史

日期	修订	变更
2014 年 2 月 18 日	1	**初始版本。**
2014 年 6 月 23 日	2	**封面更新：** – 文件标题 – 参照 STM32Cube 中的 STM32CubeF4

请仔细阅读：

中文翻译仅为方便阅读之目的。该翻译也许不是对本文档最新版本的翻译，如有任何不同，以最新版本的英文原版文档为准。

本文档中信息的提供仅与 ST 产品有关。意法半导体公司及其子公司 （“ST”）保留随时对本文档及本文所述产品与服务进行变更、更正、修改或改进的权利，恕不另行通知。

所有 ST 产品均根据 ST 的销售条款出售。

买方自行负责对本文所述 ST 产品和服务的选择和使用，ST 概不承担与选择或使用本文所述 ST 产品和服务相关的任何责任。

无论之前是否有过任何形式的表示，本文档不以任何方式对任何知识产权进行任何明示或默示的授权或许可。如果本文档任何部分涉及任何第三方产品或服务，不应被视为 ST 授权使用此类第三方产品或服务，或许可其中的任何知识产权，或者被视为涉及以任何方式使用任何此类第三方产品或服务或其中任何知识产权的保证。

除非在 ST 的销售条款中另有说明，否则，ST 对 ST 产品的使用和 / 或销售不做任何明示或默示的保证，包括但不限于有关适销性、适合特定用途 （及其依据任何司法管辖区的法律的对应情况），或侵犯任何专利、版权或其他知识产权的默示保证。

意法半导体的产品不得应用于武器。此外，意法半导体产品也不是为下列用途而设计并不得应用于下列用途：（A）对安全性有特别要求的应用，例如，生命支持、主动植入设备或对产品功能安全有要求的系统； （B）航空应用； （C）汽车应用或汽车环境，且 / 或 （D）航天应用或航天环境。如果意法半导体产品不是为前述应用设计的，而采购商擅自将其用于前述应用，即使采购商向意法半导体发出了书面通知，采购商仍将独自承担因此而导致的任何风险，意法半导体的产品设计规格明确指定的汽车、汽车安全或医疗工业领域专用产品除外。根据相关政府主管部门的规定，ESCC、QML 或 JAN 正式认证产品适用于航天应用。

经销的 ST 产品如有不同于本文档中提出的声明和 / 或技术特点的规定，将立即导致 ST 针对本文所述 ST 产品或服务授予的任何保证失效，并且不应以任何形式造成或扩大 ST 的任何责任。

ST 和 ST 徽标是 ST 在各个国家或地区的商标或注册商标。

本文档中的信息取代之前提供的所有信息。

ST 徽标是意法半导体公司的注册商标。其他所有名称是其各自所有者的财产。

意法半导体集团公司

澳大利亚 - 比利时 - 巴西 - 加拿大 - 中国 - 捷克共和国 - 芬兰 - 法国 - 德国 - 中国香港 - 印度 - 以色列 - 意大利 - 日本 - 马来西亚 - 马耳他 - 摩洛哥 - 菲律宾 - 新加坡 - 西班牙 - 瑞典 - 瑞士 - 英国 - 美国

www.st.com