AN13094

在LPC55Sxx系列微控制器上使用 FreeRTOS 并支持TrustZone

第 0 版 — 2020年1月15日 应用笔记

作者：恩智浦半导体

目录

1 简介

LPC55Sxx 系列 MCU 是一款基于 Arm® Cortex®- M33 内核的微控制器，使用了支持TrustZone的ARMv8-M 架构。LPC55S69是高性能MCU之一，包括两个Cortex- M33内核，CPU0支持TrustZone-M的安全扩展。FreeRTOS 是一个轻量级的嵌入式操作系统；具有代码开源、可移植、可裁剪、调度策略灵活等特点；可方便地移植到各种嵌入式控制器中，已被广泛应用于各种嵌入式产品中。本文以LPC55S69为例，介绍如何在支持 TrustZone 的 ARMv8-M 处理器中使用FreeRTOS。

2 TrustZone技术特点

TrustZone 技术具有以下特点：

- 允许用户将内存映射划分为安全和非安全区域。
- 阻止未经授权的对安全区的代码/数据的调试访问。
- CPU 包括安全属性单元（SAU）和双份的（安全和非安全）的 NVIC、MPU、SYSTICK、核心控制寄存器等。安全/非安全代码可以访问自己分配的资源。
- 堆栈管理从原来的 Cortex-M 内核的主堆栈指针（MSP）和进程堆栈指针（PSP）两个寄存器扩展成4 个寄存器，来支持安全和非安全。
- 引入安全网关操作码的概念，非安全代码通过一套严格定义好的安全入口切如安全代码。

TrustZone 技术直接满足嵌入式系统的以下一些安全要求。

- **数据保护**

  敏感数据存储在安全存储空间中，只能由安全软件访问。只有经过安全检查或认证后，非安全软件才能访问调用为非安全域提供服务的安全API。

- **固件保护**

  预加载的固件存储在安全内存中，以防止它被逆向工程、破坏或恶意攻击。ARMv8-M 的 TrustZone 技术可以与额外的安全保护技术配合使用。例如，设备级读出保护是当今行业中常用的一种技术，可与 TrustZone 技术一起用于ARMv8-M 控制器来保护最终产品的固件。

- **操作保护**

  核心软件可以作为安全固件预加载，并且相应的外围设备可以配置为仅允许从安全状态下进行访问。通过这种方式，可以保护操作免受来自非安全端的入侵。

- **安全启动机制**

  安全启动机制可增强平台的安全性、保密性，因为它始终从安全内存启动。

3 安全环境配置

本节介绍如何配置安全环境，使用 TrustZone 技术保护系统重要资源。LPC55S69提供两级保护：CPU 级保护和系统级保护。TrustZone 位于 CPU0 内部，属于 CPU 级保护。此外，LPC55S69使用安全的 AHB 控制器提供一层系统级保护，如图 1所示。

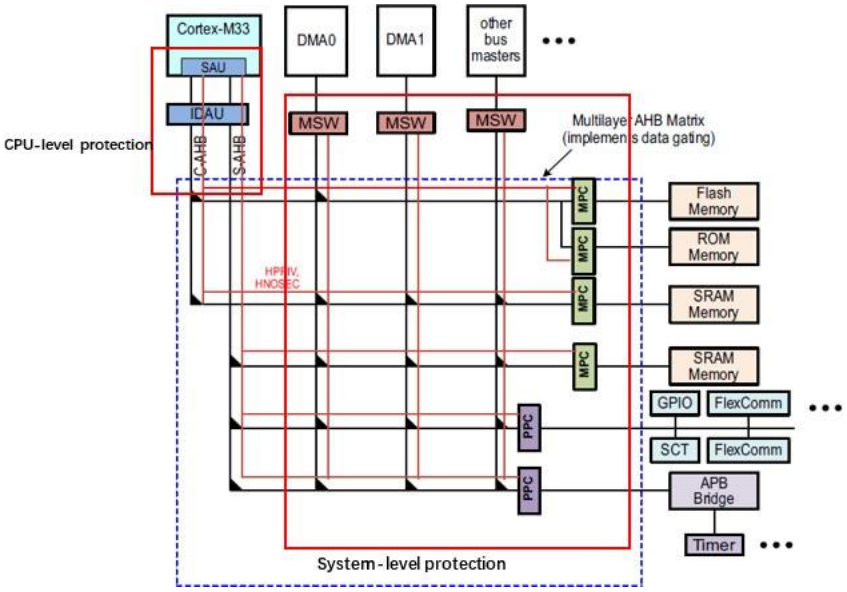

图1. LPC55S69的两级保护

LPC55S69 安全环境的配置包括 TrustZone 的配置和 Secure AHB 控制器的配置两部分。TrustZone 的配置主要是 SAU 的配置。

3.1 TEE工具

安全环境的配置可以通过手动配置相应的寄存器来实现，也可以使用恩智浦的可信执行环境（TEE）工具来实现。建议开发者使用 TEE 工具快速实现 TrustZone 和安全 AHB 控制器的配置。图 2是 TEE 工具的 GUI 界面。

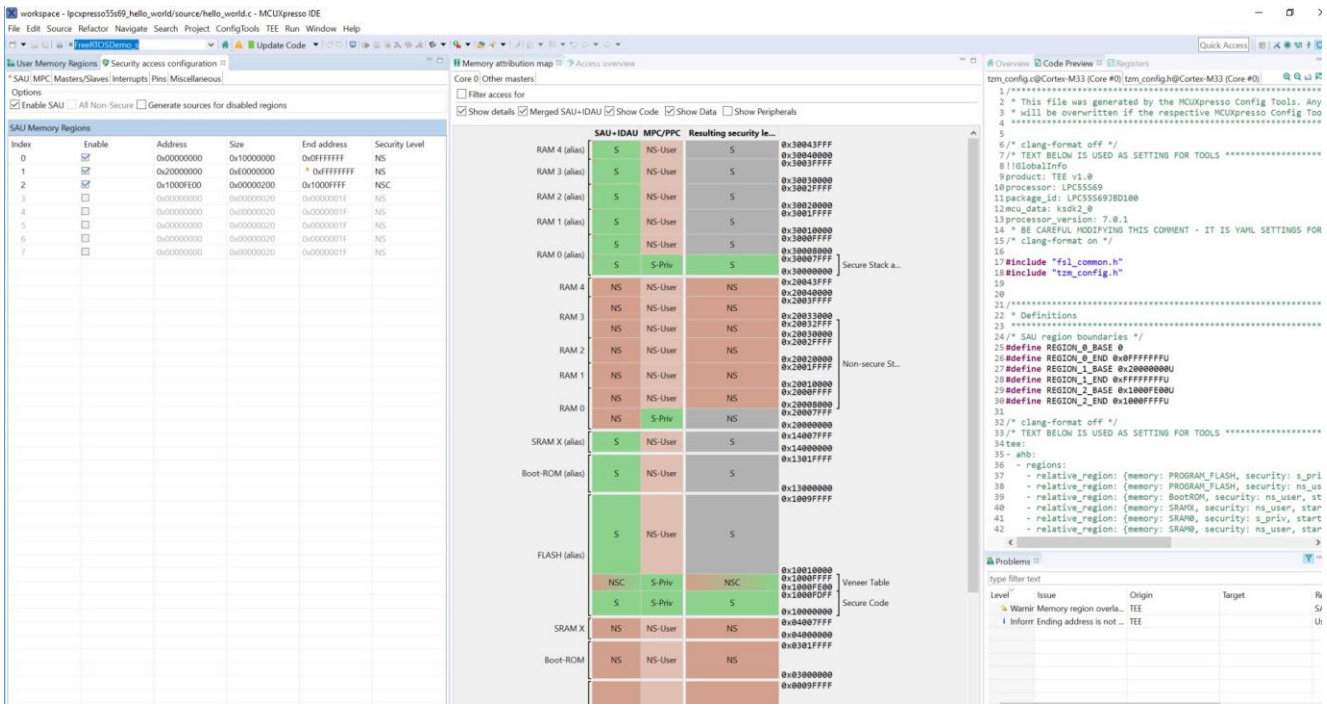

图2. TEE 工具的 GUI 界面

有关使用 TEE 工具的详细信息，请参阅 MCUXpresso 配置工具（桌面）用户指南（文档GSMCUXCTUG）。

3.2 安全/非安全状态切换

配置完 LPC55S69 的安全环境后，用户可以在实际项目中使用一些特殊的功能在安全和非安全状态之间进行切换。这里有两个特殊函数：非安全可调用函数（NSC function/Entry function）和非安全函数（Non-secure function）。

- 非安全可调用函数（NSC）

  NSC 函数是可以被非安全函数调用的安全函数。NSC 函数需要使用`attribute_((cmse_nonsecure_entry))`属性来定义。示例如下。

```
__attribute_((cmse_nonsecure_entry)) void vToggleGreenLED(void)
{
     /* Toggle the on-board green LED. */
     GPIO_PortToggle(GPIO, LED_PORT, (1U << GREEN_LED_PIN));
}
```

- 非安全函数

  非安全函数是可以被安全函数调用的函数。非安全函数需要使用`attribute_((cmse_nonsecure_call))`属性来定义。示例如下。

```
Typedef void _attribute_((cmse_nonsecure_call)) nsfunc(void);
Nsfunc *FunctionPointer;
FunctionPointer = cmse_nsfptr_create((nsfunc *) (0x21000248u));
If (cmse_is_nsfptr(FunctionPointer))
FunctionPointer();
```

注意

非安全函数必须通过函数指针的方式调用。

更多关于安全/非安全状态切换的信息，请参阅有关 ARMv8-M 架构的 TrustZone 技术的文档。

4 FreeRTOS 中 TrustZone 的使用

本节介绍如何在 LPC55S69 中运行 FreeRTOS 并启用TrustZone。FreeRTOS 官方提供了在LPC55S69 上支持Trustzone 运行的 FreeRTOS 示例。FreeRTOS 可以从 FreeRTOS 下载。LPC55S69 示例的路径为 *FreeRTOS/Demo/CORTEX_MPU_M33F_NXP_LPC55S69_MCUXpresso*，如图 3所示。

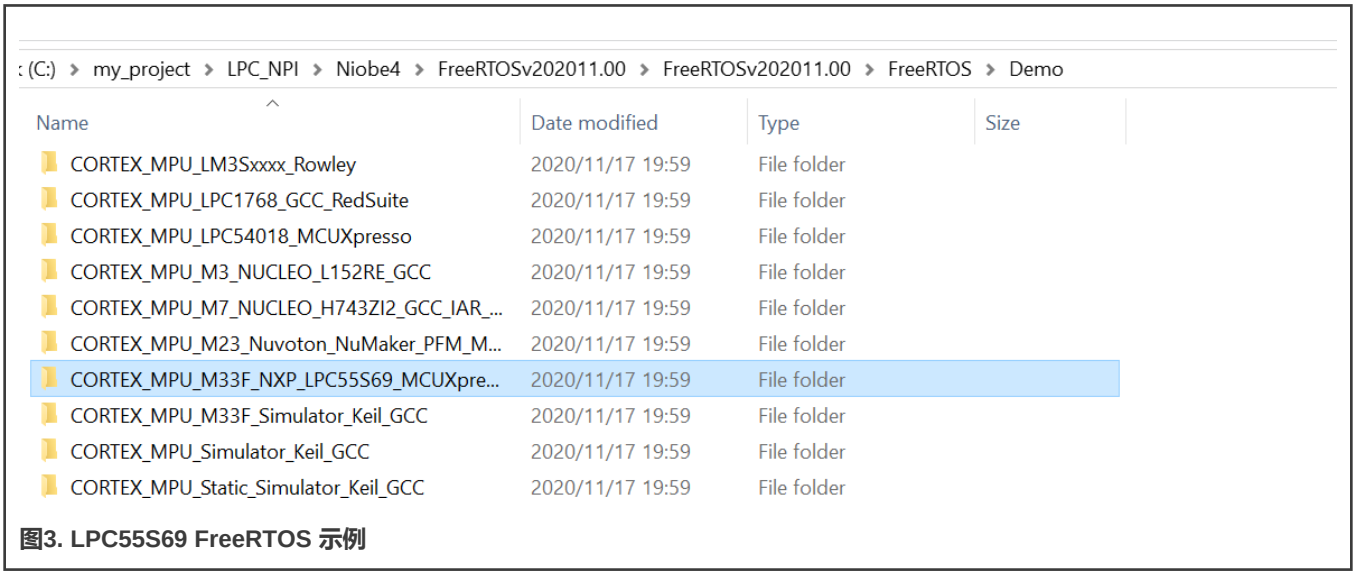

图3. LPC55S69 FreeRTOS 示例

FreeRTOS 提供此演示的文档， 他们可以从RTOS下载。

NXP 的 LPC55S69 SDK 包也提供了一个支持 TrustZone 的 FreeRTOS 工程示例。项目名称为*freertos_tzm*，示例路径为 */SDK_2.8.2_LPCXpresso55S69_IAR/boards/lpcxpresso55s69/rtos_examples/freertos_tzm*。*freertos_tzm* 项目如图 4所示。

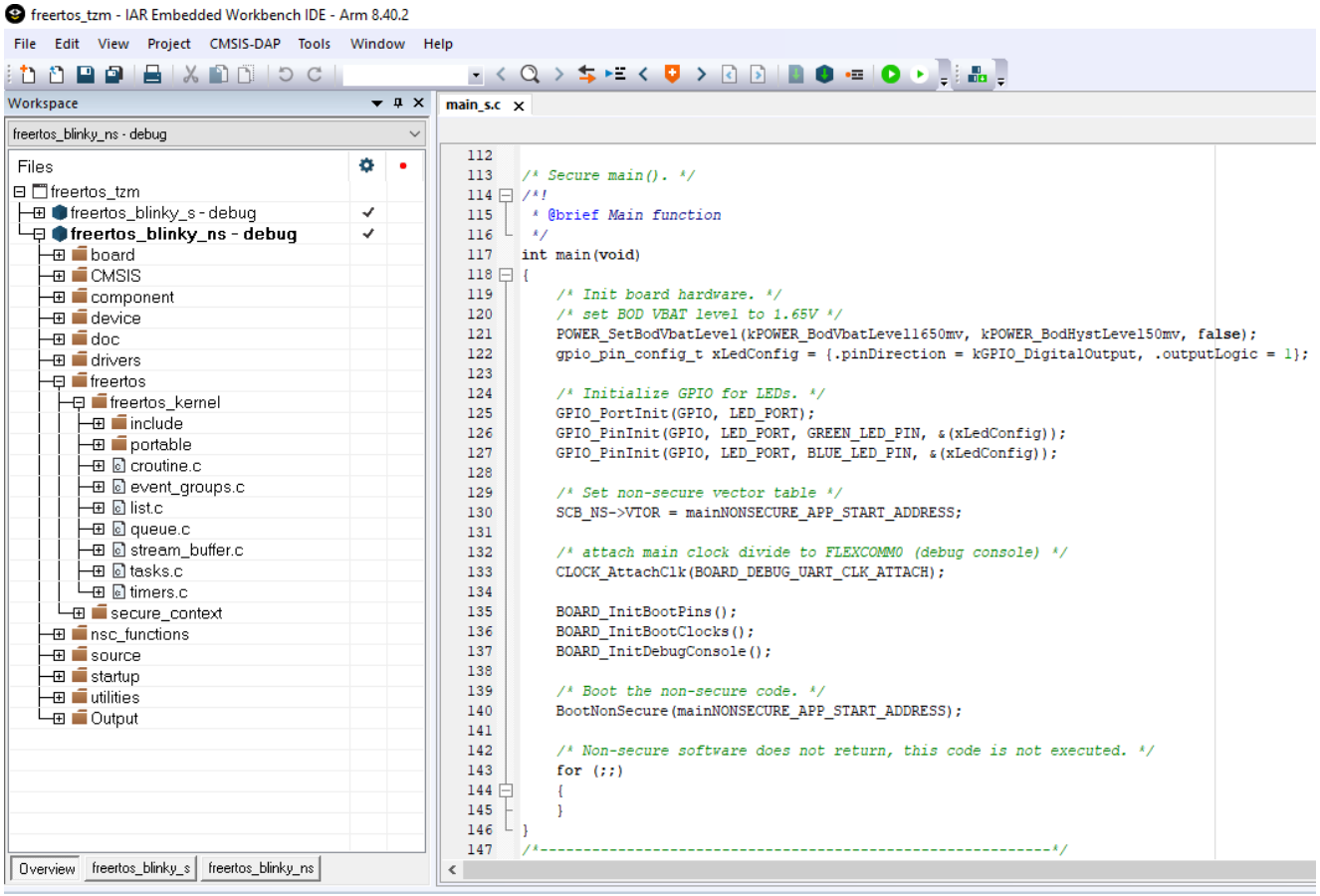

图4. SDK v2.8 中的 LPC55S69 freertos_tzm 示例

4.1 SDK v2.8 中使用 TrustZone 的 FreeRTOS 示例

本节以 SDK v2.8 中的 *freertos_tzm* 项目为例，介绍 FreeRTOS 支持 TrustZone 的工作流程。*freertos_tzm* 示例的工作流程如图 5所示。

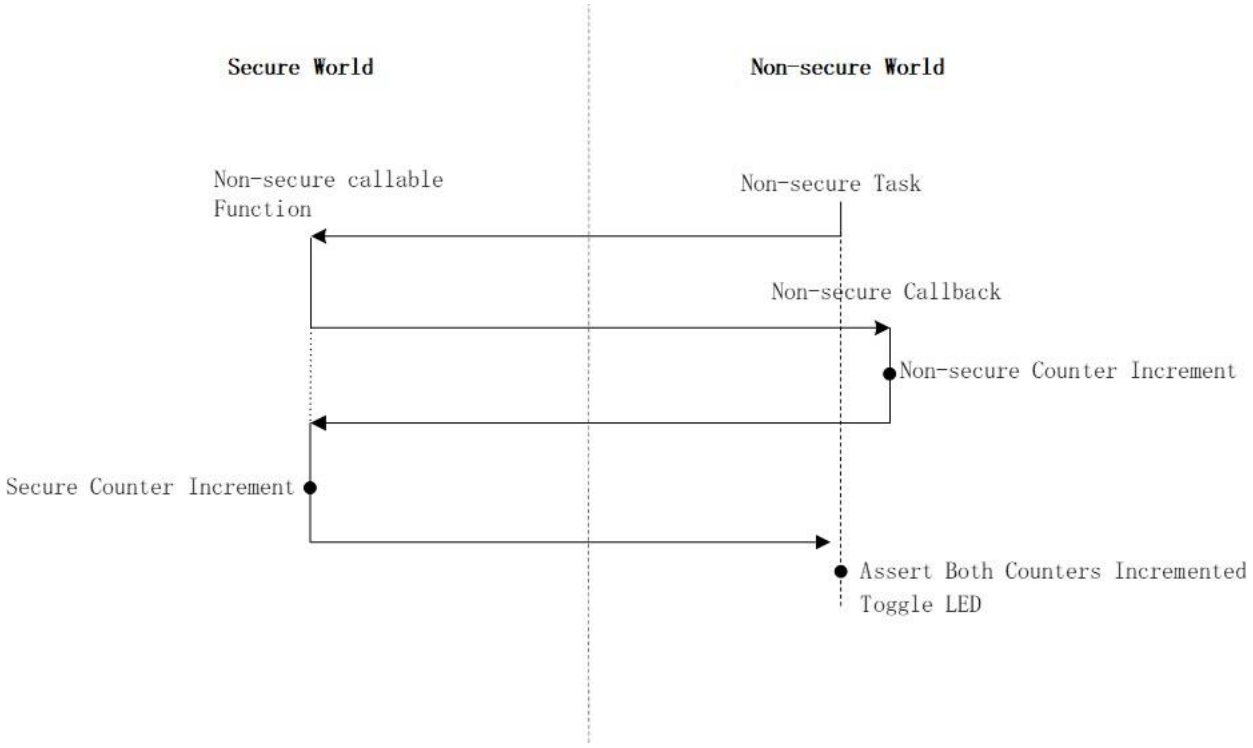

图5. freertos_tzm 演示的工作流程

本项目的内存分区如图 6所示。

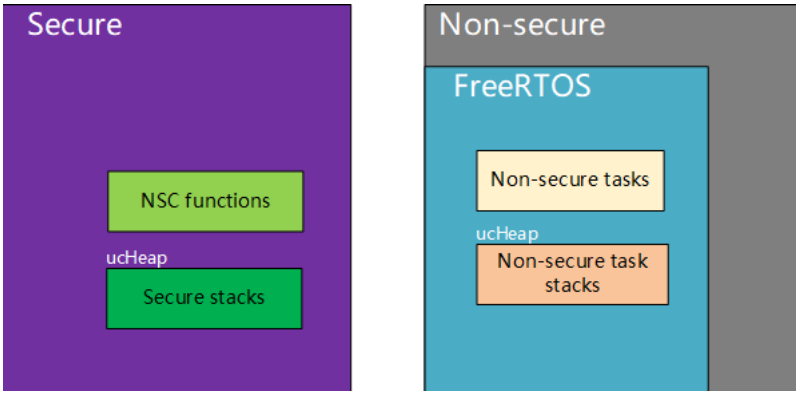

图 6. freertos_tzm 示例中的内存分区

用户可以将一些重要的资源存储在一个安全的区域。FreeRTOS 内核存储在非安全区域，任务创建和调度过程也在非安全区域完成。这些非安全任务可以通过调用 NSC 函数来访问安全区域中的某些特定资源，安全函数可以调用非安全函数跳转到非安全区域。

4.1.1 创建用户任务

在 freertos_tzm 示例中，创建了两个用户任务，如图 7所示。

```
144  static void prvCreateTasks(void)
145  {
146      /* Create the secure calling task. */
147      (void)xTaskCreate(prvSecureCallingTask,          /* The function that implements the demo task. */
148                        "ScCall",                      /* The name to assign to the task being created. */
149                        configMINIMAL_STACK_SIZE + 100, /* The size, in WORDS (not bytes), of the stack to allocate for
150                                                           the task being created. */
151                        NULL,                          /* The task parameter is not being used. */
152                        portPRIVILEGE_BIT | tskIDLE_PRIORITY, /* The priority at which the task being created will run. */
153                        NULL);
154
155      /* Create the LED toggling task. */
156      (void)xTaskCreate(prvLEDTogglingTask,            /* The function that implements the demo task. */
157                        "LedToggle",                   /* The name to assign to the task being created. */
158                        configMINIMAL_STACK_SIZE + 100, /* The size, in WORDS (not bytes), of the stack to allocate for
159                                                           the task being created. */
160                        NULL,                          /* The task parameter is not being used. */
161                        portPRIVILEGE_BIT | tskIDLE_PRIORITY, /* The priority at which the task being created will run. */
162                        NULL);
163  }
```

图7. 在非安全环境中创建用户任务

当用户使用`xTaskCreate()`函数在非安全区域创建任务时，会分配一个非安全任务栈。当任务切换时，非安全context可以存储在非安全任务堆栈中。

4.1.1.1 分配安全堆栈

由于 TrustZone 技术将系统内存划分为安全区域和非安全区域，因此在执行非安全任务时可以调用NSC 函数。因此，还需要为调用 NSC 函数的非安全任务分配一个安全堆栈来存储安全上下文（context）。安全栈的分配在第一次执行非安全任务时完成，如图 8所示。

```
175  static void prvSecureCallingTask(void *pvParameters)
176  {
177      uint32_t ulLastSecureCounter = 0, ulLastNonSecureCounter = 0;
178      uint32_t ulCurrentSecureCounter = 0;
179
180      /* This task calls secure side functions. So allocate a
181       * secure context for it. */
182      portALLOCATE_SECURE_CONTEXT(configMINIMAL_SECURE_STACK_SIZE);
183
184      for (;;)
185      {
186          /* Call the secure side function which does two things:
187           * - It calls the supplied function (prvCallback) which in turn
188           *   increments the non-secure counter.
189           * - It increments the secure counter and returns the incremented value.
190           * Therefore at the end of this function call both the secure and
191           * the non-secure counters must have been incremented.
192           */
193          ulCurrentSecureCounter = NSCFunction(prvCallback);
194
195          /* Make sure that both the counters are incremented. */
196          configASSERT(ulCurrentSecureCounter == ulLastSecureCounter + 1);
197          configASSERT(ulNonSecureCounter == ulLastNonSecureCounter + 1);
```

图8. 分配安全堆栈

在调用`portALLOCATE_SECURE_CONTEXT()`函数时，会触发一个SVC 中断，调用`SecureContext_AllocateContext()`函数（NSC 函数）在安全ucHeap 数组中分配一个安全空间作为这个非安全任务的安全栈，以及`PSP_S` and `PSPLIM_S` 被初始化。

4.1.1.2 定义 NSC 功能

创建任务后，用户需要定义任务函数。如果非安全任务需要调用 NSC 函数，则还需要定义相应的 NSC 函数。NSC 函数在安全项目工程中的 `nsc_functions.c`文件中定义。`freertos_tzm` 项目所需的NSC 函数如表 1所示。

表1. NSC 函数

NSC function	`uint32_t NSCFunction(Callback_t pxCallback)`
	`void vToggleGreenLED(void)`
	`void vToggleBlueLED(void)`
	`uint32_t getSystemCoreClock(void)`

4.1.2 任务切换

对于支持 TrustZone 的 MCU，FreeRTOS 中的context 切换与基于 cortex-M3、M4 和 M7 内核的 MCU 不同。本节介绍支持 TrustZone 的 FreeRTOS 的任务切换过程。由于涉及安全函数和非安全函数之间的相互调用，CPU 在安全和非安全区域使用了一些 banked core 寄存器，如PSP、MSP、PSPLIM、MSPLIM、CONTROL 寄存器等。它使 FreeRTOS 任务切换过程比较复杂。

4.1.2.1 PendSV中断服务流程

FreeRTOS 的任务切换过程在 PendSV 中断服务函数中完成。FreeRTOS 支持 TrustZone 的任务切换流程如图 9所示。

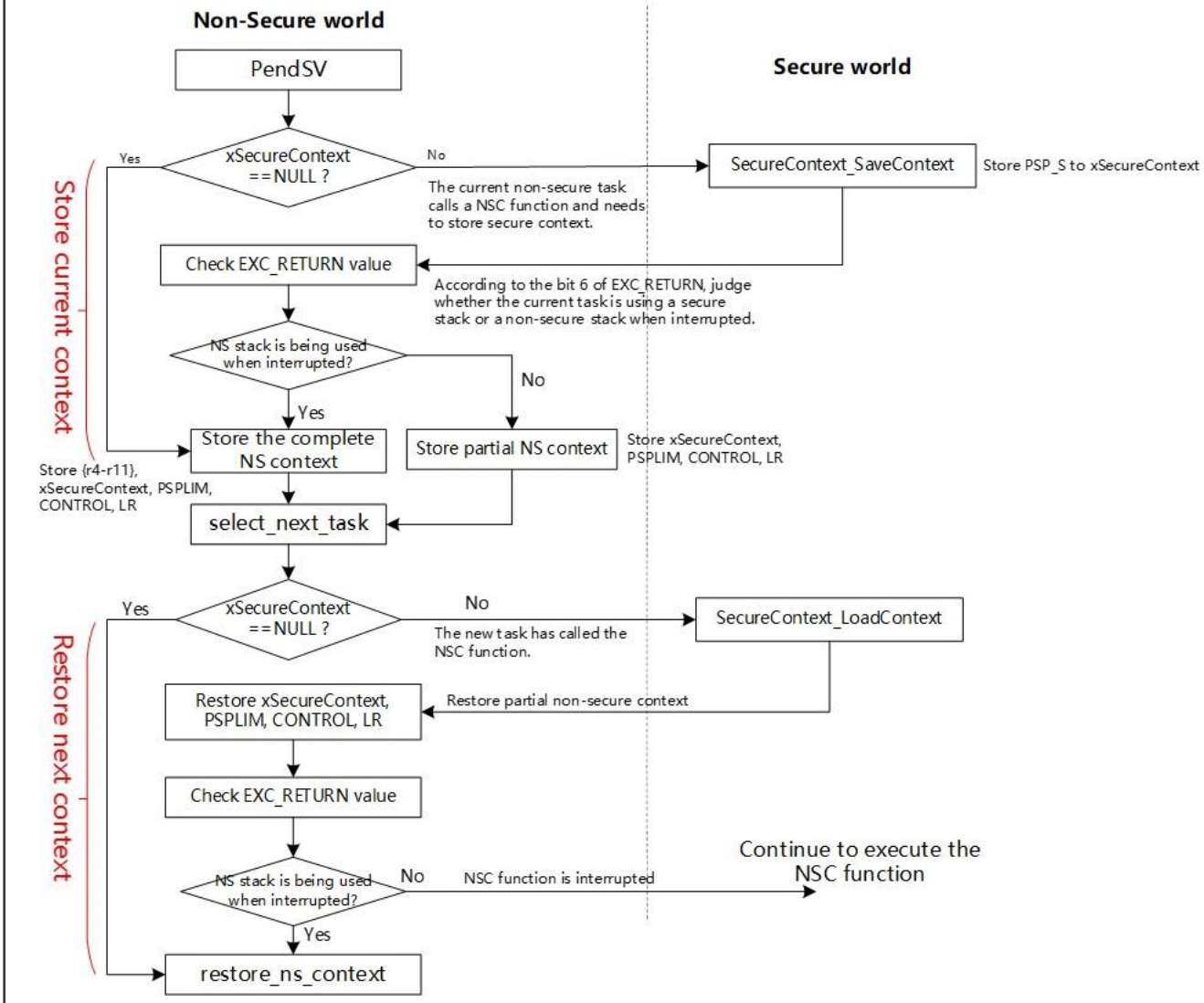

图9. 支持 TrustZone 的 FreeRTOS 任务切换流程

注意

图 9 中完整的 NS context 包含 12 个值：{R4-R11}、xSecureContext、PSPLIM、CONTROL、LR。NS context 仅包含四个值：xSecureContext、PSPLIM、CONTROL、LR。

4.1.2.2 存储当前任务的上下文Context

在进行任务切换时，用户需要先存储当前任务的上下文。基于 CM33 内核的 MCU 在执行非安全任务时可能会调用 NSC 功能。因此，在存储当前任务上下文时，需要先判断当前任务是否调用了NSC 函数，即是否存在与当前任务相关的安全上下文（根据变量 xSecureContext 的值来判断 NSC 是否调用了NSC 函数函数被调用，如果变量值不为 0，则表示调用的是安全函数）：

- 如果有安全上下文，程序必须首先跳转到安全区域来存储安全上下文。将`PSP_S`和`PSPLIM_S`的值存储到相应的安全堆栈中。
  - 存储安全上下文后，需要判断NSC 函数或非安全函数在任务切换时是否被中断。可以根据EXC_RETURN/LR 的bit 6 的值来判断，如果bit 6 为 1，则表示NSC 函数的执行被中断；如果第 6 位为 0，则表示非安全功能中断。如果 NSC 函数被中断，因为 {r4 -r11} 寄存器在调用NSC 函数之前已经存储到非安全栈中，并且安全 context 也已经存储到安全栈中，因此只有部分非安全上下文（xSecureContext , PSPLIM, CONTROL, LR）需要存储。

— 如果非安全功能被中断，用户需要存储完整的非安全context，包括{r4-r11}、xSecureContext、PSPLIM、CONTROL、LR。

— 如果当前任务没有调用 NSC 函数，用户只需存储完整的非安全上下文，如图 10所示。

```
212     save_ns_context:
213         ldr r3, =pxCurrentTCB               /* Read the location of pxCurrentTCB i.e. &( pxCurrentTCB ). */
214         ldr r2, [r3]                        /* Read pxCurrentTCB. */
215     #if ( configENABLE_FPU == 1 )
216         tst lr, #0x10                       /* Test Bit[4] in LR. Bit[4] of EXC_RETURN is 0 if the FPU is in use. */
217         it eq
218         vstmdbeq r1!, {s16-s31}             /* Store the FPU registers which are not saved automatically. */
219     #endif /* configENABLE_FPU */
220     #if ( configENABLE_MPU == 1 )
221         subs r1, r1, #48                    /* Make space for xSecureContext, PSPLIM, CONTROL, LR and the remaining registers on the stack. */
222         str r1, [r2]                        /* Save the new top of stack in TCB. */
223         adds r1, r1, #16                    /* r1 = r1 + 16. */
224         stm r1, {r4-r11}                    /* Store the registers that are not saved automatically. */
225         mrs r2, psplim                      /* r2 = PSPLIM. */
226         mrs r3, control                     /* r3 = CONTROL. */
227         mov r4, lr                          /* r4 = LR/EXC_RETURN. */
228         subs r1, r1, #16                    /* r1 = r1 - 16. */
229         stm r1, {r0, r2-r4}                 /* Store xSecureContext, PSPLIM, CONTROL and LR on the stack. */
230     #else /* configENABLE_MPU */
231         subs r1, r1, #44                    /* Make space for xSecureContext, PSPLIM, LR and the remaining registers on the stack. */
232         str r1, [r2]                        /* Save the new top of stack in TCB. */
233         adds r1, r1, #12                    /* r1 = r1 + 12. */
234         stm r1, {r4-r11}                    /* Store the registers that are not saved automatically. */
235         mrs r2, psplim                      /* r2 = PSPLIM. */
236         mov r3, lr                          /* r3 = LR/EXC_RETURN. */
237         subs r1, r1, #12                    /* r1 = r1 - 12. */
238         stmia r1!, {r0, r2-r3}              /* Store xSecureContext, PSPLIM and LR on the stack. */
239     #endif /* configENABLE_MPU */
```

图10. 存储非安全context

注意

`freertos_tzm` 项目默认启用了 MPU。

4.1.2.3 选择下一个准备好的任务

存储当前任务的上下文后，程序会跳转到 *select_next_task* 阶段，在任务就绪表中搜索优先级最高的任务，如图 11所示。

```
241     select_next_task:
242         mov r0, #configMAX_SYSCALL_INTERRUPT_PRIORITY
243         msr basepri, r0                     /* Disable interrupts upto configMAX_SYSCALL_INTERRUPT_PRIORITY. */
244         dsb
245         isb
246         bl vTaskSwitchContext
247         mov r0, #0                          /* r0 = 0. */
248         msr basepri, r0                     /* Enable interrupts. */
249
250         ldr r2, =pxCurrentTCB               /* Read the location of pxCurrentTCB i.e. &( pxCurrentTCB ). */
251         ldr r3, [r2]                        /* Read pxCurrentTCB. */
252         ldr r1, [r3]                        /* The first item in pxCurrentTCB is the task top of stack. r1 now points to the top of stack. */
```

图11. 选择下一个任务

4.1.2.4 恢复下一个任务的上下文

上下文恢复和存储的过程类似。需要先恢复一些非安全上下文（xSecureContext、PSPLIM、CONTROL、LR），然后根据变量 xSecureContext 的值判断下一个任务是否调用NSC 函数， 即是否存在安全上下文。

- 如果有安全 context，
  1. 恢复安全 context。

2. 恢复 `PSP_S` 和 `PSPLIM_S` 的值。
3. 返回非安全区域恢复一些非安全context，xSecureContext、PSPLIM、CONTROL、LR。
4. 根据 `EXC_RETURN` 的值判断是 NSC 函数的执行被中断还是非安全函数的执行被中断。
   - 如果非安全函数的执行中断，用户需要继续恢复 r4 – r11 的值。
   - 如果NSC 函数的执行被中断，直接退出PendSV 中断，因为安全上下文已经恢复。

- 如果没有安全上下文，用户只需恢复非安全上下文。

```
280     #if ( configENABLE_MPU == 1 )
281         ldmia r1!, {r0, r2-r4}          /* Read from stack - r0 = xSecureContext, r2 = PSPLIM, r3 = CONTROL and r4 = LR. */
282         msr psplim, r2                  /* Restore the PSPLIM register value for the task. */
283         msr control, r3                 /* Restore the CONTROL register value for the task. */
284         mov lr, r4                      /* LR = r4. */
285         ldr r2, =xSecureContext         /* Read the location of xSecureContext i.e. &( xSecureContext ). */
286         str r0, [r2]                    /* Restore the task's xSecureContext. */
287         cbz r0, restore_ns_context      /* If there is no secure context for the task, restore the non-secure context. */
288         push {r1,r4}
289         bl SecureContext_LoadContext    /* Restore the secure context. */
290         pop {r1,r4}
291         mov lr, r4                      /* LR = r4. */
292         lsls r2, r4, #25                /* r2 = r4 << 25. Bit[6] of EXC_RETURN is 1 if secure stack was used, 0 if non-secure stack was used to store stack frame. */
293         bpl restore_ns_context          /* bpl - branch if positive or zero. If r2 >= 0 ==> Bit[6] in EXC_RETURN is 0 i.e. non-secure stack was used. */
294         msr psp, r1                     /* Remember the new top of stack for the task. */
295         bx lr
296     #else /* configENABLE_MPU */
297         ldmia r1!, {r0, r2-r3}          /* Read from stack - r0 = xSecureContext, r2 = PSPLIM and r3 = LR. */
298         msr psplim, r2                  /* Restore the PSPLIM register value for the task. */
299         mov lr, r3                      /* LR = r3. */
300         ldr r2, =xSecureContext         /* Read the location of xSecureContext i.e. &( xSecureContext ). */
301         str r0, [r2]                    /* Restore the task's xSecureContext. */
302         cbz r0, restore_ns_context      /* If there is no secure context for the task, restore the non-secure context. */
303         push {r1,r3}
304         bl SecureContext_LoadContext    /* Restore the secure context. */
305         pop {r1,r3}
306         mov lr, r3                      /* LR = r3. */
307         lsls r2, r3, #25                /* r2 = r3 << 25. Bit[6] of EXC_RETURN is 1 if secure stack was used, 0 if non-secure stack was used to store stack frame. */
308         bpl restore_ns_context          /* bpl - branch if positive or zero. If r2 >= 0 ==> Bit[6] in EXC_RETURN is 0 i.e. non-secure stack was used. */
309         msr psp, r1                     /* Remember the new top of stack for the task. */
310         bx lr
311     #endif /* configENABLE_MPU */
312
313     restore_ns_context:
314         ldmia r1!, {r4-r11}             /* Restore the registers that are not automatically restored. */
315     #if ( configENABLE_FPU == 1 )
316         tst lr, #0x10                   /* Test Bit[4] in LR. Bit[4] of EXC_RETURN is 0 if the FPU is in use. */
317         it eq
318         vldmiaeq r1!, {s16-s31}         /* Restore the FPU registers which are not restored automatically. */
319     #endif /* configENABLE_FPU */
320         msr psp, r1                     /* Remember the new top of stack for the task. */
321         bx lr
```

Determine whether there is a secure context according to xSecureContext

Exit PendSV directly to execute the NSC function

Restore non-secure context {r4-r11}

图12. 恢复下一个任务上下文

4.1.3 运行 freertos_tzm 示例

在此示例中，创建了两个用户任务。

- `prvSecureCallingTask`

  此任务调用 NSC 函数将 ulSecureCounter 加 1，然后调用非安全回调将 ulNonSecureCounter 加 1。判断 `ulSecureCounter` 和 `ulNonSecureCounter` 是否都加了1，然后切换绿色LED。工作流如图 5所示，此任务每秒执行一次。

- `prvLEDTogglingTask`

  切换蓝色 LED 一次。该任务每秒执行一次。

运行程序后，我们可以发现绿色LED 和蓝色LED 交替闪烁，闪烁周期为 1 秒。

4.2 一种安全使用 FreeRTOS 的方式

某些情况下，将 FreeRTOS 内核和用户任务放在非安全区域不足以满足系统的安全要求。这时候，我们可以考虑将 FreeRTOS 内核和一些重要的任务放在安全的区域和非安全区域中的其他区域，从而确保操作系统免受非安全攻击，提高系统的安全性。

本节介绍一种在 FreeRTOS 中支持 TrustZone 的新方法：将 FreeRTOS 内核和重要任务放在安全区域，将其他用户任务放在非安全区域，如图 13所示。

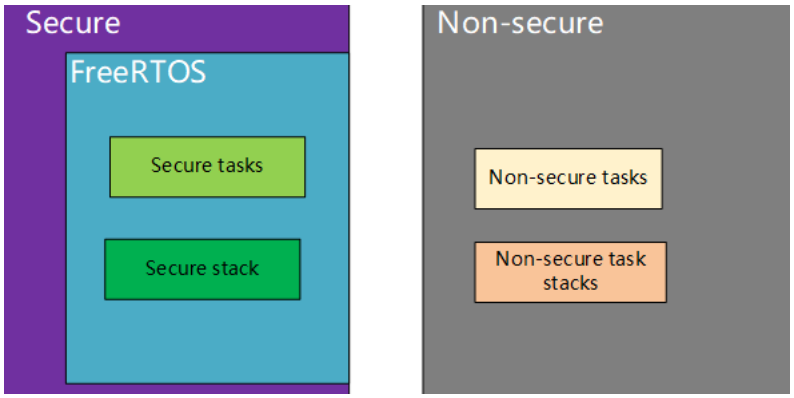

图13. 一种更安全的 FreeRTOS 使用方式

有关更详细信息，请参阅Method to support TrustZone-M in FreeRTOS。用户可以参考这个方法，让他们的系统更加安全。

5 参考资料

- *User Guide for MCUXpresso Config Tools (Desktop)* (document GSMCUXCTUG)
- *LPC55S6x/LPC55S2x/LPC552x User manual* (document UM11126)
- TrustZone technology for ARMv8-M Architecture
- Method to support TrustZone-M in FreeRTOS

How To Reach Us

Home Page:

nxp.com

Web Support:

nxp.com/support

For more information, please visit: http://www.nxp.com
For sales office addresses, please send an email to: salesaddresses@nxp.com

Date of release: January 15, 2020
Document identifier: AN13094

arm