AN13094
Using FreeRTOS on LPC55Sxx Series Microcontrollers with TrustZone

Rev. 0 — January 15, 2020

Application Note

by: NXP Semiconductors

Contents

1 Introduction......................................1
2 Features of TrustZone technology 1
3 Security environment configuration 2
3.1 TEE tool...................................... 2
3.2 Secure/Non-secure state switch 3
4 FreeRTOS usage with TrustZone... 4
4.1 FreeRTOS example with TrustZone in SDK v2.8................5
4.2 A safer way to use FreeRTOS... 11
5 Reference.....................................12

1 Introduction

The LPC55Sxx series MCU is a microcontroller based on the Arm® Cortex®-M33 core, using the ARMv8-M architecture with TrustZone enabled. LPC55S69 is one of the high-performance MCUs, including two Cortex-M33 cores, and CPU0 supports the security extension of TrustZone-M. FreeRTOS is a lightweight embedded operating system. It has the characteristics of open source code, portability, tailorability, and flexible scheduling strategy. It can be easily transplanted to various embedded controllers and has been widely used in various embedded products. This document takes an LPC55S69 as example to describe how to use FreeRTOS in an ARMv8-M processor that supports TrustZone.

2 Features of TrustZone technology

TrustZone technology has the following features:

- Allows users to divide memory map into Secure and Non-Secure regions.
- Blocks the debugging of secure code/data when not authenticated.
- CPU includes Security Attribution Unit (SAU) as well as a duplication of NVIC, MPU, SYSTICK, core control registers, etc. Secure/Non-Secure codes have access to their own allocated resources.
- Stack management expands from two-stack pointers in original Cortex-M, Main Stack Pointer (MSP) and Process Stack Pointer (PSP), to four, providing the above pairs individually to both Secure and Non-Secure.
- Introduces the concept of Secure Gateway opcode to allow secure code to define a strict set of entry points into it from a Non-secure code.

TrustZone technology address some of the following security requirements of embedded systems directly.

- Data protection

  Sensitive data are stored in Secure memory spaces and are only accessed by Secure software. Only after security check or authentication, non-secure software can access to Secure APIs providing services to the Non-secure domain.

- Firmware protection

  The pre-loaded firmware is stored in Secure memories to prevent it from being reverse engineered and compromised by malicious attacks. TrustZone technology for ARMv8-M can work with extra protection techniques. For example, device level read-out protection, a technique that is commonly used in the industry today, can be used with TrustZone technology for ARMv8-M to protect the completed firmware of the final product.

- Operation protection

  Software for critical operations can be pre-loaded as Secure firmware and the appropriate peripherals can be configured to permit access from the Secure state only. In this way, the operations can be protected from intrusion from the Non-secure side.

- Secure boot

The Secure boot mechanism enables the confidence in the platform, as it will always boot from Secure memory.

3 Security environment configuration

This section introduces how to configure the security environment to use TrustZone technology to protect the important resources of the system. LPC55S69 provides two levels of protection: CPU-level protection and system-level protection. TrustZone is located inside CPU0 and belongs to CPU-level protection. In addition, LPC55S69 uses secure AHB controller to provide a layer of system-level protection, as shown in Figure 1.

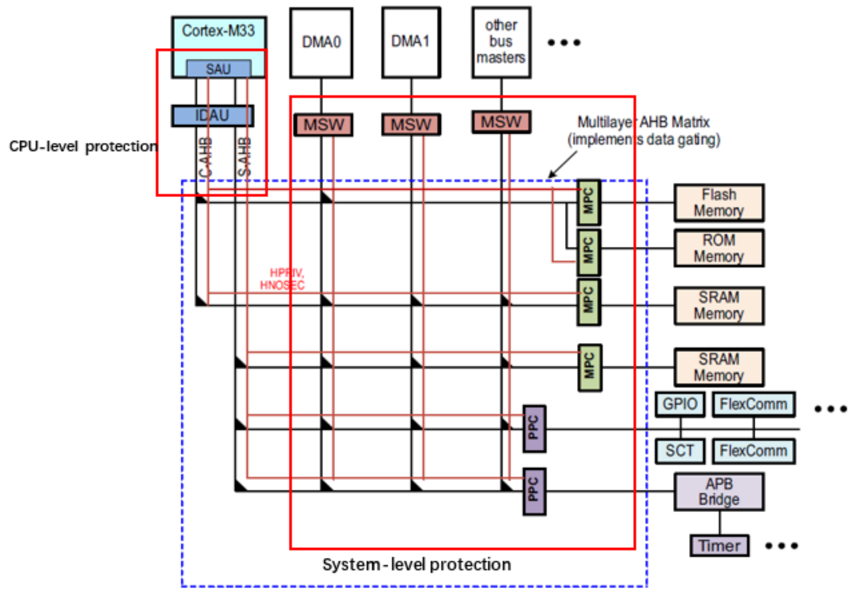
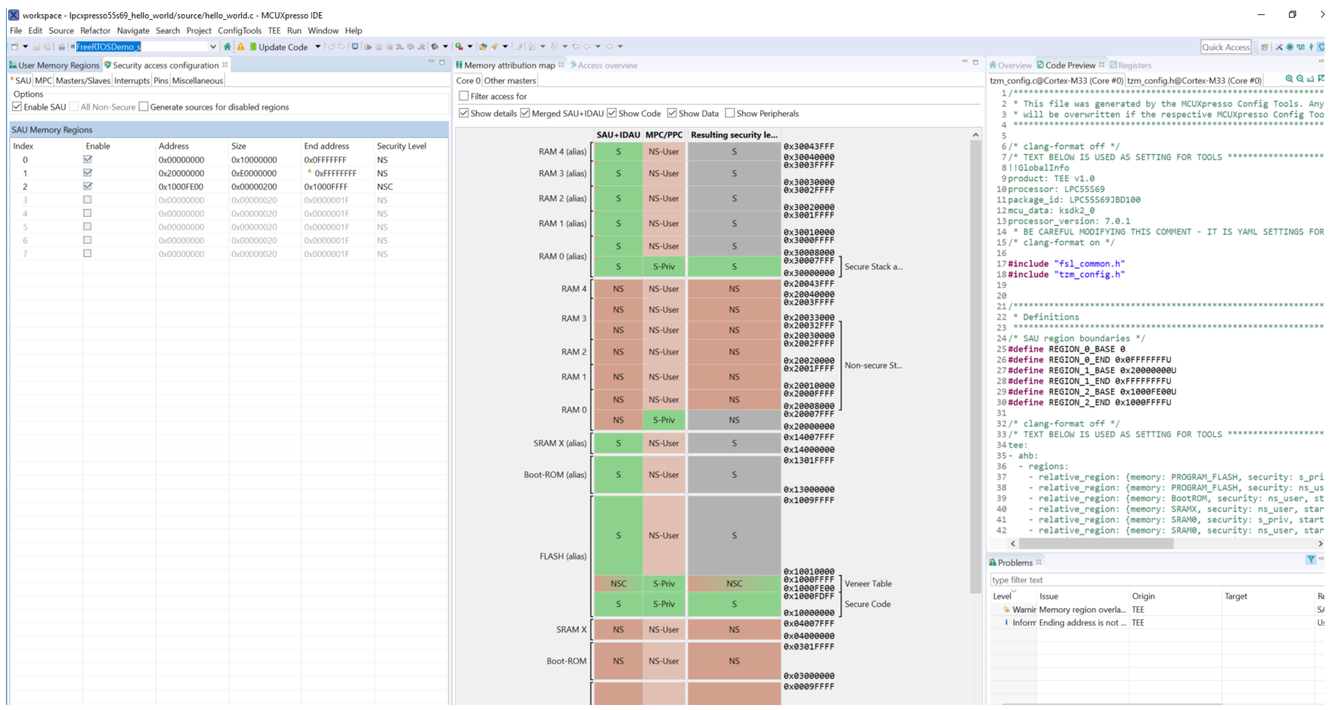


Figure 1. Two-level protection of LPC55S69

The configuration of the LPC55S69 security environment includes two parts: the configuration of TrustZone and the configuration of the Secure AHB controller. The configuration of TrustZone is mainly the configuration of SAU.

3.1 TEE tool

The configuration of the security environment can be implemented by manually configuring the corresponding registers, or by using NXP's Trusted Execution Environment (TEE) tool. It is recommended that developers use TEE tool to quickly implement the configuration of TrustZone and secure AHB controller. Figure 2 shows the GUI interface of TEE tool.


workspace - lpcxpresso55s69_hello_world/source/hello_world.c - MCUXpresso IDE
File Edit Source Refactor Navigate Search Project ConfigTools TEE Run Window Help
FreeRTOSDemo_s
Update Code
Quick Access
User Memory Regions
Security access configuration
SAU MPC Masters/Slaves Interrupts Pins Miscellaneous
Memory attribution map
Access overview
Core 0 Other masters
Overview
Code Preview
Registers
tzm_config.c@Cortex-M33 (Core #0)
tzm_config.h@Cortex-M33 (Core #0)
Problems
type filter text

Options

[✓] Enable SAU [ ] All Non-Secure [ ] Generate sources for disabled regions

SAU Memory Regions

Index	Enable	Address	Size	End address	Security Level
0	✓	0x00000000	0x10000000	0x0FFFFFFF	NS
1	✓	0x20000000	0xE0000000	0xFFFFFFFF	NS
2	✓	0x1000FE00	0x00000200	0x1000FFFF	NSC
3	☐	0x00000000	0x00000020	0x0000001F	NS
4	☐	0x00000000	0x00000020	0x0000001F	NS
5	☐	0x00000000	0x00000020	0x0000001F	NS
6	☐	0x00000000	0x00000020	0x0000001F	NS
7	☐	0x00000000	0x00000020	0x0000001F	NS

[ ] Filter access for

[✓] Show details [✓] Merged SAU+IDAU [✓] Show Code [✓] Show Data [ ] Show Peripherals

	SAU+IDAU	MPC/PPC	Resulting security le...		
RAM 4 (alias)	S	NS-User	S	0x30043FFF 0x30040000	
RAM 3 (alias)	S	NS-User	S	0x3003FFFF 0x30030000	
RAM 2 (alias)	S	NS-User	S	0x3002FFFF 0x30020000	
RAM 1 (alias)	S	NS-User	S	0x3001FFFF 0x30010000	
RAM 0 (alias)	S	NS-User	S	0x3000FFFF 0x30008000	
	S	S-Priv	S	0x30007FFF 0x30000000	Secure Stack a...
RAM 4	NS	NS-User	NS	0x20043FFF 0x20040000	
RAM 3	NS	NS-User	NS	0x2003FFFF 0x20033000	
	NS	NS-User	NS	0x20032FFF 0x20030000	Non-secure St...
RAM 2	NS	NS-User	NS	0x2002FFFF 0x20020000	
RAM 1	NS	NS-User	NS	0x2001FFFF 0x20010000	
RAM 0	NS	NS-User	NS	0x2000FFFF 0x20008000	
	NS	S-Priv	NS	0x20007FFF 0x20000000	
SRAM X (alias)	S	NS-User	S	0x14007FFF 0x14000000	
Boot-ROM (alias)	S	NS-User	S	0x1301FFFF 0x13000000	
FLASH (alias)	S	NS-User	S	0x1009FFFF 0x10010000	
	NSC	S-Priv	NSC	0x1000FFFF 0x1000FE00	Veneer Table
	S	S-Priv	S	0x1000FDFF 0x10000000	Secure Code
SRAM X	NS	NS-User	NS	0x04007FFF 0x04000000	
Boot-ROM	NS	NS-User	NS	0x0301FFFF 0x03000000	
				0x0009FFFF	

```
/*************************************************************************
 * This file was generated by the MCUXpresso Config Tools. Any
 * will be overwritten if the respective MCUXpresso Config Too
 *************************************************************************

/* clang-format off */
/* TEXT BELOW IS USED AS SETTING FOR TOOLS ******************
!!GlobalInfo
product: TEE v1.0
processor: LPC55S69
package_id: LPC55S69JBD100
mcu_data: ksdk2_0
processor_version: 7.0.1
 * BE CAREFUL MODIFYING THIS COMMENT - IT IS YAML SETTINGS FOR
/* clang-format on */

#include "fsl_common.h"
#include "tzm_config.h"


/*************************************************************************
 * Definitions
 *************************************************************************
/* SAU region boundaries */
#define REGION_0_BASE 0
#define REGION_0_END 0x0FFFFFFFU
#define REGION_1_BASE 0x20000000U
#define REGION_1_END 0xFFFFFFFFU
#define REGION_2_BASE 0x1000FE00U
#define REGION_2_END 0x1000FFFFU

/* clang-format off */
/* TEXT BELOW IS USED AS SETTING FOR TOOLS ******************
tee:
- ahb:
  - regions:
    - relative_region: {memory: PROGRAM_FLASH, security: s_pri
    - relative_region: {memory: PROGRAM_FLASH, security: ns_us
    - relative_region: {memory: BootROM, security: ns_user, st
    - relative_region: {memory: SRAMX, security: ns_user, star
    - relative_region: {memory: SRAM0, security: s_priv, start
    - relative_region: {memory: SRAM0, security: ns_user, star
```

Level	Issue	Origin	Target	Re
Warnir	Memory region overla...	TEE		S/
Inforr	Ending address is not ...	TEE		U:

Figure 2. GUI interface of TEE tool

For details on using TEE tool, see *User Guide for MCUXpresso Config Tools (Desktop)* (document GSMCUXCTUG).

3.2 Secure/Non-secure state switch

After configuring the security environment of LPC55S69, users can use some special functions in the actual project to switch between secure and non-secure states. Here are two special functions: Non-secure Callable (NSC function/Entry function) and Non-secure function.

- NSC function

  NSC functions are the secure functions that can be called by non-secure functions. NSC function needs to be defined with the `__attribute__((cmse_nonsecure_entry))` attribute. The example is as below.

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

- Non-secure function

  Non-secure functions are the functions that can be called by secure functions. Non-secure function needs to be defined with the `__attribute_((cmse_nonsecure_call))` attribute. The example is as below.

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

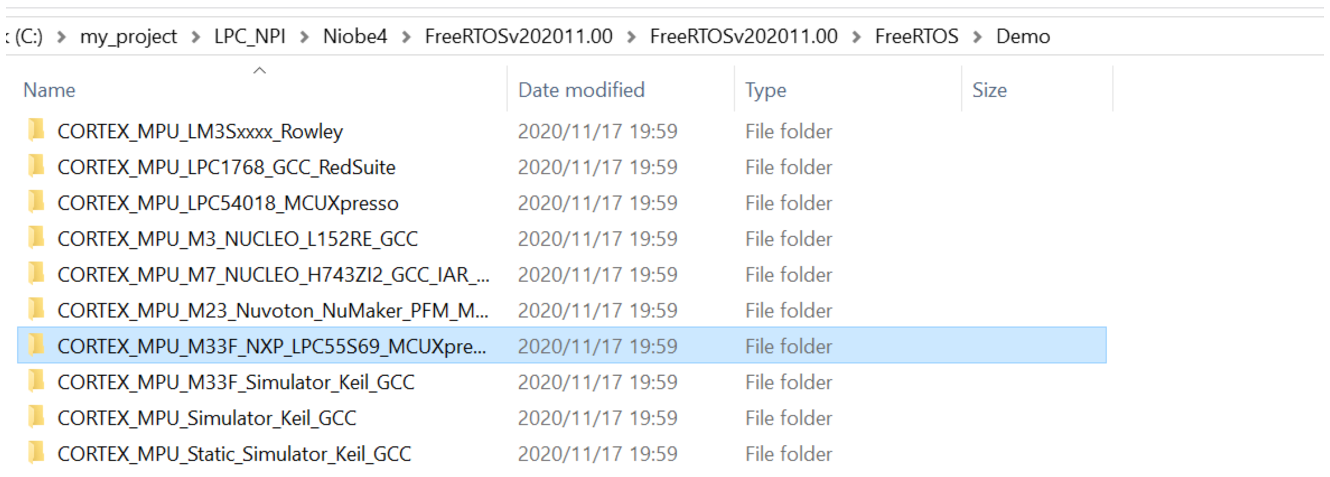
NOTE

Non-secure functions must be called by the way of function pointers.

For more details about secure/non-secure state switching, see documents on TrustZone technology for ARMv8-M Architecture.

4 FreeRTOS usage with TrustZone

This section describes how to run FreeRTOS in LPC55S69 with TrustZone enabled. Officially FreeRTOS provides FreeRTOS examples that support TrustZone to run on LPC55S69. FreeRTOS can be downloaded from Free RTOS. The path of LPC55S69 example is *FreeRTOS/Demo/CORTEX_MPU_M33F_NXP_LPC55S69_MCUXpresso*, as shown in Figure 3.



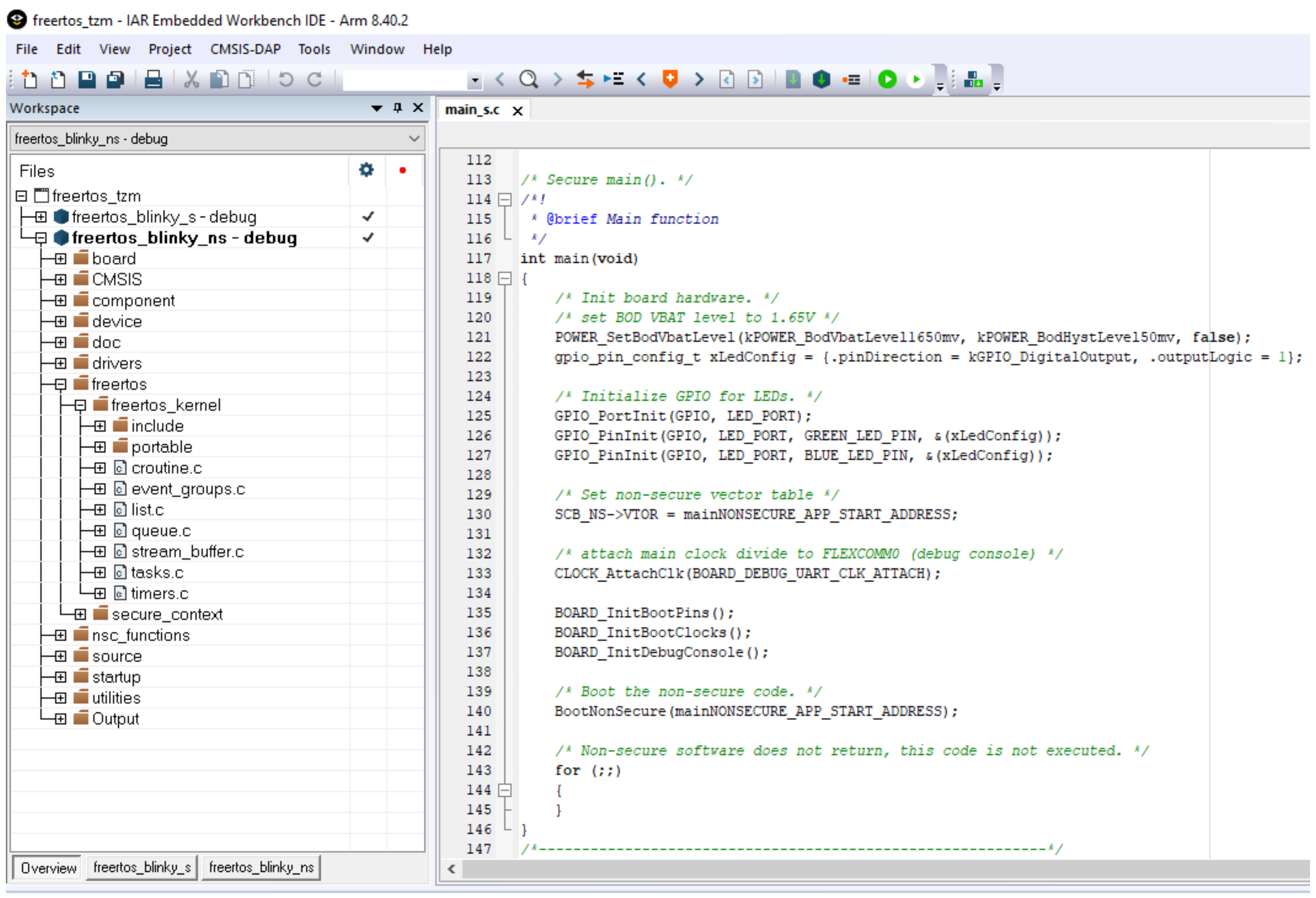
: (C:) > my_project > LPC_NPI > Niobe4 > FreeRTOSv202011.00 > FreeRTOSv202011.00 > FreeRTOS > Demo

Name	Date modified	Type	Size
CORTEX_MPU_LM3Sxxxx_Rowley	2020/11/17 19:59	File folder	
CORTEX_MPU_LPC1768_GCC_RedSuite	2020/11/17 19:59	File folder	
CORTEX_MPU_LPC54018_MCUXpresso	2020/11/17 19:59	File folder	
CORTEX_MPU_M3_NUCLEO_L152RE_GCC	2020/11/17 19:59	File folder	
CORTEX_MPU_M7_NUCLEO_H743ZI2_GCC_IAR_...	2020/11/17 19:59	File folder	
CORTEX_MPU_M23_Nuvoton_NuMaker_PFM_M...	2020/11/17 19:59	File folder	
CORTEX_MPU_M33F_NXP_LPC55S69_MCUXpre...	2020/11/17 19:59	File folder	
CORTEX_MPU_M33F_Simulator_Keil_GCC	2020/11/17 19:59	File folder	
CORTEX_MPU_Simulator_Keil_GCC	2020/11/17 19:59	File folder	
CORTEX_MPU_Static_Simulator_Keil_GCC	2020/11/17 19:59	File folder	

Figure 3. LPC55S69 FreeRTOS example

FreeRTOS provide documents about this demo. The documents can be downloaded from RTOS.

NXP's LPC55S69 SDK also provides a FreeRTOS example that supports TrustZone. The project name is *freertos_tzm* and the path of the example is */SDK_2.8.2_LPCXpresso55S69_IAR/boards/lpcxpresso55s69/rtos_examples/freertos_tzm* . The *freertos_tzm* project is as shown in Figure 4.



```

/* Secure main(). */
/*!
 * @brief Main function
 */
int main(void)
{
    /* Init board hardware. */
    /* set BOD VBAT level to 1.65V */
    POWER_SetBodVbatLevel(kPOWER_BodVbatLevel1650mv, kPOWER_BodHystLevel50mv, false);
    gpio_pin_config_t xLedConfig = {.pinDirection = kGPIO_DigitalOutput, .outputLogic = 1};

    /* Initialize GPIO for LEDs. */
    GPIO_PortInit(GPIO, LED_PORT);
    GPIO_PinInit(GPIO, LED_PORT, GREEN_LED_PIN, &(xLedConfig));
    GPIO_PinInit(GPIO, LED_PORT, BLUE_LED_PIN, &(xLedConfig));

    /* Set non-secure vector table */
    SCB_NS->VTOR = mainNONSECURE_APP_START_ADDRESS;

    /* attach main clock divide to FLEXCOMM0 (debug console) */
    CLOCK_AttachClk(BOARD_DEBUG_UART_CLK_ATTACH);

    BOARD_InitBootPins();
    BOARD_InitBootClocks();
    BOARD_InitDebugConsole();

    /* Boot the non-secure code. */
    BootNonSecure(mainNONSECURE_APP_START_ADDRESS);

    /* Non-secure software does not return, this code is not executed. */
    for (;;)
    {
    }
}
/*-----------------------------------------------------------*/
```

Figure 4. LPC55S69 freertos_tzm example in SDK v2.8

4.1 FreeRTOS example with TrustZone in SDK v2.8

This section takes the *freertos_tzm* project in SDK v2.8 as an example to introduce the workflow of FreeRTOS supporting TrustZone. The workflow of *freertos_tzm* example is as shown in Figure 5.

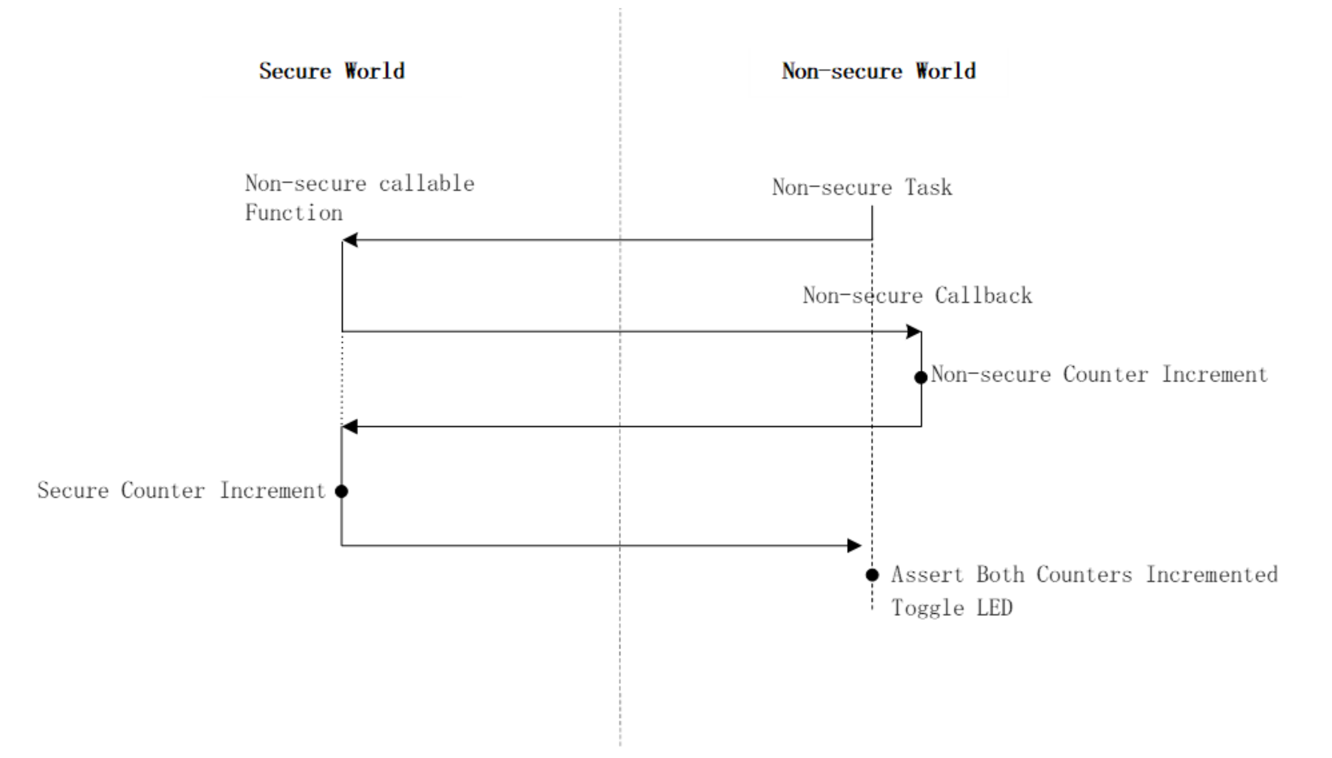


Figure 5. Workflow of the freertos_tzm demo

The memory partition of this project is as shown in Figure 6.

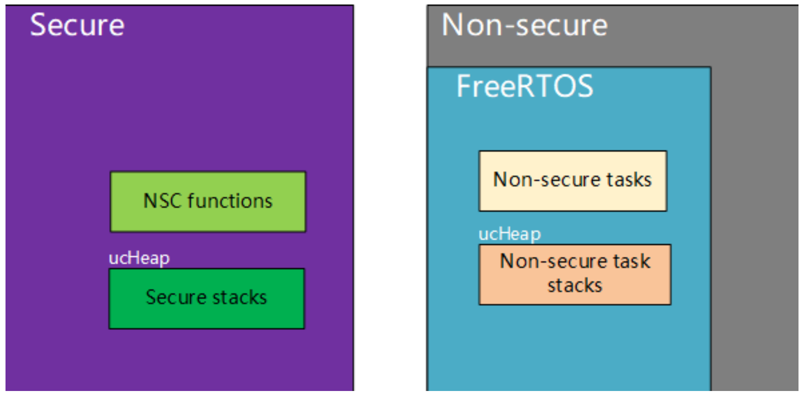
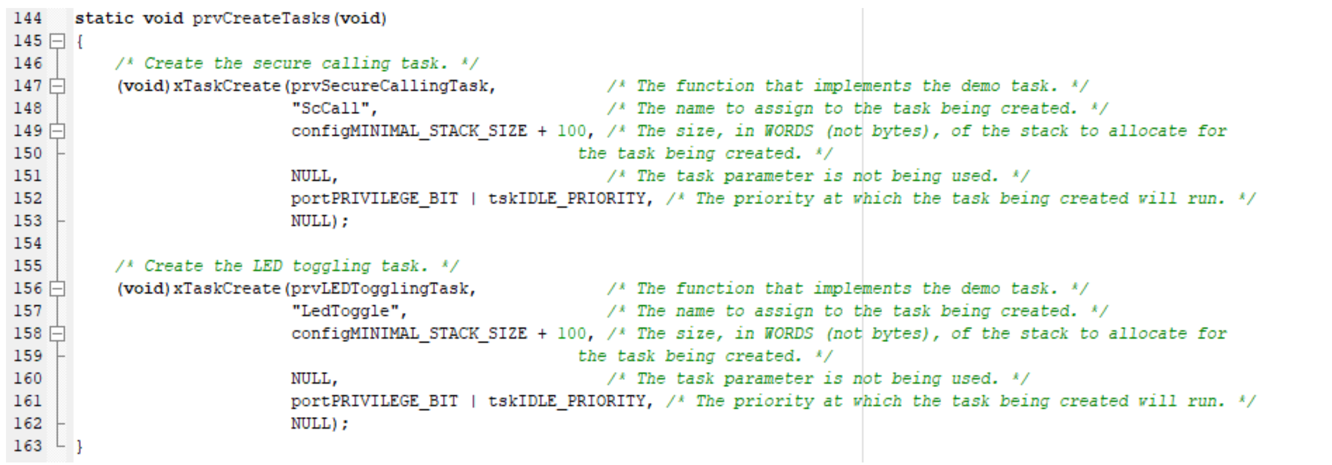


Figure 6. Memory partition in freertos_tzm example

Users can store some important resources in a secure region. FreeRTOS kernel is stored in a non-secure region, and the task creation and scheduling process is also completed in the non-secure region. These non-secure tasks can access some specific resources in the secure region by calling NSC functions, and secure functions can call the non-secure function to jump to the non-secure region.

4.1.1 Create user task

In the *freertos_tzm* example, two user tasks are created, as shown in Figure 7.



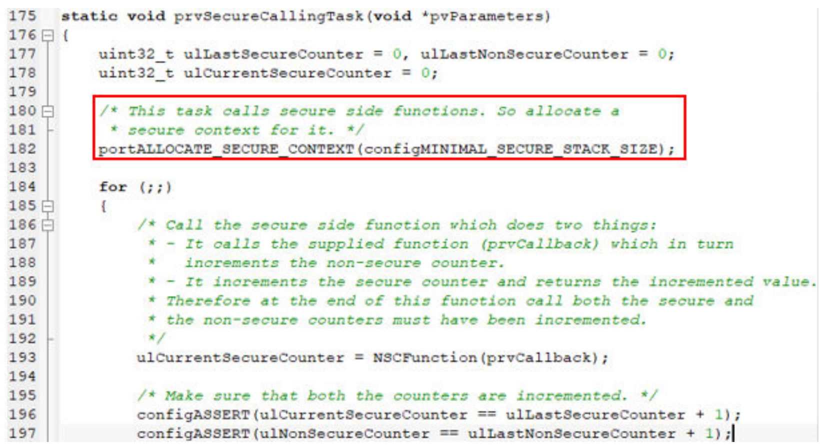
```
static void prvCreateTasks(void)
{
    /* Create the secure calling task. */
    (void)xTaskCreate(prvSecureCallingTask,          /* The function that implements the demo task. */
                      "ScCall",                      /* The name to assign to the task being created. */
                      configMINIMAL_STACK_SIZE + 100, /* The size, in WORDS (not bytes), of the stack to allocate for
                                                        the task being created. */
                      NULL,                          /* The task parameter is not being used. */
                      portPRIVILEGE_BIT | tskIDLE_PRIORITY, /* The priority at which the task being created will run. */
                      NULL);

    /* Create the LED toggling task. */
    (void)xTaskCreate(prvLEDTogglingTask,            /* The function that implements the demo task. */
                      "LedToggle",                   /* The name to assign to the task being created. */
                      configMINIMAL_STACK_SIZE + 100, /* The size, in WORDS (not bytes), of the stack to allocate for
                                                        the task being created. */
                      NULL,                          /* The task parameter is not being used. */
                      portPRIVILEGE_BIT | tskIDLE_PRIORITY, /* The priority at which the task being created will run. */
                      NULL);
}
```

Figure 7. Create user tasks in non-secure world

When the user using the `xTaskCreate()` function to create a task in the non-secure region, a non-secure task stack will be allocated. When the task is switched, the non-secure context can be stored in the non-secure task stack.

4.1.1.1 Allocate secure stack

Since the system memory is divided into a secure region and a non-secure region by TrustZone technology, it is possible to call NSC functions when performing non-secure tasks. Therefore, it is also necessary to allocate a secure stack for the non-secure task that call NSC functions to store the secure context. The allocation of the secure stack is completed when the non-secure task is executed for the first time, as shown in Figure 8.



```
static void prvSecureCallingTask(void *pvParameters)
{
    uint32_t ulLastSecureCounter = 0, ulLastNonSecureCounter = 0;
    uint32_t ulCurrentSecureCounter = 0;

    /* This task calls secure side functions. So allocate a
     * secure context for it. */
    portALLOCATE_SECURE_CONTEXT(configMINIMAL_SECURE_STACK_SIZE);

    for (;;)
    {
        /* Call the secure side function which does two things:
         * - It calls the supplied function (prvCallback) which in turn
         *   increments the non-secure counter.
         * - It increments the secure counter and returns the incremented value.
         * Therefore at the end of this function call both the secure and
         * the non-secure counters must have been incremented.
         */
        ulCurrentSecureCounter = NSCFunction(prvCallback);

        /* Make sure that both the counters are incremented. */
        configASSERT(ulCurrentSecureCounter == ulLastSecureCounter + 1);
        configASSERT(ulNonSecureCounter == ulLastNonSecureCounter + 1);
```

Figure 8. Allocate secure stack

When calling the `portALLOCATE_SECURE_CONTEXT()` function, an SVC interrupt is triggered, the `SecureContext_AllocateContext()` function (NSC function) is called to allocate a secure space in the secure ucHeap array as the secure stack for this non-secure task, and `PSP_S` and `PSPLIM_S` is initialized.

4.1.1.2 Define NSC function

After creating the task, users need to define the task function. If the non-secure task need to call the NSC function, they also need to define the corresponding NSC function. The NSC function is defined in the `nsc_functions.c` file of the secure project. The NSC function required by the `freertos_tzm` project is as shown in Table 1.

Table 1. NSC functions

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

4.1.2 Task switching

For MCUs supporting TrustZone, the context switch in FreeRTOS is different from the MCUs based on cortex-M3, M4, and M7 cores. This section introduces the task switching process of FreeRTOS supporting TrustZone. As it involves mutual call between secure functions and non-secure functions, the CPU uses some banked core registers in the secure and non-secure region, such as PSP, MSP, PSPLIM, MSPLIM, CONTROL registers, etc.. It makes the FreeRTOS task switching process more complex.

4.1.2.1 PendSV interrupt service process

The task switching process of FreeRTOS is completed in the PendSV interrupt service function. The task switching process of FreeRTOS supporting TrustZone is as shown in Figure 9.

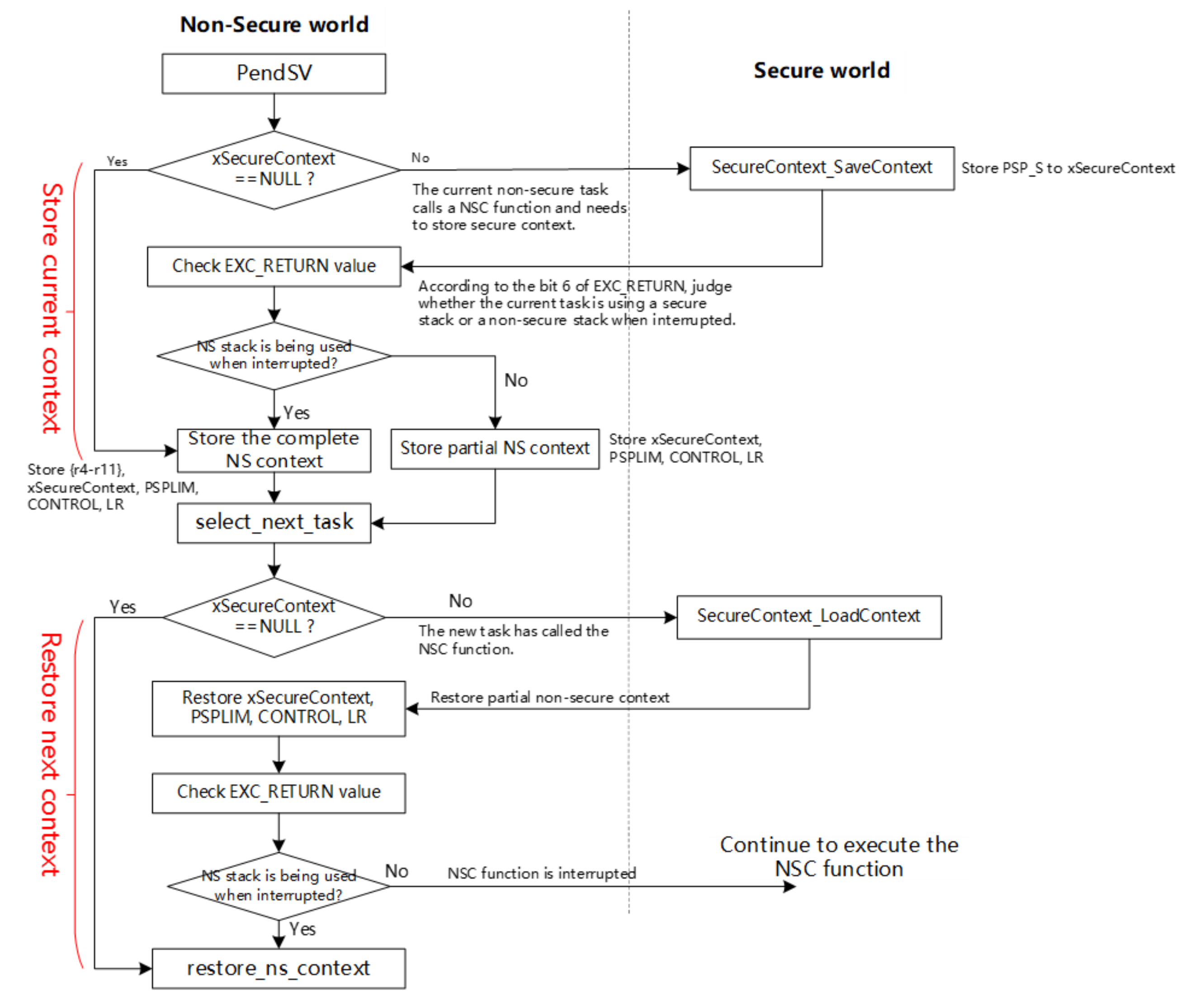

Non-Secure world
Secure world
PendSV
xSecureContext ==NULL ?
Yes
No
SecureContext_SaveContext
Store PSP_S to xSecureContext
The current non-secure task calls a NSC function and needs to store secure context.
Check EXC_RETURN value
According to the bit 6 of EXC_RETURN, judge whether the current task is using a secure stack or a non-secure stack when interrupted.
NS stack is being used when interrupted?
No
Yes
Store the complete NS context
Store partial NS context
Store xSecureContext, PSPLIM, CONTROL, LR
Store {r4-r11}, xSecureContext, PSPLIM, CONTROL, LR
select_next_task
Store current context
xSecureContext ==NULL ?
Yes
No
SecureContext_LoadContext
The new task has called the NSC function.
Restore xSecureContext, PSPLIM, CONTROL, LR
Restore partial non-secure context
Check EXC_RETURN value
NS stack is being used when interrupted?
No
NSC function is interrupted
Continue to execute the NSC function
Yes
restore_ns_context
Restore next context

Figure 9. Task switching process of FreeRTOS supporting TrustZone

NOTE

The complete NS context in Figure 9 contains 12 values: {R4-R11}, xSecureContext, PSPLIM, CONTROL, LR. The partial NS context contains only four values: xSecureContext, PSPLIM, CONTROL, LR.

4.1.2.2 Store the context of the current task

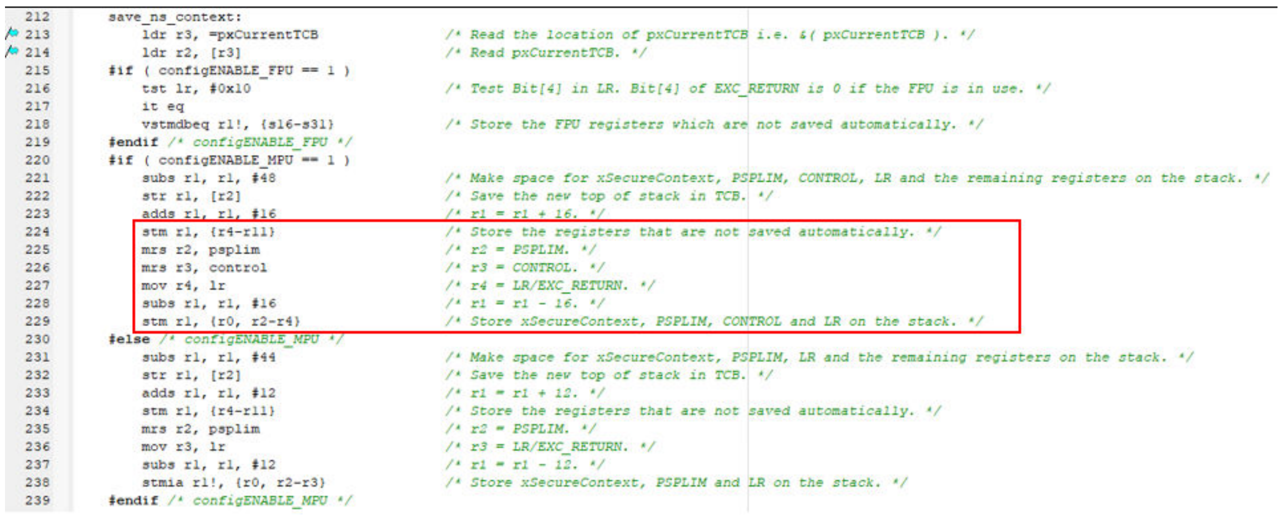
When performing task switching, users need to store the current task context first. The MCU based on CM33 core may call NSC functions when performing non-secure tasks. Therefore, when storing the current task context, they need to judge whether the current task calls NSC function firstly, that is, whether there is a secure context related to the current task (according to the value of the variable xSecureContext to determine whether the NSC function is called, if the variable value is not 0, it means that the secure function is called):

- If there is a secure context, the program must first jump to the secure region to store the secure context. Store the value of the `PSP_S` and `PSPLIM_S` to the corresponding secure stack.
  - — After storing the secure context, it is necessary to judge whether the NSC function is interrupted or the non-secure function is interrupted during the task switching. According to the value of bit 6 of `EXC_RETURN/LR`, if bit 6 is 1, it means the execution of the NSC function is interrupted; and if bit 6 is 0, it means that the non-secure function

is interrupted). If the NSC function is interrupted, because {r4 -r11} registers have been stored to the non-secure stack before calling NSC function and the secure context has also been stored to the secure stack, only part of the non-secure context (xSecureContext , PSPLIM, CONTROL, LR) needs to be stored.

— If the non-secure function is interrupted, users need to store the complete non-secure context including {r4-r11}, xSecureContext, PSPLIM, CONTROL, LR.

- If the current task does not call the NSC function, users only need to store the complete non-secure context, as shown in Figure 10.



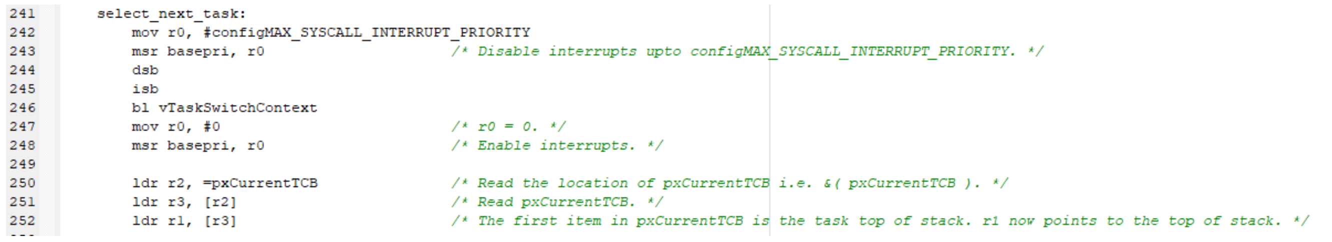
```
    save_ns_context:
        ldr r3, =pxCurrentTCB                  /* Read the location of pxCurrentTCB i.e. &( pxCurrentTCB ). */
        ldr r2, [r3]                           /* Read pxCurrentTCB. */
    #if ( configENABLE_FPU == 1 )
        tst lr, #0x10                          /* Test Bit[4] in LR. Bit[4] of EXC_RETURN is 0 if the FPU is in use. */
        it eq
        vstmdbeq r1!, {s16-s31}                /* Store the FPU registers which are not saved automatically. */
    #endif /* configENABLE_FPU */
    #if ( configENABLE_MPU == 1 )
        subs r1, r1, #48                       /* Make space for xSecureContext, PSPLIM, CONTROL, LR and the remaining registers on the stack. */
        str r1, [r2]                           /* Save the new top of stack in TCB. */
        adds r1, r1, #16                       /* r1 = r1 + 16. */
        stm r1, {r4-r11}                       /* Store the registers that are not saved automatically. */
        mrs r2, psplim                         /* r2 = PSPLIM. */
        mrs r3, control                        /* r3 = CONTROL. */
        mov r4, lr                             /* r4 = LR/EXC_RETURN. */
        subs r1, r1, #16                       /* r1 = r1 - 16. */
        stm r1, {r0, r2-r4}                    /* Store xSecureContext, PSPLIM, CONTROL and LR on the stack. */
    #else /* configENABLE_MPU */
        subs r1, r1, #44                       /* Make space for xSecureContext, PSPLIM, LR and the remaining registers on the stack. */
        str r1, [r2]                           /* Save the new top of stack in TCB. */
        adds r1, r1, #12                       /* r1 = r1 + 12. */
        stm r1, {r4-r11}                       /* Store the registers that are not saved automatically. */
        mrs r2, psplim                         /* r2 = PSPLIM. */
        mov r3, lr                             /* r3 = LR/EXC_RETURN. */
        subs r1, r1, #12                       /* r1 = r1 - 12. */
        stmia r1!, {r0, r2-r3}                 /* Store xSecureContext, PSPLIM and LR on the stack. */
    #endif /* configENABLE_MPU */
```

Figure 10. Store completed non-secure context

NOTE
The `freertos_tzm` project enables MPU by default.

4.1.2.3 Select the next ready task

After storing the context of the current task, the program will jump to the *select_next_task* stage to search for the task with the highest priority in the task ready table, as shown in Figure 11.



```
    select_next_task:
        mov r0, #configMAX_SYSCALL_INTERRUPT_PRIORITY
        msr basepri, r0                    /* Disable interrupts upto configMAX_SYSCALL_INTERRUPT_PRIORITY. */
        dsb
        isb
        bl vTaskSwitchContext
        mov r0, #0                         /* r0 = 0. */
        msr basepri, r0                    /* Enable interrupts. */

        ldr r2, =pxCurrentTCB              /* Read the location of pxCurrentTCB i.e. &( pxCurrentTCB ). */
        ldr r3, [r2]                       /* Read pxCurrentTCB. */
        ldr r1, [r3]                       /* The first item in pxCurrentTCB is the task top of stack. r1 now points to the top of stack. */
```

Figure 11. Select next task

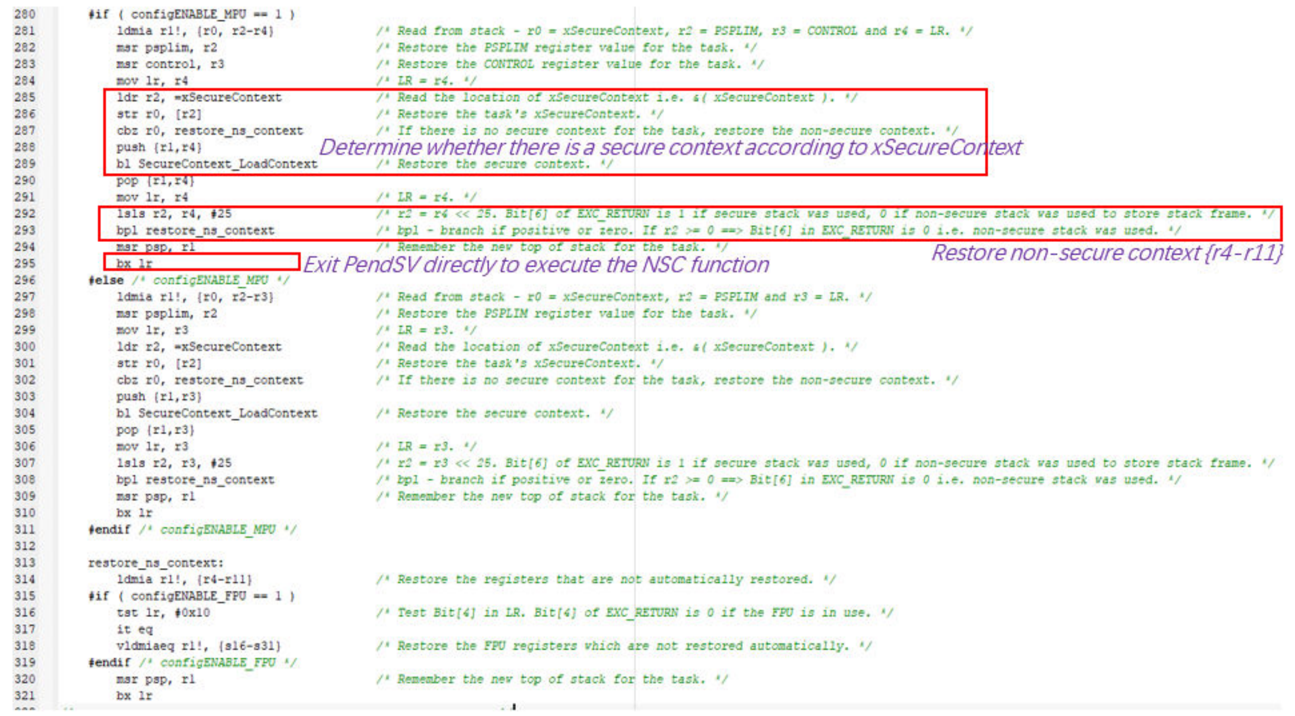
4.1.2.4 Restore the context of the next task

The process of context restoration and storation is similar. It is necessary to restore some non-secure context (xSecureContext, PSPLIM, CONTROL, LR) first, and then judge whether the next task calls the NSC function according to the value of the variable xSecureContext, that is, whether there is a secure context.

- If there is a secure context,
    1. Restore the secure context.

2. Restore the value of `PSP_S` and `PSPLIM_S`.
3. Return to the non-secure region to restore some non-secure context, xSecureContext, PSPLIM, CONTROL, LR.
4. Juddge whether the execution of the NSC function was interrupted or the execution of the non-secure function was interrupted according to the value of `EXC_RETURN`.
   — If the execution of the non-secure function is interrupted, users need to continue to restore the value of the r4 – r11 registers.
   — If the execution of the NSC function is interrupted, just exit PendSV interrupt, because the secure context has been restored.

- If there is no secure context, users only need to restore the non-secure context.



```
#if ( configENABLE_MPU == 1 )
    ldmia r1!, {r0, r2-r4}          /* Read from stack - r0 = xSecureContext, r2 = PSPLIM, r3 = CONTROL and r4 = LR. */
    msr psplim, r2                  /* Restore the PSPLIM register value for the task. */
    msr control, r3                 /* Restore the CONTROL register value for the task. */
    mov lr, r4                      /* LR = r4. */
    ldr r2, =xSecureContext         /* Read the location of xSecureContext i.e. &( xSecureContext ). */
    str r0, [r2]                    /* Restore the task's xSecureContext. */
    cbz r0, restore_ns_context      /* If there is no secure context for the task, restore the non-secure context. */
    push {r1,r4}
    bl SecureContext_LoadContext    /* Restore the secure context. */
    pop {r1,r4}
    mov lr, r4                      /* LR = r4. */
    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. */
    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. */
    msr psp, r1                     /* Remember the new top of stack for the task. */
    bx lr
#else /* configENABLE_MPU */
    ldmia r1!, {r0, r2-r3}          /* Read from stack - r0 = xSecureContext, r2 = PSPLIM and r3 = LR. */
    msr psplim, r2                  /* Restore the PSPLIM register value for the task. */
    mov lr, r3                      /* LR = r3. */
    ldr r2, =xSecureContext         /* Read the location of xSecureContext i.e. &( xSecureContext ). */
    str r0, [r2]                    /* Restore the task's xSecureContext. */
    cbz r0, restore_ns_context      /* If there is no secure context for the task, restore the non-secure context. */
    push {r1,r3}
    bl SecureContext_LoadContext    /* Restore the secure context. */
    pop {r1,r3}
    mov lr, r3                      /* LR = r3. */
    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. */
    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. */
    msr psp, r1                     /* Remember the new top of stack for the task. */
    bx lr
#endif /* configENABLE_MPU */

restore_ns_context:
    ldmia r1!, {r4-r11}             /* Restore the registers that are not automatically restored. */
#if ( configENABLE_FPU == 1 )
    tst lr, #0x10                   /* Test Bit[4] in LR. Bit[4] of EXC_RETURN is 0 if the FPU is in use. */
    it eq
    vldmiaeq r1!, {s16-s31}         /* Restore the FPU registers which are not restored automatically. */
#endif /* configENABLE_FPU */
    msr psp, r1                     /* Remember the new top of stack for the task. */
    bx lr
```

Determine whether there is a secure context according to xSecureContext
Restore non-secure context {r4-r11}
Exit PendSV directly to execute the NSC function

Figure 12. Restore next task context

4.1.3 Run freertos_tzm example

In this example, two user tasks are created.

- `prvSecureCallingTask`

  This task calls the NSC function to increase ulSecureCounter by 1 and then call the non-secure callback to increase ulNonSecureCounter by 1. Assert whether `ulSecureCounter` and `ulNonSecureCounter` are both increased by 1, and then toggle the Green LED once. The workflow is as shown in Figure 5. This task is executed once every second.

- `prvLEDTogglingTask`

  Toggle Blue LED once. This task is executed once every second.

After running the program, we can find that the Green LED and Blue LED flash alternately, and the flashing period is 1 second.

4.2 A safer way to use FreeRTOS

In some cases, placing the FreeRTOS kernel and user tasks in the non-secure region is not enough to meet the security requirements of the system. At this time, we can consider placing the FreeRTOS kernel and some important tasks in the secure region and others in the non-secure region, thus ensuring that the operating system is protected from non-secure attacks and improving the security of the system.

This section introduces a new method to support TrustZone in FreeRTOS: place the FreeRTOS kernel and important tasks in the secure region and other user tasks in the non-secure region, as shown in Figure 13.

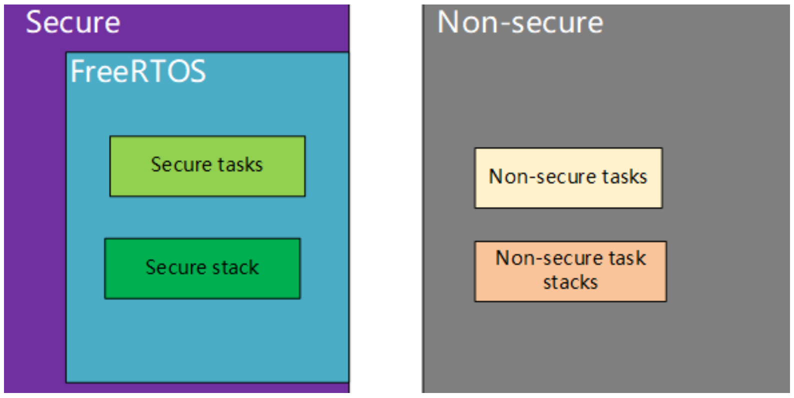


Figure 13. A safer way to use FreeRTOS

For the details, see Method to support TrustZone-M in FreeRTOS. Users can refer to this method to make their system more secure.

5 Reference

- *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

Date of release: January 15, 2020

Document identifier: AN13094