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](image)

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.
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.

  ```c
  __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.

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

---

Figure 2. GUI interface of TEE tool

[Diagram of GUI interface of TEE tool]

For details on using TEE tool, see User Guide for MCUXpresso Config Tools (Desktop) (document GSMCUXCTUG).
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 FreeRTOS. The path of LPC55S69 example is FreeRTOS/Demo/CORTEX_MPU_M33F_NXP_LPC55S69_MCUXpresso, as shown in Figure 3.

![Figure 3. LPC55S69 FreeRTOS example](image)

FreeRTOS provides 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.
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.

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

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.
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.

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

<table>
<thead>
<tr>
<th>NSC function</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td><code>uint32_t NSCFunction(Callback_t pxCallback)</code></td>
</tr>
<tr>
<td></td>
<td><code>void vToggleGreenLED(void)</code></td>
</tr>
<tr>
<td></td>
<td><code>void vToggleBlueLED(void)</code></td>
</tr>
<tr>
<td></td>
<td><code>uint32_t getSystemCoreClock(void)</code></td>
</tr>
</tbody>
</table>

#### 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.
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 current non-secure task calls a NSC function and needs to store secure context:
  - 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.

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.
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.

![Figure 10. Store completed non-secure context](image)

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.

![Figure 11. Select next task](image)

4.1.2.4 Restore the context of the next task

The process of context restoration and storage 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. Judge 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.

![Figure 12. Restore next task context](image)

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.

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
Information in this document is provided solely to enable system and software implementers to use NXP products. There are no express or implied copyright licenses granted hereunder to design or fabricate any integrated circuits based on the information in this document. NXP reserves the right to make changes without further notice to any products herein.

NXP makes no warranty, representation, or guarantee regarding the suitability of its products for any particular purpose, nor does NXP assume any liability arising out of the application or use of any product or circuit, and specifically disclaims any and all liability, including without limitation consequential or incidental damages. “Typical” parameters that may be provided in NXP data sheets and/or specifications can and do vary in different applications, and actual performance may vary over time. All operating parameters, including “typicals,” must be validated for each customer application by customer's technical experts. NXP does not convey any license under its patent rights nor the rights of others. NXP sells products pursuant to standard terms and conditions of sale, which can be found at the following address: nxp.com/SalesTermsandConditions.

Right to make changes - NXP Semiconductors reserves the right to make changes to information published in this document, including without limitation specifications and product descriptions, at any time and without notice. This document supersedes and replaces all information supplied prior to the publication hereof.

Security — Customer understands that all NXP products may be subject to unidentified or documented vulnerabilities. Customer is responsible for the design and operation of its applications and products throughout their lifecycles to reduce the effect of these vulnerabilities on customer’s applications and products. Customer’s responsibility also extends to other open and/or proprietary technologies supported by NXP products for use in customer’s applications. NXP accepts no liability for any vulnerability. Customer should regularly check security updates from NXP and follow up appropriately. Customer shall select products with security features that best meet rules, regulations, and standards of the intended application and make the ultimate design decisions regarding its products and is solely responsible for compliance with all legal, regulatory, and security related requirements concerning its products, regardless of any information or support that may be provided by NXP. NXP has a Product Security Incident Response Team (PSIRT) (reachable at PSIRT@nxp.com) that manages the investigation, reporting, and solution release to security vulnerabilities of NXP products.

NXP, the NXP logo, NXP SECURE CONNECTIONS FOR A SMARTER WORLD, COOLFLUX, EMBRACE, GREENCHIP, HITAG, ICODE, JCOP, LIFE, VIBES, MIFARE, MIFARE CLASSIC, MIFARE DESFire, MIFARE PLUS, MIFARE FLEX, MANTIS, MIFARE ULTRALIGHT, MIFARE4MOBILE, MIGLO, NTAG, ROADLINK, SMARTFLX, SMARTMX, STARPLUG, TOPFET, TRENCHMOS, UCODE, Freescale, the Freescale logo, Atmel, CodeWarrior, ColdFire, ColdFire+, the Energy Efficient Solutions logo, Kinetis, Layerscape, MagniV, mobileGT, PEG, PowerQUICC, Processor Expert, QorIQ, QorIQ Converge, SafeAssure, the SafeAssure logo, StarCore, Symphony, VortiQa, Vybrid, Airfast, BeeKit, BeeStack, CoreNet, Flexis, MXC, Platform in a Package, QUICC Engine, Tower, TurboLink, EdgeScale, EdgeLock, eIQ, and Immersive3D are trademarks of NXP B.V. All other product or service names are the property of their respective owners. AMBA, Arm, Arm7, ArmTDMI, Arm9, Arm11, Artisan, big.LITTLE, Cordio, CoreLink, CoreSight, Cortex, DesignStart, DynamIQ, Jazelle, Keil, Mali, Mbed Enabled, NEON, POP, RealView, SecurCore, Socrates, Thumb, TrustZone, ULINK, ULINK2, ULINK-ME, ULINK-PLUS, ULINKpro, µVision, Versatile are trademarks or registered trademarks of Arm Limited (or its subsidiaries) in the US and/or elsewhere. The related technology may be protected by any or all of patents, copyrights, designs and trade secrets. All rights reserved. Oracle and Java are registered trademarks of Oracle and/or its affiliates. The Power Architecture and Power.org word marks and the Power and Power.org logos and related marks are trademarks and service marks licensed by Power.org.