理解与应用 MPU 的特权与用户模式

前言

STM32 系列支持 MPU 内存保护单元，可用来设定内存的属性和访问权限。MPU 的应用笔记提到，将属性寄存器（MPU_RASR）配置成某一个值，在特权(Privileged permissions)和用户模式(Unprivileged permissions)的访问许可是不同的，甚至可将用户模式的权限设置成不可访问。那么，什么是 MPU 的特权模式和用户模式呢？ 接下来我们在这篇文章来理解这些名词，并讨论在 STM32 MCU 代码中如何使用内存保护单元 MPU 的特权与用户模式。

MPU(Memory Protection Unit)

MPU 内存保护单元可以用来使嵌入式系统更加健壮与安全。它可以阻止用户应用程序破坏系统关键数据。它可以通过将内存 SRAM 区域定义成不可执行，来阻止代码注入型攻击。也可以用来改变内存的性质，例如是否允许缓存(Cache)。

用来设置内存属性的 MPU_RASR 寄存器字段描述如下：

位	名字	描述
28	XN	从不执行
26 : 24	**AP**	**数据访问许可(RO, RW 或者无权限)**
21 : 19	TEX	类型扩展字段
18	S	共享
17	C	可缓存
16	B	可缓冲
15 :8	SRD	禁止子块
5 :1	SIZE	指定 MPU 保护区域的大小

图表 1 MPU_RASR 字段

在 MPU_RASR 中用来设置数据访问许可的 **AP** 字段详细设置选项如下：

AP[2 :0]	特权模式	用户模式	描述
000	不可访问	不可访问	所有的访问产生一个内存管理异常
001	RW	不可访问	仅可从特权模式访问
010	RW	RO	在用户模式下的写访问会产生内存管理异常
011	RW	RW	完全访问
100	不可预知	不可预知	保留
101	RO	不可访问	仅可在特权模式下进行读访问
110	RO	RO	特权和用户模式只读
111	RO	RO	特权和用户模式只读

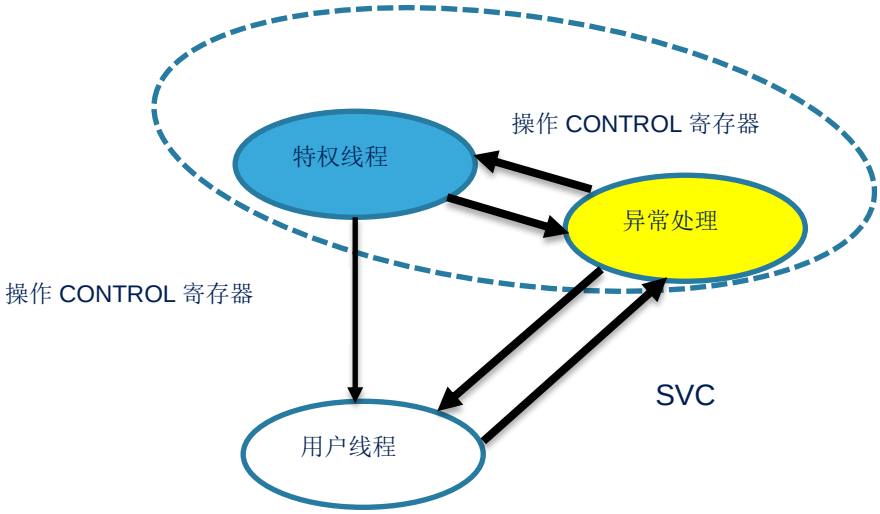
图表 2 内存访问权限设置

特权模式（Privileged mode）与用户模式(UnPrevileged mode)

特权模式与用户模式是指 Cortex 内核的执行模式。它不是 MPU 的一部分，但与 MPU 单元有关联。当代码运行在特权模式下，代码拥有所有的访问许可；而代码运行在用户模式，则访问权限受限制。限制包括在系统设计阶段就定义的可运行指令限制，可访问内存以及外设限制。也包括由 MPU 单元动态所定义的内存访问规则。

从特权模式进入用户模式，只需使用 MSR 指令操作 CONTROL 寄存器。但是，不可以直接从用户模式转入特权模式，必须经过一个异常处理操作模式，比如 SVC（ Supervisor Calls）。在异常处理通过操作 CONTROL 寄存器，可从用户模式回到特权模式。请注意，特权线程与异常处理线程，都是在特权模式下运行。

如下图所示：



图表 3 特权模式与用户模式的切换

在代码中结合特权与用户模式使用 MPU

1. 开发环境

开发板： STM32 L476RG NUCLEO

开发工具：STM32Cube_FW_L4_V1.7.0

IAR/Keil

注：也可以选择其他 STM32 系列并选择相应的开发板与固件库。

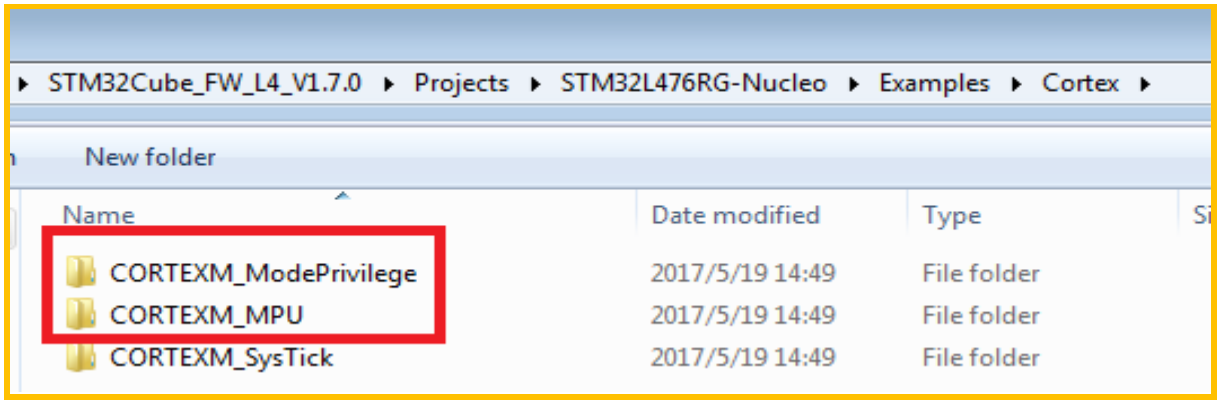
2. 开发目标定义

在软件中定义一块数组，使用 MPU 将该区域配成仅可从特权模式下进行访问，并验证用户模式下访问会导致内存管理异常或者硬错误。

AP[2 :0]	特权模式	用户模式	描述
001	RW	不可访问	仅可从特权模式访问

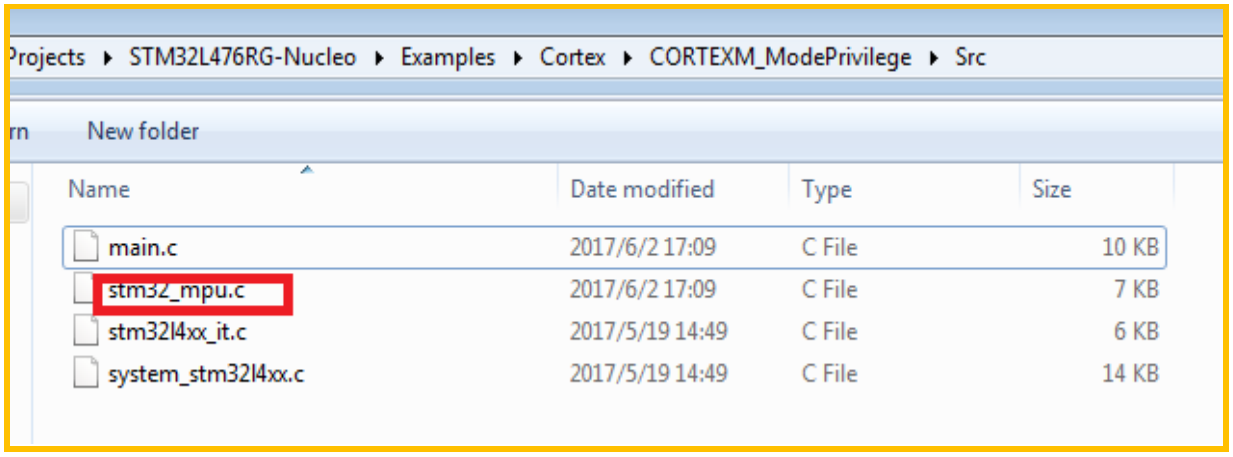
3. 代码资源与整合

STM32 Cube 固件库提供了大量的例程供学习和重用。查找 STM32Cube_FW_L4_V1.6.0 可以发现，固件库已分别实现了特权模式与用户模式的切换（**CORTEXM_ModePrivilege**），和 MPU 的配置（**CORTEXM_MPU**）。

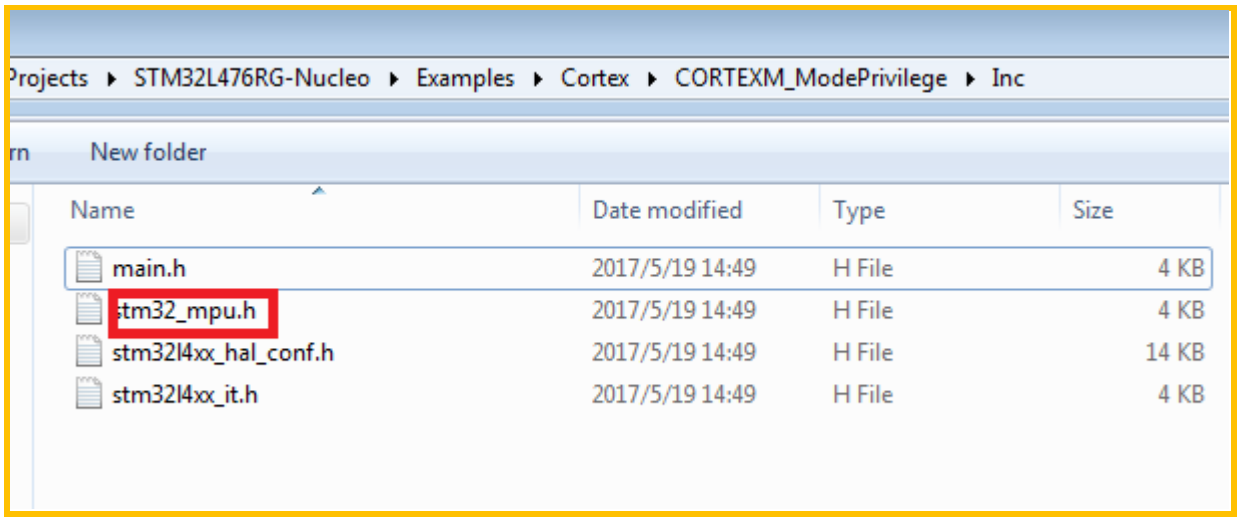


我们需要将两个例程整合在一起来实现我们的开发目标。注意你选择其他 STM32 系列也是没有问题的，也支持 MPU 功能。

复制 **CORTEXM_MPU\Src\stm32_mpu.c** 到 **CORTEXM_ModePrivilege\Src** 目录下



复制 **CORTEXM_MPU\Inc\stm32_mpu.h** 到 **CORTEXM_ModePrivilege\Inc** 目录下



打开 **CORTEXM_ModePrivilege** 工程文件，将 **stm32_mpu.c** 和 **stm32_mpu.h** 添加到工程文件中。IAR 与 Keil 略有不同，在 IAR 下只需添加 C 文件。

4. 修改与增加代码

新增加的代码以红色表示

- 在主程序中包含 stm32_mpu.h

```
/* Includes ------------------------------------------------------------------*/
#include "main.h"
#include "stm32_mpu.h" /*added*/

/** @addtogroup STM32L4xx_HAL_Examples
  * @{
  */

/** @addtogroup CORTEXM_ModePrivilege
  * @{
  */
```

- 在主程序里调用 MPU 配置函数

要对使用的内存进行 MPU 配置，未被 MPU 配置的资源将视之为不可访问。这是 MPU_Config 必须被使用的原因。

```
/* Get the Thread mode stack used */
  if((__get_CONTROL() & 0x02) == SP_MAIN)
  {
    /* Main stack is used as the current stack */
    CurrentStack = SP_MAIN;
  }
  else
  {
    /* Process stack is used as the current stack */
    CurrentStack = SP_PROCESS;

    /* Get process stack pointer value */
    PSPValue = __get_PSP();
  }
  MPU_Config(); /*added*/
  MPU_AccessPermConfig(); /*added*/
```

- 修改 MPU_AccessPermConfig 将数组区域配置成我们设计的权限并增加测试代码。

MPU_AccessPermConfig 在特权模式下运行，故测试代码不会触发异常。

```
void MPU_AccessPermConfig(void)
{
  MPU_Region_InitTypeDef MPU_InitStruct;

  /* Configure region for PrivilegedReadOnlyArray as REGION N?, 32byte and R
     only in privileged mode */
  /* Disable MPU */
  HAL_MPU_Disable();
```

```
  MPU_InitStruct.Enable = MPU_REGION_ENABLE;
  MPU_InitStruct.BaseAddress = ARRAY_ADDRESS_START;
  MPU_InitStruct.Size = ARRAY_SIZE;
  MPU_InitStruct.AccessPermission = portMPU_REGION_PRIVILEGED_READ_WRITE;
  MPU_InitStruct.IsBufferable = MPU_ACCESS_NOT_BUFFERABLE;
  MPU_InitStruct.IsCacheable = MPU_ACCESS_NOT_CACHEABLE;
  MPU_InitStruct.IsShareable = MPU_ACCESS_NOT_SHAREABLE;
  MPU_InitStruct.Number = ARRAY_REGION_NUMBER;
  MPU_InitStruct.TypeExtField = MPU_TEX_LEVEL0;
  MPU_InitStruct.SubRegionDisable = 0x00;
  MPU_InitStruct.DisableExec = MPU_INSTRUCTION_ACCESS_ENABLE;

  HAL_MPU_ConfigRegion(&MPU_InitStruct);

  /* Enable MPU (any access not covered by any enabled region will cause a fault) */
  HAL_MPU_Enable(MPU_HFNMI_PRIVDEF_NONE);

  /* Read from PrivilegedReadOnlyArray. This will not generate error */
  if(PrivilegedReadOnlyArray[0])
  {
    PrivilegedReadOnlyArray[0] = 'e';
  }

  /* Uncomment the following line to write to PrivilegedReadOnlyArray. This will
     generate error */
   PrivilegedReadOnlyArray[0] = 'e';
}
```

- 在用户模式下尝试读写数组

参考代码中直接访问数组，很容易被编译器优化掉而不能发挥测试作用。这里加了个判断后写的操作，使编译器不去优化它。

```
/* Switch Thread mode from privileged to unprivileged -------------------------*/
  /* Thread mode has unprivileged access */
  __set_CONTROL(THREAD_MODE_UNPRIVILEGED | SP_PROCESS);

  /* Execute ISB instruction to flush pipeline as recommended by Arm */
  __ISB();

  /* Unprivileged access mainly affect ability to:
      - Use or not use certain instructions such as MSR fields
      - Access System Control Space (SCS) registers such as NVIC and SysTick */

  /* Check Thread mode privilege status */
  if((__get_CONTROL() & 0x01) == THREAD_MODE_PRIVILEGED)
  {
    /* Thread mode has privileged access  */
    ThreadMode = THREAD_MODE_PRIVILEGED;
  }
  else
  {
```

```
    /* Thread mode has unprivileged access*/
    ThreadMode = THREAD_MODE_UNPRIVILEGED;
  }
  if(PrivilegedReadOnlyArray[0])
  {
    PrivilegedReadOnlyArray[0] = 'e';
  }
  PrivilegedReadOnlyArray[0] = 'e';
```

至此，可以编译成功运行来体会特权模式与用户模式的 MPU 的不同配置。执行到主程序中新加的数组访问代码，会产生内存管理异常。

5. 关键代码分析

- 从特权模式进入用户模式。直接设置 CONTROL 寄存器。

```
/* Switch Thread mode from privileged to unprivileged -------------------------*/
  /* Thread mode has unprivileged access */
  __set_CONTROL(THREAD_MODE_UNPRIVILEGED | SP_PROCESS);
```

- 从用户模式回到特权模式，需要触发异常处理

```
/* Generate a system call exception, and in the ISR switch back Thread mode
    to privileged */
  __SVC();
```

- 在异常处理中将线程回到特权模式执行

```
/**
  * @brief  This function handles SVCall exception.
  * @param  None
  * @retval None
  */
void SVC_Handler(void)
{
  /* Switch back Thread mode to privileged */
  __set_CONTROL(THREAD_MODE_PRIVILEGED | SP_PROCESS);

  /* Execute ISB instruction to flush pipeline as recommended by Arm */
  __ISB();
}
```

- 若用户模式下代码试图访问无权限数据将产生内存管理异常，从而进入下述函数。

```
/**
  * @brief  This function handles Memory Manage exception.
  * @param  None
  * @retval None
  */
void MemManage_Handler(void)
{
  /* Go to infinite loop when Memory Manage exception occurs */
  while (1)
```

```
  {
  }
}
```

结论

本文档介绍了 MPU 的特权与用户模式，并整合与修改已有代码进行了应用。在实际中可将 MPU 与 Cortex 运行模式结合，可使应用更加健壮与安全。

重要通知 – 请仔细阅读

意法半导体公司及其子公司（“ST”）保留随时对ST 产品和/ 或本文档进行变更、更正、增强、修改和改进的权利，恕不另行通知。买方在订货之前应获取关于ST 产品的最新信息。ST 产品的销售依照订单确认时的相关ST 销售条款。

买方自行负责对ST 产品的选择和使用， ST 概不承担与应用协助或买方产品设计相关的任何责任。

ST 不对任何知识产权进行任何明示或默示的授权或许可。

转售的ST 产品如有不同于此处提供的信息的规定，将导致ST 针对该产品授予的任何保证失效。

ST 和ST 徽标是ST 的商标。所有其他产品或服务名称均为其各自所有者的财产。

本文档中的信息取代本文档所有早期版本中提供的信息。