管理 STM32 MCU 中的内存保护单元

引言

本应用笔记介绍如何管理 STM32 产品中的内存保护单元（MPU）。MPU 是用于存储器保护的可选组件。STM32 微控制器（MCU）中嵌入 MPU 之后变得更稳健可靠。在使用 MPU 之前，必须对其进行编程并加以启用。如果 MPU 没有启用，则存储系统的行为不会变化。

该应用笔记涉及表 1 中列出的所有 STM32 产品，包括支持 MPU 的 Cortex®-M0+/M3/M4 和 M7 设计。

如需详细了解 MPU，请参阅 www.st.com 上提供的以下文档

- 编程手册 *STM32F7 系列和 STM32H7 系列 Cortex®-M7 处理器*（PM0253）
- 编程手册 *STM32F10xxx/20xxx/21xxx/L1xxxx Cortex®-M3*（PM0056）
- 编程手册 *Cortex®-M0+面向 STM32L0、STM32G0、STM32WL 和 STM32WB 系列*（PM0223）
- 编程手册 *STM32 Cortex®-M4 MCUs 和 MPU*（PM0214）
- 编程手册 *STM32 Cortex®-M33 MCU*（PM0264）

表 1. 适用产品

类型	产品系列
微控制器	• STM32F1 系列、STM32F2 系列、STM32F3 系列、STM32F4 系列、STM32F7 系列 • STM32G0 系列、STM32G4 系列 • STM32H7 系列 • STM32L0 系列，STM32L1 系列，STM32L4 系列，STM32L4+系列，STM32L5 系列 • STM32U5 系列 • STM32WB 系列

1 概述

该应用笔记适用于基于 Arm®的 STM32 微控制器器件。

提示 *Arm is a registered trademark of Arm Limited (or its subsidiaries) in the US and/or elsewhere.*

2 概述

MPU 可以使嵌入式系统更加稳健和安全：

- 禁止用户应用程序破坏关键任务（例如操作系统核心）使用的数据
- 将 SRAM 存储区域定义为非可执行（禁止执行 XN），以防止代码注入攻击
- 修改存储访问属性

MPU可最多保护16个内存区域。在 Armv6、Armv7 架构（Cortex-M0+、M3、M4、M7）下，这些区域可以依次拥有 8 个子区域（前提是区域至少有 256 字节）。在 STM32 中，受保护区域的确切数量可能因内核和器件而有所不同，请参阅 Cortex-M33 MPU 寄存器获取详细信息。子区域的大小都是相等的，可以根据子区域号进行启用或禁用。因为最小区域大小是由缓存行长度（32 字节）驱动的，所以 8 个 32 字节的子区域对应一个 256 字节的区域。

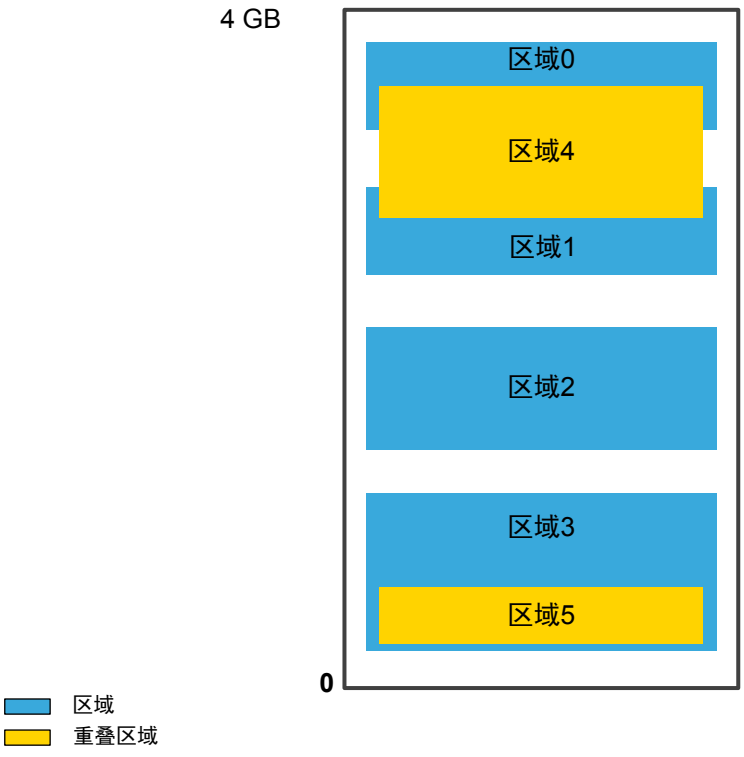
区域的编号为 0 至 15。此外，还有一处默认区域，其 id 为-1。所有编号 0-15 的存储区域的优先级高于默认区域。

这些区域可以重叠，也可以嵌套。区域 0-15 的优先级由低到高，这也决定了区域重叠的方式。优先级是固定的，不可更改。

在 Armv8 架构（Cortex-M33）中，使用起始地址和终止地址来定义区域，使开发人员能够以灵活、简单的方式组织这些区域。此外，正是区域大小的可灵活配置得到提升，故Cortex-M33就没有子区域的概念了。

下图显示的示例包含六个区域。该示例显示区域 4 与区域 0 和 1 重叠。区域 5 完全包含在区域 3 内。因为优先级是递增的，所以重叠区域（橙色）优先。因此，如果区域 0 是可写的，而区域 4 不可写，那么位于区域 0 和区域 4 重叠部分的地址为不可写。

图 1. 重叠区域示例



Caution: 在 Armv8 架构（Cortex-M33）中，现在不允许区域重叠。由于 MPU 区域的定义更加灵活，因此没有必要重叠 MPU 区域。

MPU 是统一的，意味着没有单独的区域用于数据和指令。

MPU 还可以用于定义其他存储器属性（如可缓存性），可以导出到系统级缓存单元或存储存控制器。Arm 架构中的存储器属性设置可以支持两种级别的缓存：内部缓存和外部缓存。STM32F7 和 STM32H7 系列仅支持一种级别的缓存（L1-缓存）。

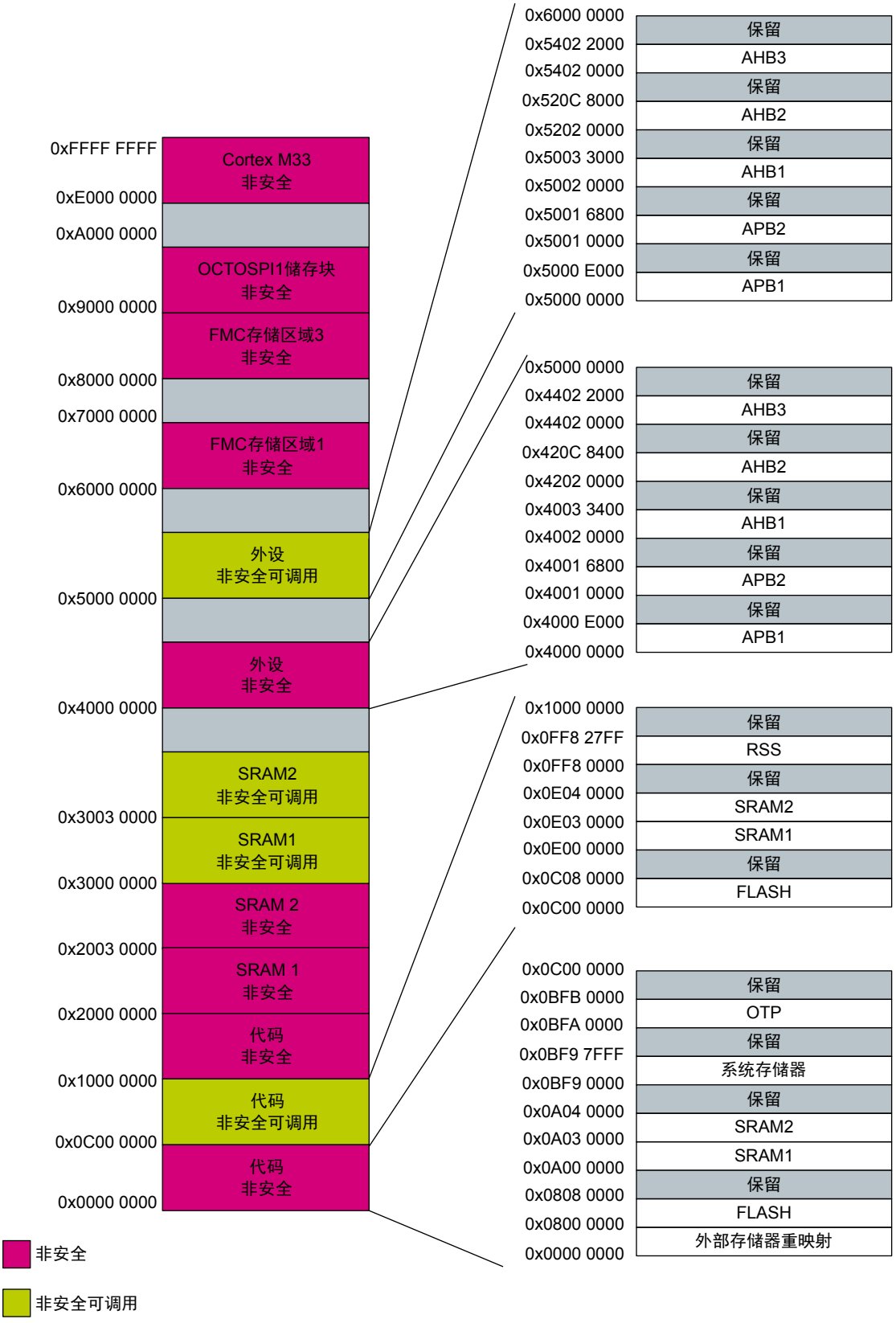
缓存控制由缓存控制寄存器实现全局控制，但 MPU 可以指定缓存策略以及区域是否可缓存。

2.1 存储器模型

在 STM32 产品中，处理器具有固定的默认存储器映射，可提供最多 4 Gb 的可寻址存储器。

图 2. Cortex-M0+/M3/M4/M7 处理器存储器映射

图 3. **Cortex-M33 处理器存储器映射**



3 Cortex-M0+/M3/M4/M7 存储器类型、寄存器和属性

存储器映射和 MPU 编程将存储器映射分为多个区域。每个区域都有已定义的存储器类型和存储器属性。存储器类型和属性决定该区域的访问行为。

3.1 存储器类型

有三种常见的存储器类型：

- 普通存储器：允许 CPU 以有效方式安排字节、半字和字的加载和存储（编译器不了解存储器区域类型）。对于普通存储器区域，CPU 不一定按照程序中列出的顺序执行加载/存储操作。
- 器件存储器：在器件区域内，负载和存储是严格按照顺序进行的。这是为了确保以正确的顺序设置寄存器。
- 强排序存储器：所有操作始终按以编程方式列出的顺序执行，CPU 会等待加载/存储指令执行（有效的总线访问）结束，然后执行程序流中的下一条指令。这可能导致性能损失。

3.3 存储器属性

区域的属性与大小寄存器（MPU_RASR）是设置所有存储器属性的地方。该表显示了 MPU_RASR 寄存器中对于区域的属性和大小的简要描述。

表 2. MPU_RASR 寄存器中的区域属性和大小

位	名称	说明
28	XN	禁止执行
26:24	AP	数据访问权限字段（只读、读写、或无访问权限）
21:19	TEX	类型扩展字段
18	S	可共享
17	C	可缓存
16	B	可缓冲
15:8	SRD	子区域已禁用。对于每个子区域，1 = 已禁用，0 = 已启用。
5:1	SIZE	指定 MPU 保护区域的大小。

前一个表格中的参数详情如下：

- XN 标志位控制代码的执行。为了在区域内执行指令，特权级别必须有读访问权限，而 XN 必须为 0。否则，会产生 MemManage 报错。
- 数据访问权限（AP）字段定义存储区域的 AP。下表对访问权限进行了说明：

表 3. 区域访问权限

AP[2:0]	特权权限	非特权权限	说明
000	无访问权限	无访问权限	所有访问都会产生权限故障
001	RW	无访问权限	仅通过特权级软件访问
010	RW	RO	由非特权级软件写入会产生权限故障
011	RW	RW	完全访问权限
100	不可预测	不可预测	保留
101	RO	无访问权限	仅通过特权级软件读取
110	RO	RO	只读，由特权级或非特权级软件读取
111	RO	RO	只读，由特权级或非特权级软件读取

- S 字段面向可共享的存储区域：存储系统在一个有多个总线主控的系统（例如，一个处理器带一个 DMA 控制器）中提供总线主控之间的数据同步。强排序的存储器始终可共享。如果多个总线主控可以访问一个不可共享的存储区域，软件必须确保总线主控之间的数据一致性。STM32F7 系列和 STM32H7 系列不支持硬件一致性。S 字段相当于不可缓存的存储器。
- TEX、C 和 B 位用于定义区域的缓存属性，以及（在某种程度上）可共享性。按下表对其进行编码。

表 4. 缓存属性和可共享性

TEX	C	B	存储器类型	说明	可共享
000	0	0	强序	强序	有
000	0	1	设备	共享设备	有
000	1	0	正常	透写无写入分配	S 位
000	1	1	正常	回写，采用 no write allocate 策略	S 位
001	0	0	正常	不可缓存	S 位
001	0	1	保留	保留	保留
001	1	0	未定义	未定义	未定义
001	1	1	正常	回写，采用 write and read allocate 策略	S 位
010	0	0	设备	非共享设备	无
010	0	1	保留	保留	保留

- 子区域禁用位（SRD）标志特定的子区域是已启用或者已禁用。禁用某个子区域意味着与禁用区域重叠的另一个区域会匹配。如果没有其他已启用的区域与已禁用的子区域重叠，则 MPU 会报错。

对于实现缓存的产品（仅适用于实现 L1 缓存的 STM32F7 系列和 STM32H7 系列），额外的存储属性包括：

- 可缓存/不可缓存：表明专用区域可否缓存。
- 透写（采用 no write allocate 策略）：在写命中时，由其写入缓存和主存。在写失效时，由其更新主存中的块，而不是将块调入缓存中。
- 回写（采用 no write allocate 策略）：在写命中时，由其写入缓存并设置 dirty 位监测块的变化，不更新主存。在写失效时，由其更新主存中的块，而不将块调入缓存中。
- 回写（采用 write and read allocat 策略）：在写命中时，由其写入缓存并设置 dirty 位监测块的变化，不更新主存。在写失效时，由其更新主存中的块，并将块调入缓存中。

提示 *对于 Cortex-M7, TCM 存储器始终表现为不可缓存的非共享式普通存储器，与 MPU 中定义的存储器类型属性（针对包含 TCM 地址的存储区域）无关。*

否则，将针对 TCM 地址空间内的 MPU 区域，处理与之相关的访问权限，处理方法与处理 TCM 地址空间外部地址的方法相同。

3.4 Cortex-M7 强制推测性预取

Cortex-M7 具有推测性预取特性，允许推测性访问普通存储位置（例如：FMC、Quad-SPI 器件）。当一次推测性预取发生时，可能会影响对多个访问敏感的存储器或器件（如 FIFO、LCD 控制器）。当推测性预取发生时，也可能干扰其他带宽消耗较高的主控（如 LCD-TFT 或 DMA2D）产生的流量。为了保护普通存储器不受推测预取的影响，建议将存储器属性从普通更改为强排序或器件存储器（得益于 MPU）。如需详细了解存储器属性配置，请参阅第 6 节 带 STM32Cube HAL 的 MPU（采用 Armv6 和 Armv7 架构）设置示例。

4 Cortex-M33 存储器类型、寄存器和属性

虽然 MPU 操作的概念是相似的，但 Armv8-M 架构中的 MPU 的程序模型与之前版本的 M-profile Arm 架构中的 MPU 不同。

重要的是要认识到所有 MPU 寄存器是分组的。如果 Arm TrustZone®已启用，则有一组 MPU 寄存器用于安全状态，还有一组镜像寄存器用于非安全状态。在访问位于 0xE000 ED90 和 0xE000 EDC4 之间的 MPU 地址时，受访 MPU 寄存器的类型由处理器的当前状态决定。

非安全代码可以访问非安全 MPU 寄存器，安全代码可以访问安全 MPU 寄存器。安全代码可以访问非安全 MPU 寄存器的别名地址。

安全访问看到安全 MPU 寄存器，非安全访问看到非安全 MPU 寄存器。安全软件也可以访问使用别名地址的非安全 MPU 寄存器。

4.1 存储器类型和属性

在 Armv8-M 架构中，存储器类型分为：

- 正常存储器
- 器件存储器

提示

Armv6-M 和 Armv7-M 中的强排序（SO）器件存储器类型现在是器件存储器类型的一个子集。

一种正常的存储器类型被用于 MPU 区域，这些区域被用于访问通用指令或数据存储器。普通存储器允许处理器执行一些存储器访问优化，例如访问重排序或合并。普通存储器还允许对存储器进行缓存，适合存放可执行代码。普通存储器不能用于访问外部 MMIO 寄存器，器件存储器类型可用于该用途。对于 Armv7-M 架构，普通存储器定义基本保持不变。

普通存储器具有以下属性：

- 可缓存性：表明存储器可否缓存
- 可共享性：表明普通存储器可否共享
- 禁止执行：表明存储器可否执行（XN）

器件存储器必须用于覆盖外设控制寄存器的存储器区域。有时会允许对普通存储器进行一些优化（比如访问合并或重复），对于外设寄存器来说具有一定的风险。

器件存储器具有以下属性：

- G 或 nG：gathering 或 non-gathering。（对器件的多个访问（除了采用存储器排序语义的操作，例如存储器屏蔽指令、负载获取和存储释放）可以合并为单一事务）。
- R 或 nR：重排序
- E 或 nE：early write acknowledge（类似于 bufferable）

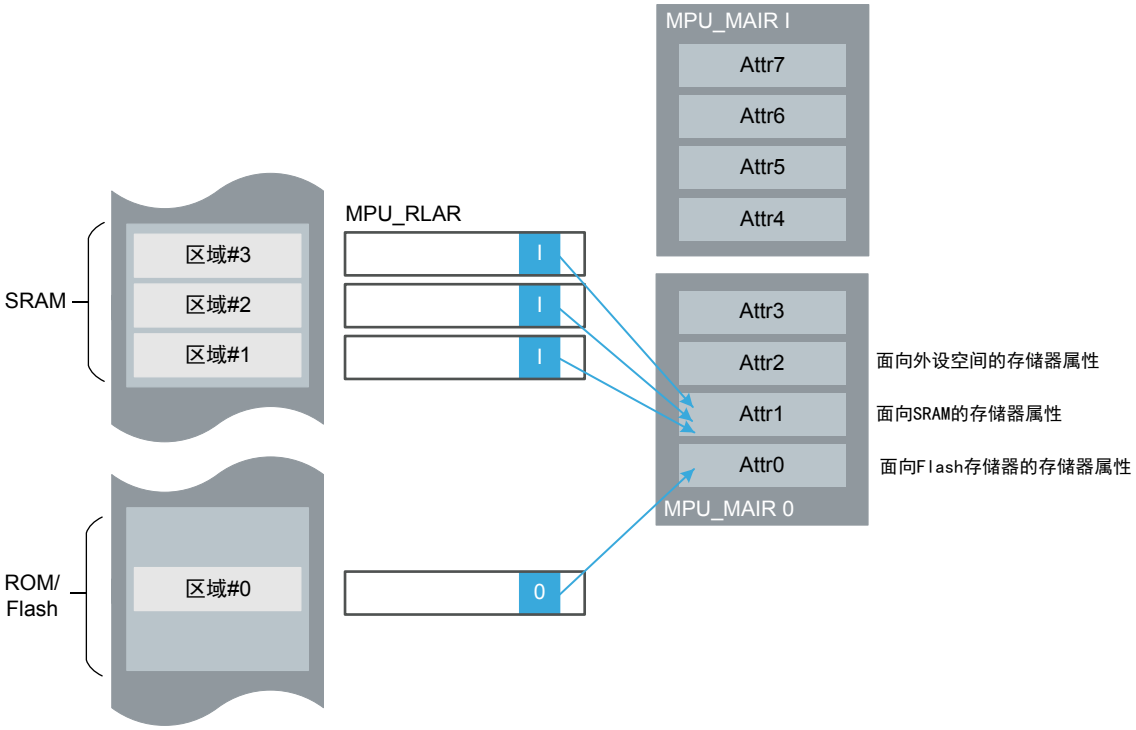
这些属性只有四种组合是有效的：

- device-nGnRnE：相当于 Armv7-M 强排序存储器类型
- device-nGnRE：相当于 Armv7-M 器件存储器
- device-nGRE：对于 Armv8-M 是新增的
- device-GRE：对于 Armv8-M 是新增的

4.2 属性间接

属性间接机制允许多个 MPU 区域共享一组存储器属性。例如，在下图中，MPU 区域 1、2 和 3 都分配给了 SRAM，因此它们可以共享与缓存相关的存储器属性。

图 4. 属性间接示例



与此同时，区域 1、2 和 3 仍然可以保留自己的访问权限、XN，以及可共享性属性。这是必需的，因为每个区域在应用程序中的用途可以不一样。

4.3 MPU 寄存器

Cortex-M33 MPU 寄存器不同于以前的 Cortex 内核，提供更多的灵活性，且兼容 Arm TrustZone。因此，以前产品中使用的编程方法不能应用于该产品。例如，MPU 区域起始地址和终止地址寄存器的引入使用户能够轻松定义保护区域的起始和终止地址。

表 5. Cortex-M33 MPU 寄存器

安全地址	NS 地址别名	寄存器	说明
0xE000ED90	0xE002ED90	MPU_TYPE	MPU 类型寄存器
0xE000ED94	0xE002ED94	MPU_CTRL	MPU 控制寄存器
0xE000ED98	0xE002ED98	MPU_RNR	MPU 区域编号寄存器
0xE000ED9C	0xE002ED9C	MPU_RBAR	MPU 区域基址寄存器
0xE000EDA0	0xE002EDA0	MPU_RLAR	MPU 区域起始地址和终止地址寄存器
0xE000EDA4	0xE002EDA4	MPU_RBAR_A1	MPU 区域起始地址寄存器别名 1
0xE000EDAC	0xE002EDAC	MPU_RBAR_A2	MPU 区域起始地址寄存器别名 2
0xE000EDB4	0xE002EDB4	MPU_RBAR_A3	MPU 区域起始地址寄存器别名 3
0xE000EDA8	0xE002EDA8	MPU_RLAR_A1	MPU 区域终止地址寄存器别名 1
0xE000EDB0	0xE002EDB0	MPU_RLAR_A2	MPU 区域终止地址寄存器别名 2
0xE000EDB8	0xE002EDB8	MPU_RLAR_A3	MPU 区域终止地址寄存器别名 3
0xE000EDC0	0xE002EDC0	MPU_MAIR0	MPU 存储器属性间接寄存器 0
0xE000EDC4	0xE002EDC4	MPU_MAIR1	MPU 存储器属性间接寄存器 1

5 Cortex 内核之间的 MPU 特性对比

Cortex-M0+、Cortex-M3/M4、Cortex-M7、Cortex-M33 的 MPU 有些许差异。如果要使用 MPU 配置软件，用户必须知道这些。下表说明了这些差异

表 6. Cortex 内核之间的 MPU 特性对比

特性	Cortex-M0+	Cortex-M3/M4	Cortex-M7	Cortex-M33
区域编号	8	8	8/16 (1)(2)	8 MPU_S / 8 MPU_NS
区域地址	有	有	有	有
区域容量	256 字节 - 4 GB 字节	32 字节 - 4 GB 字节	32 字节 - 4 GB 字节	32 字节 - 4 GB 字节
区域存储属性	S、C、B、XN(3)	TEX, S, C, B, XN	TEX, S, C, B, XN	S、C、E(4)，G (5)，R(6)，XN
区域访问权限（AP）	有	有	有	有（有特权或没有）
子区域可禁用	8 位	8 位	8 位	NA
在NMI/HardFault中，绕开MPU。	有	有	有	有
MPU 寄存器别名	无	有	有	有
故障异常	仅限 HardFault	HardFault/MemManage	HardFault/MemManage	HardFault/MemManage

1. *面向 STM32H7 系列器件。*
2. *面向 STM32F7 系列器件。*
3. *Cortex-M0+支持一个等级缓存策略。这就是 TEX 字段在 Cortex-M0+处理器中不可用的原因。*
4. *Early write acknowledge（类似于 bufferable）*
5. *Gathering*
6. *Reordering*

6 在STM32Cube中使用HAL时的MPU（采用 Armv6 和 Armv7 架构）设置示例

下表给出了 MPU 的存储区域设置示例：内部 SRAM、Flash 存储，以及外设。默认存储器映射用于特权访问（作为后台区域），对于HardFault 处理程序和NMI，MPU没有被使能。
内部 SRAM：内部 SRAM 的 8 Kb 配置为 Region0
存储器属性：可共享存储器，透写（采用 no write allocate 策略），具有完全访问权限且启用了代码执行
Flash 存储器：整个 Flash 存储器配置为 Region。
存储器属性：非可共享存储器，透写（采用 no write allocate 策略），具有完全访问权限且启用了代码执行
外设区域：配置为 Region2
存储器属性：共享器件，具有完全访问权限且禁止执行

表 7. MPU 设置示例

配置	存储器类型	基址	区域编号	存储器容量	存储器属性
内部 SRAM	正常存储器	0x2000 0000	Region0	8 KB	可共享，透写，采用 no write allocate 策略 C = 1，B = 0，TEX = 0，S = 1 SRD = 0，XN = 0，AP = 完全 访问权限
Flash	正常存储器	0x0800 0000	Region1	1 MB	非可共享，透写，采用 no write allocate 策略 C = 1，B = 0，TEX = 0，S = 0 SRD = 0，XN = 0，AP = 完全 访问权限
FMC	正常存储器	0x6000 0000	Region2	512 MB	可共享，透写，采用 no write allocate 策略 C = 1，B = 0，TEX = 0，S = 1 SRD = 0，XN = 0，AP = 完全 访问权限

在STM32Cube中使用HAL配置MPU

```
void MPU_RegionConfig(void)
{
MPU_Region_InitTypeDef MPU_InitStruct;
/* 禁用 MPU */
HAL_MPU_Disable();
/* 将 RAM 区域配置为 Region N°0，存储容量 8kB，读/写区域 */
MPU_InitStruct.Enable = MPU_REGION_ENABLE;
MPU_InitStruct.BaseAddress = 0x20000000;
MPU_InitStruct.Size = MPU_REGION_SIZE_8KB;
MPU_InitStruct.AccessPermission = MPU_REGION_FULL_ACCESS;
MPU_InitStruct.IsBufferable = MPU_ACCESS_NOT_BUFFERABLE;
MPU_InitStruct.IsCacheable = MPU_ACCESS_CACHEABLE;
MPU_InitStruct.IsShareable = MPU_ACCESS_SHAREABLE;
MPU_InitStruct.Number = MPU_REGION_NUMBER0;
MPU_InitStruct.TypeExtField = MPU_TEX_LEVEL0;
MPU_InitStruct.SubRegionDisable = 0x00;
MPU_InitStruct.DisableExec = MPU_INSTRUCTION_ACCESS_ENABLE;
HAL_MPU_ConfigRegion(&MPU_InitStruct);
/* 将 FLASH 区域配置为 REGION N°1，存储容量 1MB，读/写区域 */
MPU_InitStruct.BaseAddress = 0x08000000;
MPU_InitStruct.Size = MPU_REGION_SIZE_1MB;
MPU_InitStruct.IsShareable = MPU_ACCESS_NOT_SHAREABLE;
MPU_InitStruct.Number = MPU_REGION_NUMBER1;
HAL_MPU_ConfigRegion(&MPU_InitStruct);
/* 将 FMC 区域配置为 REGION N°2，存储容量 0.5GB，读/写区域 */
MPU_InitStruct.BaseAddress = 0x60000000;
MPU_InitStruct.Size = MPU_REGION_SIZE_512MB;
MPU_InitStruct.IsShareable = MPU_ACCESS_SHAREABLE;
MPU_InitStruct.Number = MPU_REGION_NUMBER2;
HAL_MPU_ConfigRegion(&MPU_InitStruct);
/* 使能 MPU */
HAL_MPU_Enable(MPU_PRIVILEGED_DEFAULT);
}
```

7 结论

在 STM32 微控制器中加入 MPU 使其稳健、可靠，而且在某些情况下更安全 - 防止应用程序任务访问或破坏其他任务使用的堆栈和数据存储器。

该应用笔记描述了不同的存储器存属性、类型和 MPU 寄存器。其中还提供了 MPU（使用 STM32Cube HAL）设置示例，以说明如何配置 STM32 MCU 中的 MPU。

如需详细了解 MPU 寄存器，请参阅 Cortex 内核编程手册。

版本历史

表 8. 文档版本历史

日期	版本	变更
2016 年 3 月 24 日	1	初始版本。
2018 年 5 月 4 日	2	整个文档中增加了 STM32H7 系列。 更新了图 1。区域重叠示例… 增加了第 1 节：常规信息 增加了第 3.4 节 Cortex-M7 强制推测预取…
2019 年 7 月 17 日	3	更新了简介，新增 STM32G0 系列、STM32G4 系列、STM32L4+系列、STM32L5 系列和 STM32WB 系列。
2020 年 2 月 10 日	4	增加了： • 章节介绍中增加了 PM0214 • 第 4 节 CM33 的存储器类型、寄存器和属性 • 第 5 节 Cortex-M0+、Cortex-M3/M4、Cortex-M7，以及 Cortex-M33 的 MPU 特性对比 更新了： • 文档标题 • 章节介绍 • 第 2 节 概述 • 第 2.1 节 存储器模型
2021 年 9 月 20 日	5	更新了： • 表 1. 适用产品

目录

表一览

图一览