SPC1068 MPU 单元使用指南

概述

在 Cortex-M3/4 处理器中可以选配一个存储器保护单元（MPU），实施对存储器（主要是内存和外设寄存器）的保护，从而使软件更加可靠。如果打算启用 MPU，则在使用前，必须根据需要对其编程；如果没有启用 MPU，则等同于系统中没有配 MPU。

注意：　本文档主要以 SPC1068 为例进行介绍。

目录

1 MPU 概述 7
1.1 功能简介 7
1.2 基本特性描述 7
2 MPU 寄存器 9
2.1 寄存器列表 9
2.2 MPU 类型寄存器（TYPE） 9
2.3 MPU 控制寄存器（CTRL） 10
2.4 MPU Region 号寄存器（RNR） 11
2.5 MPU Region 基址寄存器（RBAR） 11
2.6 MPU Region 属性及大小寄存器（RASR） 12
2.6.1 MPU Region 属性常见位域介绍 12
2.6.2 地址空间不同属性对于写操作的影响 12
2.6.3 MPU Region 属性复杂位域介绍 14
2.7 MPU Region 别名寄存器 14
2.8 微控制器常规地址空间属性 15
3 MPU 使用举例 16
3.1 配置 MPU 流程图 16
3.2 使用 RNR 寄存器配置 MPU 示例代码 19
3.3 不使用 RNR 寄存器配置 MPU 示例代码 24
3.4 使用别名寄存器配置 MPU 示例代码 29
4 MPU 使用时的考虑因素 33

图片列表

图 1–1：MPU 功能示意图 8
图 2–1：Background Region 功能示意图 10
图 2–2：内存属性对写入/读取操作的影响 13
图 3–1：配置 MPU 流程图 16

表格列表

表 2-1：MPU 寄存器列表 9
表 2-2：MPU 类型寄存器（TYPE） 9
表 2-3：MPU 控制寄存器（CTRL） 10
表 2-4：MPU Region 号寄存器（RNR） 11
表 2-5：MPU Region 基址寄存器（RBAR） 11
表 2-6：MPU Region 属性及大小寄存器（RASR） 12
表 2-7：MPU Region 属性复杂位域列表 14
表 2-8：片内片外访问属性控制列表 14
表 2-9：MCU 常见地址属性 15

版本历史

版本	日期	作者	状态	变更
C/0	2024-02-21	周佳莉	Released	首次发布。

术语或缩写

术语或缩写	描述
MPU	Memory Protection Unit，存储器保护单元

1 MPU 概述

指针在 C 语言中由于其狂野不羁的性格，和放任自由的汇编相差无几，而闻名于码农界，它是一把双刃剑，用的好，会让编码简介且高效，用的不好，则无法预料其结果；在指针中有一类更是臭名昭著——野指针（指某个指针变量的值因故超出合法的范围），程序逻辑错误、数组越界、堆栈溢出、指针未经初始化、对缓存与缓冲的处理不当、多任务环境中的紊乱危象，甚至是恶意地破坏等，都可以制造出野指针，在一些使命——关键系统中，这些操作将会产生致命的结果；所以，对系统中某些地址控件进行保护在某些应用中是必不可少的部分，而在 Cortex-M3/4 中提供了 MPU 单元，此单元的功能就是对特定的地址控件进行保护，本文档将尽可能详细介绍 MPU 的使用方法。

1.1 功能简介

在 Cortex-M3/4 处理器中可以选配一个存储器保护单元（MPU），它可以实施对存储器（主要是内存和外设寄存器）的保护，从而使软件更加健壮和可靠。如果打算启用 MPU，则在使用前，必须根据需要对其编程，如果没有启用 MPU，则等同于系统中没有配 MPU。MPU 有如下功能可以提高系统的可靠性：

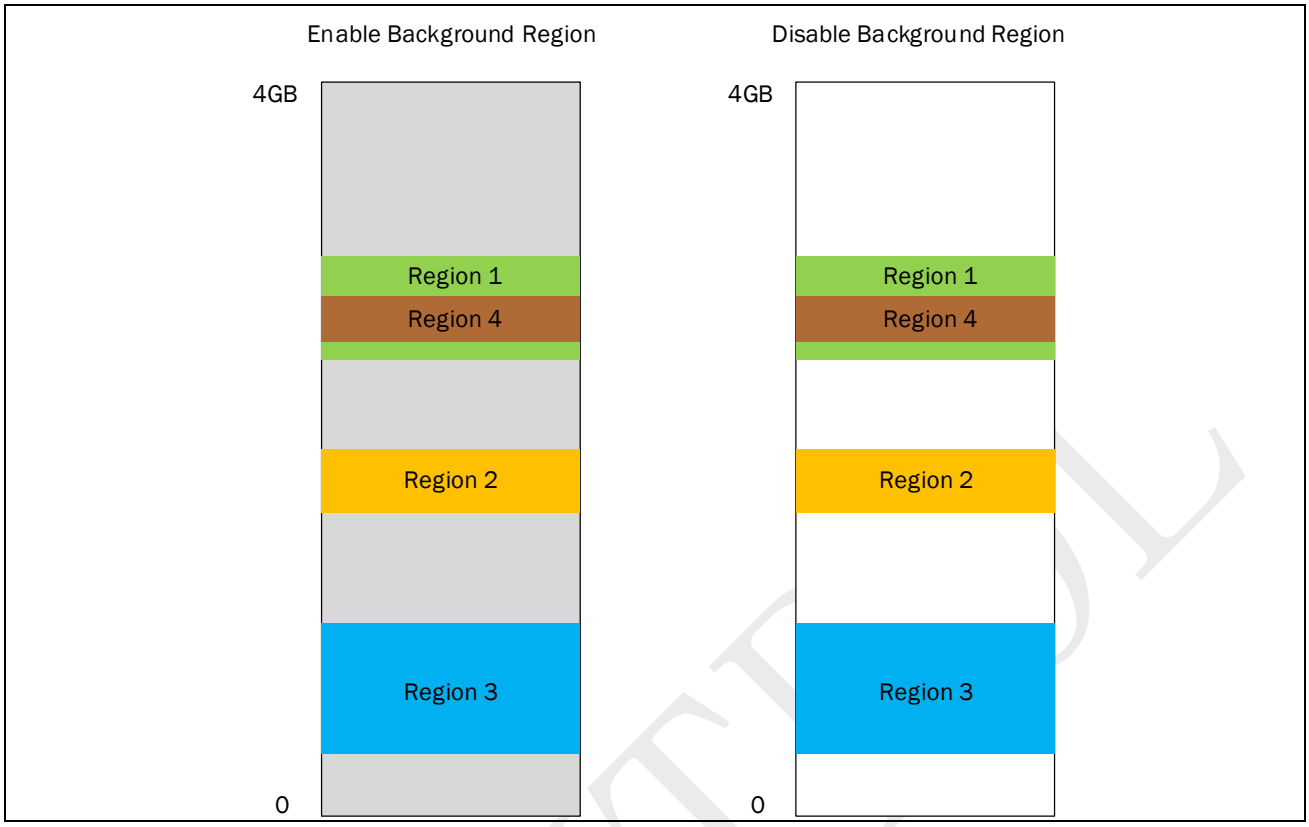
- 阻止用户应用程序破坏操作系统使用的数据。
- 阻止一个任务访问其它任务的数据区，从而把任务隔开。
- 可以把关键数据区设置为只读，从根本上消除了被破坏的可能。
- 检测意外的存储访问；如：堆栈溢出，数组越界。
- 设置存储器 Regions 的其它属性；比如：是否 Cacheable、是否 Bufferable 等。

1.2 基本特性描述

Region ： MPU 在执行其功能时的单位。一个 Region 其实就是一段连续的地址，只是它们的位置和范围都要满足一些限制（地址对齐、最小容量等）。

- CM3/4 共支持 8 个 Region，每个 Region 可以单独 Enable 或 Disable，且每个 Region 最小为 256 Bytes；每个 Region 的基址必须与 Region 容量大小对齐，例如：Region 容量大小为 64KB，那么 Region 基址必须要能被 64KB 整除（低 16bit 为 0 的地址，0x10000、0x20000 等）。
- 每个 Region 进一步划分成更小的“Sub-Region”，在进一步划分 Sub-Region 时，每个 Region 只能等分成 8 个 Sub-Region，且每个 Sub-Region 可以单独 Enable/Disable。
- 允许启用一个“Background Region”（如图 1-1），不过它只能由特权级使用。在启用 MPU 后，就不得再访问定义之外的地址区间，否则，将以“访问违例”处理，并触发 MemManage Fault。
- MPU 定义的 Regions 可以相互交叠。如果某块内存落在多个 Region 中，则访问属性和权限将由编号最大的 Region 来决定；比如，若 1 号 Region 与 4 号 Region 交迭，则交叠的部分受 4 号 Region 的属性控制，如图 1-1 所示。

图 1–1：MPU 功能示意图



2 MPU 寄存器

2.1 寄存器列表

表 2-1：MPU 寄存器列表

Address	Register	CMSIS-Core Symbol	Function
0xE000ED90	MPU Type Register	MPU->TYPE	Provides information about the MPU
0xE000ED94	MPU Control Register	MPU->CTRL	MPU enable/disable and background region control
0xE000ED98	MPU Region Number Register	MPU->RNR	Select which MPU region to be configured
0xE000ED9C	MPU Region Base Address Register	MPU->RBAR	Defines base address of a MPU region
0xE000EDA0	MPU Region Base Attribute and Size Register	MPU->RASR	Defines size and attributes of a MPU region
0xE000EDA4	MPU Alias 1 Region Base Address Register	MPU->RBAR_A1	Alias of MPU->RBAR
0xE000EDA8	MPU Alias 1 Region Base Attribute and Size Register	MPU->RASR_A1	Alias of MPU->RASR
0xE000EDAC	MPU Alias 2 Region Base Address Register	MPU->RBAR_A2	Alias of MPU->RBAR
0xE000EDB0	MPU Alias 2 Region Base Attribute and Size Register	MPU->RASR_A2	Alias of MPU->RASR
0xE000EDB4	MPU Alias 3 Region Base Address Register	MPU->RBAR_A3	Alias of MPU->RBAR
0xE000EDB8	MPU Alias 3 Region Base Attribute and Size Register	MPU->RASR_A3	Alias of MPU->RASR

2.2 MPU 类型寄存器（TYPE）

表 2-2：MPU 类型寄存器（TYPE）

Bit	Name	Type	Reset Value	Description
23：16	IREGION	R	0	MPU 支持的指令 Region 个数，由于 ARMv7-M 结构采用一个一元化的 MPU，所以此位段一直为 0。
15：8	DREGION	R	0 or 8	MPU 支持的 Region 总个数，对于 Cortex-M3/4 而言，此位段要么是 0（没有 MPU），要么是 8（有 MPU）。
0	SEPARATE	R	0	因为采用的是一个一元化的 MPU，所以此位段固定为 0。

使用类型寄存器的 DREGION 位段，可以查看芯片内部是否配备有 MPU。

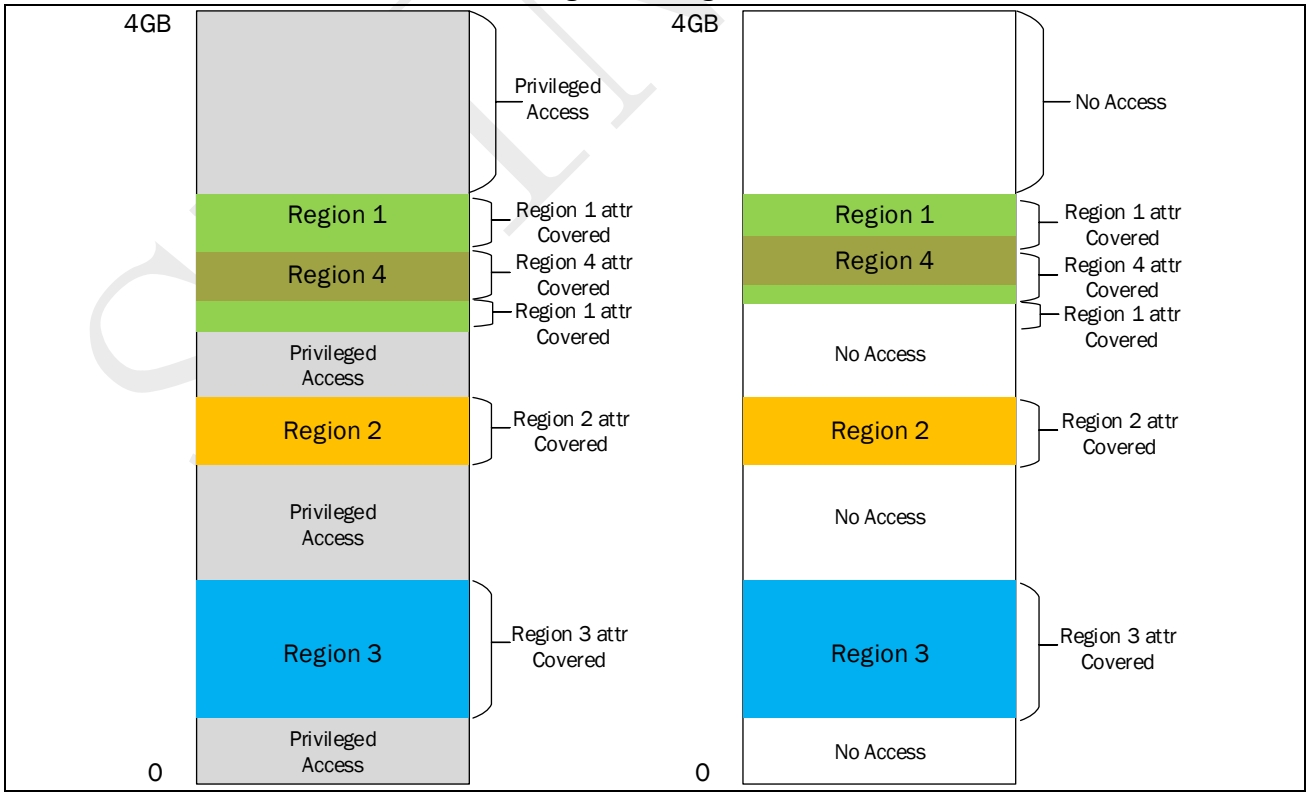
2.3 MPU 控制寄存器（CTRL）

表 2-3：MPU 控制寄存器（CTRL）

Bit	Name	Type	Reset Value	Description
2	PRIVDEFENA	RW	0	特权模式可访问的默认地址空间使能位。 1：当 Bit 1 = 1 时，全地址空间将作为 Background Region 供特权模式使用。 0：当 Bit 0 = 0 时，Background Region 关闭，任何对未使能的 Region 的访问将会引起 MemManage Fault。
1	HFNMIENA	RW	0	1：在 Hard Fault/NMI 处理例程中使能 MPU 0：在 Hard Fault/NMI 处理例程中不使能 MPU
0	ENABLE	RW	0	1：使能 MPU 0：关闭 MPU

若 PRIVDEFENA 位置位，可以在没有建立任何 Region 就使能 MPU 的情况下，依然允许特权级程序访问全地址空间，但用户级程序对全地址空间的访问将被拒之门外。然而，如果设置了其它 Region，并且使能 MPU，则 Backround Region 与这些 Region 重合的部分，就要受各 Region 的属性限制。具体如下图所示：

图 2–1：Background Region 功能示意图



另外需要注意：

- 如果没有非常特别的需求，就一定要在 Region 完全配置完成之后再 Enbale MPU，否则可能会出现意外的 MemManage Fault。最为保险的做法是：在配置 MPU 之前 Disable MPU，然后执行 MPU 配置，最后 Enable MPU。
- 响应异常时的取向量提取动作对 MPU 而言一直是允许的，而且没有必要对系统私有外设总线（PPB，包括 MPU、NVIC、SysTick、ITM）及系统控制空间（SCS）设置 Region 进行保护，因为这些地址空间都只有特权级才能访问。因此，不管 MPU Region 如何限制，响应异常时的取向量提取动作及对系统区的访问不会受到影响。

2.4 MPU Region 号寄存器（RNR）

表 2-4：MPU Region 号寄存器（RNR）

Bit	Name	Type	Reset Value	Description
7：0	REGION	RW	-	Region 编号选择寄存器。 选择需要进行属性配置的 Region 编号，由于总共只有 8 个 Region，所以实际上只有[2：0]bit 有效。

选好 Region 后，就可以在另外两个寄存器（RBAR 及 RASR）中对该 Region 的属性进行配置。

2.5 MPU Region 基址寄存器（RBAR）

表 2-5：MPU Region 基址寄存器（RBAR）

Bit	Name	Type	Reset Value	Description
31：N	ADDR	RW	-	Region 的基址，N 的数值取决于 Region 的容量大小（也即 1.1 节中所述，要对齐于 Region 的容量大小）；例如，Region 容量大小为 64K，则基址范围域为[31：16]，也即低 16bit 为 0，基址对齐于 64K。
4	VALID	RW	-	如果置 1，则[3：0]的数值将覆盖 RNR 寄存器中设定的 Region 号。
3：0	REGION	RW	-	当 VALID 位置位时，REGION 将覆盖 RNR 寄存器的数值。由于 Cortex-M3/4 只有 8 个 Region，所以当 REGION 数值大于 7 时，将忽略此值。

2.6 MPU Region 属性及大小寄存器（RASR）

表 2-6：MPU Region 属性及大小寄存器（RASR）

<table>
<tr><th>Bit</th><th>Name</th><th>Type</th><th>Reset Value</th><th>Description</th></tr>
<tr><td>31：29</td><td>Reserved</td><td>-</td><td>-</td><td>-</td></tr>
<tr><td>28</td><td>XN</td><td>RW</td><td>-</td><td>指令访问控制位。
1：此 Region 禁止取指。
0：此 Region 可以取指。</td></tr>
<tr><td>27</td><td>Reserved</td><td>-</td><td>-</td><td>-</td></tr>
<tr><td>26：24</td><td>AP</td><td>RW</td><td>-</td><td>数据访问许可域。
<table>
<tr><th>Value</th><th>Privilege Access</th><th>User Access</th><th>Description</th></tr>
<tr><td>000</td><td>不可访问</td><td>不可访问</td><td>不可访问</td></tr>
<tr><td>001</td><td>读/写</td><td>不可访问</td><td>只有特权级能访问</td></tr>
<tr><td>010</td><td>读/写</td><td>只读</td><td>用户级程序执行写操作会出错</td></tr>
<tr><td>011</td><td>读/写</td><td>读/写</td><td>Full Access</td></tr>
<tr><td>100</td><td>N/A</td><td>N/A</td><td>N/A</td></tr>
<tr><td>101</td><td>只读</td><td>不可访问</td><td>只有特权级能执行读操作</td></tr>
<tr><td>110</td><td>只读</td><td>只读</td><td>只读</td></tr>
<tr><td>111</td><td>只读</td><td>只读</td><td>只读</td></tr>
</table></td></tr>
<tr><td>23：22</td><td>Reserved</td><td>-</td><td>-</td><td>-</td></tr>
<tr><td>21：19</td><td>TEX</td><td>RW</td><td>-</td><td>Region 类型扩展域</td></tr>
<tr><td>18</td><td>S</td><td>RW</td><td>-</td><td>Shareable（是否可共享）</td></tr>
<tr><td>17</td><td>C</td><td>RW</td><td>-</td><td>Cacheable（是否可缓存）</td></tr>
<tr><td>16</td><td>B</td><td>RW</td><td>-</td><td>Bufferable（是否可缓冲）</td></tr>
<tr><td>15：8</td><td>SRD</td><td>RW</td><td>-</td><td>Sub-Region 关闭</td></tr>
<tr><td>7：6</td><td>Reserved</td><td>-</td><td>-</td><td>-</td></tr>
<tr><td>5：1</td><td>REGIONSIZE</td><td>RW</td><td>-</td><td>MPU Region 大小</td></tr>
<tr><td>0</td><td>ENABLE</td><td>RW</td><td>-</td><td>Region 使能</td></tr>
</table>

2.6.1 MPU Region 属性常见位域介绍

- XN：控制此 Region 内是否允许取指，其作用是把新得到的还不受信任的代码先存储到此区，待经过审核鉴定，确认可信任之后，再转移到其它 Region，以允许取指执行。
- SRD：每个 BIT 位控制每个 Sub-Region 的 Enable/Disable。
- REGIONSIZE：设置 Region 的容量大小，单位是 Byte，每个 Region 的最小容量是 128 Bytes。
- ENABLE：Region 的使能控制位。

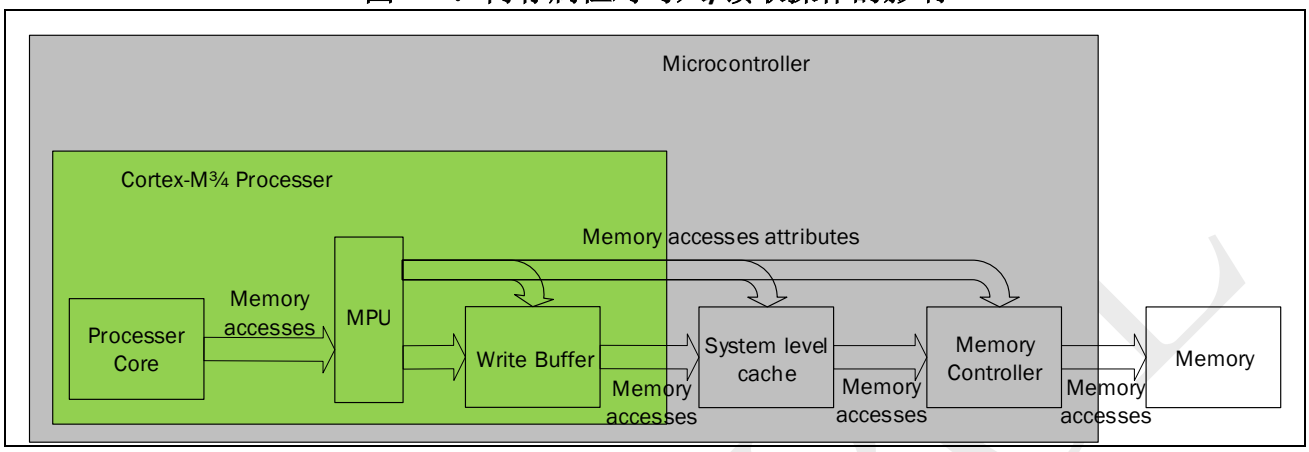
2.6.2 地址空间不同属性对于写操作的影响

在 RASR 寄存器中还有一些控制域（TEX、S、C、B）对应着存储系统中比较高级的概念。CM3/4 中没有缓存（cache），但是 CM3/4 采用 v7-M 架构设计，而 v7-M 支持外部缓存（相当于 L2 缓存，通常 L1 Cache 称为 inner（片内存储器）缓存，L2 Cache 称为 outer（片外存储器）

缓存）以及更先进的存储器系统。按 v7-M 的规格说明，可以使用 TEX、S、C、B 位段，以达到支持多样的内存管理模型。

在进一步介绍以上位段的使用方式之前，先要对系统中写入及读取操作的一些概念做些介绍：

图 2–2：内存属性对写入/读取操作的影响



- ARM Cache 架构由 Cache 存储器及写缓冲（Write-Buffer）组成，其中 Write-Buffer 是由 Cache 按照 FIFO 原则向主存写的缓冲器。
- Bufferable 与 non-Buffer 的主要区别表现在 ACK 信号的返回方式上。
- Bufferable：数据写入 Buffer 后，（某个 component）便返回 ACK；
- Non-Buffer：等数据存入到主存，再返回 ACK；

按照是否需要将写入数据进行缓存，可以将写入操作分为：

- Write-Through（写通）：同时将数据写入 Cache 及 Back-Device（主存或者外部设备），这种方式的优点是不会丢失数据，缺点是速度慢。
- Write-Back（回写）：将更新的数据写入 Cache，只有在数据替换出缓存时，更新的数据才会被写入到 Back-Device（主存或者外部设备），此种方式的优点是速度快，缺点是，如果更新的数据在没有回写到 Back-Device 时断电，数据将无法找回。

在以上的两种写入操作方式中（Write-Through/Write-Back），写入 Cache 时发生 Cache miss 的情况，针对这两种情况，我们又可以将写入方式定义两种策略：

- Write-allocate：当数据写入 Cache 时，发生 Cache miss，则先将数据从 Back-Device 读到 Cache，然后再按照 Write-Back 的方式写数据。此时对应的读操作与写操作类似。
- Write-non-allocate：直接将数据写入 Back-Device。此时对应的读操作会将数据先读取到 Cache，然后再执行度操作。

需要注意的是，不管是 Write-Through 还是 Write-Back，在 Cache miss 时都可以使用 Write-allocate 或者 Write-non-allocate，只是一般来说，Write-Back 配合 Write-allocate 使用，Write-Through 配合 Write-non-allocate 使用，因为多次往同一 Cache 写入数据时，Write-Back 配合 Write-allocate 可以提升性能，但是此时 Write-allocate 对 Write-Through 没有帮助。

2.6.3 MPU Region 属性复杂位域介绍

表 2-7：MPU Region 属性复杂位域列表

TEX	C	B	Description	Region Shareability
b000	0	0	Strongly Ordered（指令严格按照代码的顺序执行）	Shareable
b000	0	1	共享设备（写操作带 Buffer）	Shareable
b000	1	0	对于 Outer 和 inner 都是写通模式；Cache miss 时采用 no Write-allocate	[S]
b000	1	1	对于 Outer 和 inner 都是回写模式；Cache miss 时采用 no Write-allocate	[S]
b001	0	0	在 outer 和 inner 上都采用 non-cacheable 策略	[S]
b001	0	1	Reserved	Reserved
b001	1	0	按照具体实现定义	-
b001	1	1	对于 Outer 和 inner 都是回写模式；Cache miss 时读写都采用 Write-allocate	[S]
b010	0	0	非共享设备	Not Shared
b010	0	1	Reserved	Reserved
b010	1	X	Reserved	Reserved
b1BB	A	A	被缓存的内存地址空间；BB=outer 策略，AA=inner 策略	[S]
表格中[S]表示是否可共享（多处理器共享）取决于 S 位域				

当 TEX 的 MSB=1 时，如果该 Region 是片内存储器，则由 C 和 B 决定其缓存属性（AA）；如果是片外存储器，则由 TEX 的[1：0]决定其缓存属性（BB）。不管是 AA 还是 BB，每个数值的含义都是相同的，如下表所示：

表 2-8：片内片外访问属性控制列表

存储器属性编码（AA and BB）	高速缓存策略
00	不可缓存
01	写操作采用回写模式，当发生 Cache miss 时，写操作和读操作都采用 allocate 策略
10	写操作采用写通模式，当发生 Cache miss 时，写操作采用 no allocate 策略
11	写操作采用回写模式，当发生 Cache miss 时，写操作采用 no allocate 策略

2.7 MPU Region 别名寄存器

2.1 节寄存器表的最后 6 个寄存器（RBAR_A1/ RASR_A1、RBAR_A2/ RASR_A2、RBAR_A3/ RASR_A3），其实就是 RBAR 及 RASR 的别名寄存器（访问这些寄存器就相当于访问 RBAR 及 RASR），且它们在地址上是连续的，所以当需要配置多个 Region 的属性时，可以直接采用汇编代码 LDM/STM 指令一次完成配置操作，而无需采用“关闭 Region->更改 Region 属性配置->使能 Region”这种小心翼翼的操作，这样既简化了代码，也提高了程序的执行效率。

举个例子：

```
;  R1 = 一个指针，指向某 RTOS 进程控制块中的 4 个 Region 对子（共 8 个字）
MOV R0,  #NVIC_BASE
ADD R0,  #MPU_REG_CTRL
LDM R1,  [R2-R9] ;  加载 4 个 Region 的信息
STM R0,  [R2-R9] ;  一句话完成 4 个 Region 的配置
```

2.8 微控制器常规地址空间属性

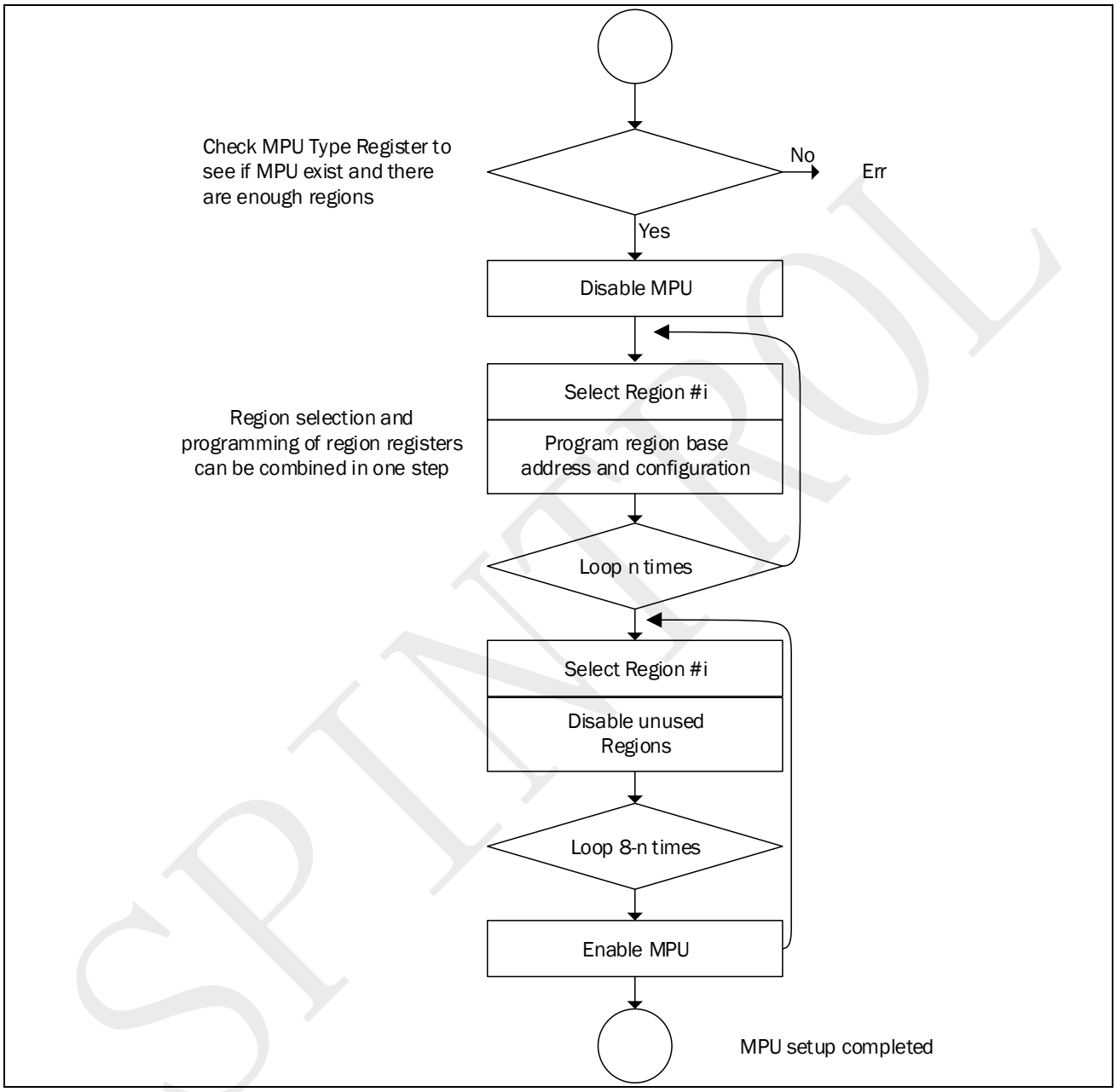
表 2-9：MCU 常见地址属性

TYPE	Memory Type	Commonly Used Memory Attributes
ROM/FLASH（Program memories）	Normal memory	不可共享，写数据时采用写通方式 C=1，B=0，TEX=0，S=0
Internal SRAM	Normal memory	可共享，写数据时采用写通方式 C=1，B=0，TEX=0，S=1
External SRAM	Normal memory	可共享，写数据时采用回写方式 C=1，B=1，TEX=0，S=1
Peripherals	Device	可共享设备 C=0，B=1，TEX=0，S=1

3 MPU 使用举例

3.1 配置 MPU 流程图

图 3–1：配置 MPU 流程图



在此我们为代码准备一些宏定义：

代码示例宏定义

```
#define MPU_DEFS_RASR_SIZE_32B                (0x04 <<
MPU_RASR_SIZE_Pos)
#define MPU_DEFS_RASR_SIZE_64B                (0x05 <<
MPU_RASR_SIZE_Pos)
#define MPU_DEFS_RASR_SIZE_128B               (0x06 <<
MPU_RASR_SIZE_Pos)
```

```
#define MPU_DEFS_RASR_SIZE_256B             (0x07 <<
MPU_RASR_SIZE_Pos)
#define MPU_DEFS_RASR_SIZE_512B             (0x08 <<
MPU_RASR_SIZE_Pos)
#define MPU_DEFS_RASR_SIZE_1KB              (0x09 <<
MPU_RASR_SIZE_Pos)
#define MPU_DEFS_RASR_SIZE_2KB              (0x0A <<
MPU_RASR_SIZE_Pos)
#define MPU_DEFS_RASR_SIZE_4KB              (0x0B <<
MPU_RASR_SIZE_Pos)
#define MPU_DEFS_RASR_SIZE_8KB              (0x0C <<
MPU_RASR_SIZE_Pos)
#define MPU_DEFS_RASR_SIZE_16KB             (0x0D <<
MPU_RASR_SIZE_Pos)
#define MPU_DEFS_RASR_SIZE_32KB             (0x0E <<
MPU_RASR_SIZE_Pos)
#define MPU_DEFS_RASR_SIZE_64KB             (0x0F <<
MPU_RASR_SIZE_Pos)
#define MPU_DEFS_RASR_SIZE_128KB            (0x10 <<
MPU_RASR_SIZE_Pos)
#define MPU_DEFS_RASR_SIZE_256KB            (0x11 <<
MPU_RASR_SIZE_Pos)
#define MPU_DEFS_RASR_SIZE_512KB            (0x12 <<
MPU_RASR_SIZE_Pos)
#define MPU_DEFS_RASR_SIZE_1MB              (0x13 <<
MPU_RASR_SIZE_Pos)
#define MPU_DEFS_RASR_SIZE_2MB              (0x14 <<
MPU_RASR_SIZE_Pos)
#define MPU_DEFS_RASR_SIZE_4MB              (0x15 <<
MPU_RASR_SIZE_Pos)
#define MPU_DEFS_RASR_SIZE_8MB              (0x16 <<
MPU_RASR_SIZE_Pos)
#define MPU_DEFS_RASR_SIZE_16MB             (0x17 <<
MPU_RASR_SIZE_Pos)
#define MPU_DEFS_RASR_SIZE_32MB             (0x18 <<
MPU_RASR_SIZE_Pos)
#define MPU_DEFS_RASR_SIZE_64MB             (0x19 <<
MPU_RASR_SIZE_Pos)
#define MPU_DEFS_RASR_SIZE_128MB            (0x1A <<
MPU_RASR_SIZE_Pos)
#define MPU_DEFS_RASR_SIZE_256MB            (0x1B <<
MPU_RASR_SIZE_Pos)
#define MPU_DEFS_RASR_SIZE_512MB            (0x1C <<
MPU_RASR_SIZE_Pos)
```

```
#define MPU_DEFS_RASR_SIZE_1GB                (0x1D <<
MPU_RASR_SIZE_Pos)
#define MPU_DEFS_RASR_SIZE_2GB                (0x1E <<
MPU_RASR_SIZE_Pos)
#define MPU_DEFS_RASR_SIZE_4GB                (0x1F <<
MPU_RASR_SIZE_Pos)
#define MPU_DEFS_RASE_AP_NO_ACCESS             (0x0 <<
MPU_RASR_AP_Pos)
#define MPU_DEFS_RASE_AP_PRIV_RW              (0x1 <<
MPU_RASR_AP_Pos)
#define MPU_DEFS_RASE_AP_PRIV_RW_USER_RO       (0x2 <<
MPU_RASR_AP_Pos)
#define MPU_DEFS_RASE_AP_FULL_ACCESS          (0x3 <<
MPU_RASR_AP_Pos)
#define MPU_DEFS_RASE_AP_PRIV_RO              (0x5 <<
MPU_RASR_AP_Pos)
#define MPU_DEFS_RASE_AP_RO                  (0x6 <<
MPU_RASR_AP_Pos)
#define MPU_DEFS_NORMAL_MEMORY_WT             (MPU_RASR_C_Msk)
#define MPU_DEFS_NORMAL_MEMORY_WB             (MPU_RASR_C_Msk |
MPU_RASR_B_Msk)
#define MPU_DEFS_NORMAL_SHARED_MEMORY_WT       (MPU_RASR_C_Msk |
MPU_RASR_S_Msk)
#define MPU_DEFS_NORMAL_SHARED_MEMORY_WB
 (MPU_DEFS_NORMAL_MEMORY_WB | MPU_RASR_S_Msk)
#define MPU_DEFS_SHARED_DEVICE                (MPU_RASR_B_Msk)
#define MPU_DEFS_STRONGLY_ORDERED_DEVICE       (0x0)
#define MPU_DEFS_MODE_PRIVILEGE               (0x0)
#define MPU_DEFS_MODE_UNPRIVILEGE             (0x1)
```

3.2 使用 RNR 寄存器配置 MPU 示例代码

RNR 配置 MPU 示例代码

```
#include "spc1068.h"
#include <stdio.h>
uint32_t ControlVal,  u32Privilege;

int MPU_SetUp (void)
{
  uint32_t i;
  uint32_t const mpu_cfg_rbar[5] = {
    /* SRAM and ROM Addr for code */
     (0x00000000),
    /* SRAM Addr for stack */
     (0x20002000),
    /* SRAM Addr for stack */
     (0x20000000),
    /* Private bus address */
     (0xE0000000),
    /* Peripher address*/
     (0x40000000)
  };

  uint32_t const mpu_cfg_rasr[5] = {
    /* SRAM and ROM for code */
     (MPU_DEFS_RASR_SIZE_512MB       | MPU_DEFS_NORMAL_MEMORY_WT |
     MPU_DEFS_RASE_AP_FULL_ACCESS    | MPU_RASR_ENABLE_Msk),
    /* SRAM for stack */
     (MPU_DEFS_RASR_SIZE_8KB         | MPU_DEFS_NORMAL_MEMORY_WT |
     MPU_DEFS_RASE_AP_PRIV_RW       | MPU_RASR_ENABLE_Msk),
    /* SRAM for stack */
     (MPU_DEFS_RASR_SIZE_8KB         | MPU_DEFS_NORMAL_MEMORY_WT |
     MPU_DEFS_RASE_AP_FULL_ACCESS    | MPU_RASR_ENABLE_Msk),
    /* Private bus address */
     (MPU_DEFS_RASR_SIZE_1MB         | MPU_DEFS_SHARED_DEVICE    |
     MPU_DEFS_RASE_AP_FULL_ACCESS    | MPU_RASR_ENABLE_Msk),
    /* Peripher address */
     (MPU_DEFS_RASR_SIZE_512MB       | MPU_DEFS_SHARED_DEVICE    |
     MPU_DEFS_RASE_AP_FULL_ACCESS    | MPU_RASR_ENABLE_Msk)
  };

  /* Return 1 to indicate error */
  if  (MPU->TYPE==0)
  {
    return 1;
  }

  /* Make sure outstanding transfers are done */
```

```
  __DMB();

  /* Disable the MPU */
  MPU->CTRL = 0;

  /* Configure only 4 regions */
  for (i=0; i<3; i++)
  {
    /* Select which MPU region to configure */
    MPU->RNR = i;
    /* Configure region base address register */
    MPU->RBAR = mpu_cfg_rbar[i];
    /* Configure region attribute and size register */
    MPU->RASR = mpu_cfg_rasr[i];
  }

  /* Disabled unused regions */
  for (i=4; i<8; i++)
  {
    /* Select which MPU region to configure */
    MPU->RNR = i;
    /* Configure region base address register */
    MPU->RBAR = 0;
    /* Configure region attribute and size register */
    MPU->RASR = 0;
  }

  /* Enable the MPU */
  MPU->CTRL = MPU_CTRL_ENABLE_Msk;

  /* Memory barriers to ensure subsequence data & instruction */
  __DSB();

  /* transfers using updated MPU settings */
  __ISB();

  /* No error */
  return 0;
}


int in_privileged(void)
{
  if (__get_IPSR() != 0)
    return 1;  // True
  else if ((__get_CONTROL() & 0x1)==0)
    return 1;  // True
  else
    return 0;  // False
```

```
}


int main ()
{
  /**********Initial Step***********
   *
   *  In SPC1068, the recommand basic intial step is as followed
   *   (1) System initial
   *   (2) Clock initial
   *   (3) Delay Initial
  */


  /*-------------------------------------------------------------
---
    Step 1: System initial
    Load calibration parameter from flash, please do not modify
this function.
  -------------------------------------------------------------
--*/
  Sys_Init () ;

  /*-------------------------------------------------------------
---
    Step 2: Clock initial
       This function initial CPU（HCLK) and configure all IP
clock as fast as
       HCLK. Except for ADC, PCLK and QSPI whose clock have
limit.
   Note:
    (1) SPC1068 provide RC Oscillator with 1% precision at full
temperature range
    (2) If external crystal is as clock source, please use
          CLOCK_Init（CLOCK_FROM_XTAL8MHZ, CLOCK_HCLK_75MHZ）;
       to initial clock however with larger code size.
  -------------------------------------------------------------
--*/
  CLOCK_InitWithRCO (CLOCK_HCLK_24MHZ) ;
  /* Optional function for more flexibility */
  //CLOCK_Init（CLOCK_FROM_XTAL8MHZ, CLOCK_HCLK_100MHZ）;


  /*-------------------------------------------------------------
---
    Step 3. Delay initial
    In this demo code, Delay is realized with timer2 which can
provide a precision
    delay. Clock of Timer is from PCLK.
```

```
  ------------------------------------------------------------------
--*/
  Delay_Init () ;

  /* Set GPIO function as UART  */
  GPIO_SetPinChannel (GPIO_34,  GPIO34_UART_TXD) ;
  GPIO_SetPinChannel (GPIO_35,  GPIO35_UART_RXD) ;

  /* Enable UART Clock */
  CLOCK_EnableModule (UART_MODULE) ;

  /* UART Init */
  UART_Init (UART,  38400) ;

  /* Enable MemManage exception */
  SCB->SHCSR |= SCB_SHCSR_MEMFAULTENA_Msk;

  /* Set up MPU */
  MPU_SetUp () ;

  /* Get the CONTROL register value */
  ControlVal = __get_CONTROL () ;

  /* CPU is privileged by default and we change it to unprivilege
there */
  __set_CONTROL ( (ControlVal & ~ (0x1UL) )  |
MPU_DEFS_MODE_UNPRIVILEGE) ;

  u32Privilege = in_privileged () ;
  if (u32Privilege)
    printf ("Execution is privileged\n") ;
  else
    printf ("Execution is unprivileged\n") ;

  /*
   * Access RAM addr 0x20002000~0x20004000 protected by MPU,  only
privilege can access,  unprivilege
   * access cause a hardfault.
   */
  * (uint32_t*) (0x20002004) = 0xacce55;
}

void HardFault_Handler (void)
{
  printf ("HardFault had happended\n") ;
}

void MemManage_Handler (void)
{
```

```
  printf ("MemManage had happended\n") ;
}
```

3.3 不使用 RNR 寄存器配置 MPU 示例代码

不使用 RNR 配置 MPU 示例代码

```
#include "spc1068.h"
#include <stdio.h>
uint32_t ControlVal,  u32Privilege;

int MPU_SetUp (void)
{
  uint32_t i;
  uint32_t const mpu_cfg_rbar[5] = {
    /* SRAM and ROM Addr for code and region num */
     (0x00000000 | MPU_RBAR_VALID_Msk |  (MPU_RBAR_REGION_Msk &
0) ) ,
    /* SRAM Addr for stack and region num */
     (0x20002000 | MPU_RBAR_VALID_Msk |  (MPU_RBAR_REGION_Msk &
1) ) ,
    /* SRAM Addr for stack and region num */
     (0x20000000 | MPU_RBAR_VALID_Msk |  (MPU_RBAR_REGION_Msk &
2) ) ,
    /* Private bus address and region num */
     (0xE0000000 | MPU_RBAR_VALID_Msk |  (MPU_RBAR_REGION_Msk &
3) ) ,
    /* Peripher address and region num */
     (0x40000000 | MPU_RBAR_VALID_Msk |  (MPU_RBAR_REGION_Msk &
4) )
  };

  uint32_t const mpu_cfg_rasr[5] = {
    /* SRAM and ROM for code */
     (MPU_DEFS_RASR_SIZE_512MB       | MPU_DEFS_NORMAL_MEMORY_WT |
     MPU_DEFS_RASE_AP_FULL_ACCESS    | MPU_RASR_ENABLE_Msk) ,
    /* SRAM for stack */
     (MPU_DEFS_RASR_SIZE_8KB         |
MPU_DEFS_NORMAL_SHARED_MEMORY_WT |
     MPU_DEFS_RASE_AP_PRIV_RW        | MPU_RASR_ENABLE_Msk) ,
    /* SRAM for stack */
     (MPU_DEFS_RASR_SIZE_8KB         |
MPU_DEFS_NORMAL_SHARED_MEMORY_WT |
     MPU_DEFS_RASE_AP_FULL_ACCESS    | MPU_RASR_ENABLE_Msk) ,
    /* Private bus address */
     (MPU_DEFS_RASR_SIZE_1MB         | MPU_DEFS_SHARED_DEVICE     |
     MPU_DEFS_RASE_AP_FULL_ACCESS    | MPU_RASR_ENABLE_Msk) ,
    /* Peripher address */
     (MPU_DEFS_RASR_SIZE_512MB       | MPU_DEFS_SHARED_DEVICE     |
     MPU_DEFS_RASE_AP_FULL_ACCESS    | MPU_RASR_ENABLE_Msk)
  };
```

```
  /* Return 1 to indicate error */
  if (MPU->TYPE==0)
  {
    return 1;
  }

  /* Make sure outstanding transfers are done */
  __DMB();

  /* Disable the MPU */
  MPU->CTRL = 0;

  /* Configure only 4 regions */
  for (i=0; i<5; i++)
  {
    /* Configure region base address register */
    MPU->RBAR = mpu_cfg_rbar[i];
    /* Configure region attribute and size register */
    MPU->RASR = mpu_cfg_rasr[i];
  }

  /* Disabled unused regions */
  for (i=5; i<8; i++)
  {
    /* Select which MPU region to configure */
    MPU->RNR = i;
    /* Configure region base address register */
    MPU->RBAR = 0;
    /* Configure region attribute and size register and close sub-
region*/
    MPU->RASR = 0;
  }

  /* Enable the MPU */

  MPU->CTRL = MPU_CTRL_ENABLE_Msk;

  /* Memory barriers to ensure subsequence data & instruction */
  __DSB();

  /* transfers using updated MPU settings */
  __ISB();

  /* No error */
  return 0;
}


int in_privileged(void)
{
```

```
  if (__get_IPSR() != 0)
    return 1;  // True
  else if ((__get_CONTROL() & 0x1)==0)
    return 1;  // True
  else
    return 0;  // False
}


int main()
{
  /**********Initial Step***********
   *
   *  In SPC1068, the recommand basic intial step is as followed
   *   (1) System initial
   *   (2) Clock initial
   *   (3) Delay Initial
  */


  /*-------------------------------------------------------------
---
    Step 1: System initial
    Load calibration parameter from flash, please do not modify
this function.
  -------------------------------------------------------------
--*/
  Sys_Init();

  /*-------------------------------------------------------------
---
    Step 2: Clock initial
       This function initial CPU(HCLK) and configure all IP
clock as fast as
       HCLK. Except for ADC, PCLK and QSPI whose clock have
limit.
    Note:
     (1) SPC1068 provide RC Oscillator with 1% precision at full
temperature range
     (2) If external crystal is as clock source, please use
           CLOCK_Init(CLOCK_FROM_XTAL8MHZ, CLOCK_HCLK_75MHZ);
       to initial clock however with larger code size.
  -------------------------------------------------------------
--*/
  CLOCK_InitWithRCO(CLOCK_HCLK_24MHZ);
  /* Optional function for more flexibility */
  //CLOCK_Init(CLOCK_FROM_XTAL8MHZ, CLOCK_HCLK_100MHZ);
```

```
  /*-------------------------------------------------------------
---
     Step 3. Delay initial
     In this demo code,  Delay is realized with timer2 which can
provide a precision
     delay. Clock of Timer is from PCLK.
  -------------------------------------------------------------
--*/
  Delay_Init () ;

  /* Set GPIO function as UART  */
  GPIO_SetPinChannel (GPIO_34,  GPIO34_UART_TXD) ;
  GPIO_SetPinChannel (GPIO_35,  GPIO35_UART_RXD) ;

  /* Enable UART Clock */
  CLOCK_EnableModule (UART_MODULE) ;

  /* UART Init */
  UART_Init (UART,  38400) ;

  MPU_SetUp () ;

  /* Enable MemManage exception */
  SCB->SHCSR |= SCB_SHCSR_MEMFAULTENA_Msk;

  /* Get the CONTROL register value */
  ControlVal = __get_CONTROL () ;

  /* CPU is privileged by default and we change it to unprivilege
there */
  __set_CONTROL ( (ControlVal & ~ (0x1UL) )  |
MPU_DEFS_MODE_UNPRIVILEGE) ;

  u32Privilege = in_privileged () ;
  if (u32Privilege)
    printf ("Execution is privileged\n") ;
  else
    printf ("Execution is unprivileged\n") ;

  /*
   * Access RAM addr 0x20002000~0x20004000 protected by MPU,  only
privilege can access,  unprivilege
   * access cause a hardfault.
   */
  * (uint32_t*)  (0x20002000)  = 0xacce55;
}

void HardFault_Handler (void)
{
```

```
  printf ("+++++++++HardFault had happended start+++++++\n") ;

  u32Privilege = in_privileged () ;
  if (u32Privilege)
    printf ("Execution is privileged\n") ;
  else
    printf ("Execution is unprivileged\n") ;

  printf ("+++++++++HardFault had happended end+++++++\n") ;
}

void MemManage_Handler (void)
{
  printf ("------MemManage had happended start-----\n") ;

  u32Privilege = in_privileged () ;
  if (u32Privilege)
    printf ("Execution is privileged\n") ;
  else
    printf ("Execution is unprivileged\n") ;

  printf ("------MemManage had happended end-----\n") ;
}
```

3.4 使用别名寄存器配置 MPU 示例代码

使用别名寄存器时最好配合汇编代码使用，这样才能发挥别名寄存器的作用（使用 LDM/STM 指令简化配置代码，从而加速配置过程），如果采用 C 语言进行编码，编译器不一定会将代码编译成 LDM/STM 指令。一下给出一段采用 C 语言和汇编语言混编来配置 MPU 的示例代码。

C 语言代码部分为：

使用别名寄存器配置 MPU 示例代码（C 代码部分）

```
#include "spc1068.h"
#include <stdio.h>
uint32_t ControlVal,  u32Privilege;

extern void mpu_cfg_copy (unsigned int src) ;
int i ;

/* MPU configuration table */
uint32_t const mpu_cfg_rbar_rasr[16] = {
  /* WDT1 - region 0 */
  /* RBAR */
   (0x40000000 | MPU_RBAR_VALID_Msk |  (MPU_RBAR_REGION_Msk &
0) ) ,
  /* RASR */
   (MPU_DEFS_RASR_SIZE_4KB       | MPU_DEFS_SHARED_DEVICE |
   MPU_DEFS_RASE_AP_FULL_ACCESS | MPU_RASR_ENABLE_Msk) ,
  /* SRAM - region 1 */
  /* RBAR_A1 */
   (0x20002000 | MPU_RBAR_VALID_Msk |  (MPU_RBAR_REGION_Msk &
1) ) ,
  /* RASR_A1 */
   (MPU_DEFS_RASR_SIZE_8KB   | MPU_DEFS_NORMAL_MEMORY_WT |
   MPU_DEFS_RASE_AP_PRIV_RW | MPU_RASR_ENABLE_Msk) ,
  /* GPIO D base address - region 2 */
  /* RBAR_A2 */
   (0x40003000 | MPU_RBAR_VALID_Msk |  (MPU_RBAR_REGION_Msk &
2) ) ,
  /* RASR_A2 */
   (MPU_DEFS_RASR_SIZE_1KB       | MPU_DEFS_SHARED_DEVICE |
   MPU_DEFS_RASE_AP_FULL_ACCESS | MPU_RASR_ENABLE_Msk) ,
  /* Region 3 -unused */
   (MPU_RBAR_VALID_Msk |  (MPU_RBAR_REGION_Msk & 3) ) ,  0,
  /* Region 4 -unused */
   (MPU_RBAR_VALID_Msk |  (MPU_RBAR_REGION_Msk & 4) ) ,  0,
  /* Region 5 -unused */
   (MPU_RBAR_VALID_Msk |  (MPU_RBAR_REGION_Msk & 5) ) ,  0,
  /* Region 6 -unused */
   (MPU_RBAR_VALID_Msk |  (MPU_RBAR_REGION_Msk & 6) ) ,  0,
  /* Region 7 -unused */
   (MPU_RBAR_VALID_Msk |  (MPU_RBAR_REGION_Msk & 7) ) ,  0,
```

```
};

int main ()
{
  /**********Initial Step************
   *
   *  In SPC1068, the recommand basic intial step is as followed
   *   (1) System initial
   *   (2) Clock initial
   *   (3) Delay Initial
  */


  /*-------------------------------------------------------------
---
    Step 1: System initial
    Load calibration parameter from flash, please do not modify
this function.
  -------------------------------------------------------------
--*/
  Sys_Init () ;

  /*-------------------------------------------------------------
---
    Step 2: Clock initial
       This function initial CPU（HCLK) and configure all IP
clock as fast as
       HCLK. Except for ADC, PCLK and QSPI whose clock have
limit.
   Note:
     (1) SPC1068 provide RC Oscillator with 1% precision at full
temperature range
     (2) If external crystal is as clock source, please use
           CLOCK_Init（CLOCK_FROM_XTAL8MHZ, CLOCK_HCLK_75MHZ）;
       to initial clock however with larger code size.
  -------------------------------------------------------------
--*/
  CLOCK_InitWithRCO (CLOCK_HCLK_24MHZ) ;
  /* Optional function for more flexibility */
  //CLOCK_Init（CLOCK_FROM_XTAL8MHZ, CLOCK_HCLK_100MHZ）;


  /*-------------------------------------------------------------
---
    Step 3. Delay initial
    In this demo code, Delay is realized with timer2 which can
provide a precision
    delay. Clock of Timer is from PCLK.
```

```
  ------------------------------------------------------------
--*/
  Delay_Init () ;

  /* Set GPIO function as UART  */
  GPIO_SetPinChannel (GPIO_34,  GPIO34_UART_TXD) ;
  GPIO_SetPinChannel (GPIO_35,  GPIO35_UART_RXD) ;

  /* Enable UART Clock */
  CLOCK_EnableModule (UART_MODULE) ;

  /* UART Init */
  UART_Init (UART,  38400) ;

  /* Enable MemManage exception */
  SCB->SHCSR |= SCB_SHCSR_MEMFAULTENA_Msk;

  /* Set up MPU */
  for (i = 0;  i < sizeof (mpu_cfg_rbar_rasr) ;  i++)
    mpu_cfg_copy ( (unsigned int)  &mpu_cfg_rbar_rasr[i]) ;

  /*
   * Access RAM addr 0x20002000~0x20004000 protected by MPU,  only
privilege can access,  unprivilege
   * access cause a hardfault.
   */
  * (uint32_t*)  (0x20002004)  = 0xacce55;
}

void HardFault_Handler (void)
{
  printf ("HardFault had happended\n") ;
}

void MemManage_Handler (void)
{
  printf ("MemManage had happended\n") ;
}
```

汇编代码部分为（必须使用 ARM 汇编器，否则编译会出错）：

使用别名寄存器配置 MPU 示例代码（汇编代码部分）

```
  AREA MY_FUNCTION, CODE, READONLY
  EXPORT mpu_cfg_copy

mpu_cfg_copy
  PUSH {R4-R9}
  LDR R1,  =0xE000ED9C ; MPU->RBAR address
```

```
  LDR R2, [R1, #-12]    ; Get MPU->TYPE
  CMP R1, #0          ; If zero
  ITT EQ ; If-Then
  MOVSEQ R0, #1 ; return 1
  BEQ mpu_cfg_copy_end
  DMB 0xF ; Make sure outstanding transfers are done
  MOVS R2, #0
  STR R2, [R1, #-8] ; MPU->CTRL = 0
  LDMIA R0!, {R2-R9} ; Read 8 words from table (base update)
  STMIA R1, {R2-R9} ; Write 8 words to MPU (no base update
  LDMIA R0!, {R2-R9} ; Read 8 words from table (base update)
  STMIA R1, {R2-R9} ; Write 8 words to MPU (no base update)
  DSB 0xF ; Memory barriers to ensure subsequence data &
instruction
  ISB 0xF ; transfers using updated MPU settings
  MOVS R0, #0 ; No error
mpu_cfg_copy_end
  POP {R4-R9}
  BX LR
  ALIGN 4
  END
```

注意：在本文的描述中，汇编器为 ARM 汇编器，GUN 汇编器所对应的汇编代码超出本文的描述范文，所以不做过多讨论。

4 MPU 使用时的考虑因素

在使用 MPU 时需要考虑多方面的因素，举例来说我们使用 MPU Region 时，通常需要考虑到工程代码大小，所需内存大小，外设属性等因素。特别的，当我们使用 OS 时，我们还应当考虑所使用的 OS 是否支持 MPU。