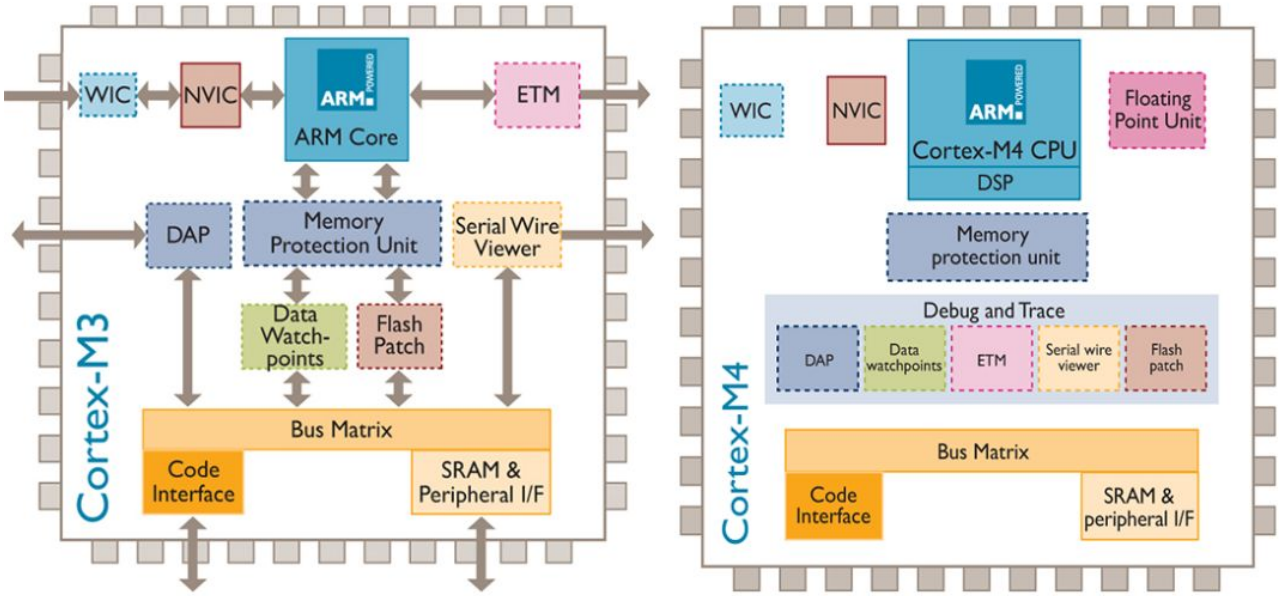
讲讲Cortex-M内核的MPU内存保护单元



估计大家经常看见MCU、MPU、MMU等这类缩写词，你们了解MPU吗？

1写在前面

不知道大家有没有关注过Cortex-M内核的一些内容，在STM32大部分型号中都有MPU。

MPU是Cortex-M的选配件，拿STM32F1来说，STM32F10X_XL系列的芯片才具有这个MPU存储保护单元，而其他STM32F1芯片没有。

Key Features

- Core: ARM® 32-bit Cortex®-M3 CPU with MPU
  - 72 MHz maximum frequency, 1.25 DMIPS/MHz (Dhrystone 2.1) performance at 0 wait state memory access
  - Single-cycle multiplication and hardware division
- Memories
  - 768 Kbytes to 1 Mbyte of Flash memory
  - 96 Kbytes of SRAM
  - Flexible static memory controller with 4 Chip Select. Supports Compact Flash, SRAM, PSRAM, NOR and NA
  - LCD parallel interface, 8080/6800 modes
- Clock, reset and supply management

可能很多人都处于简单知道，或认识MPU的阶段，今天就写点关于MPU的内容，让大家进一步认识和了解MPU。

2

认识MPU，及作用

MPU：Memory Protection Unit，内存保护单元。

MPU存储器保护单元，它可以实施对存储器（主要是内存和外设寄存器）的保护，以使软件更加健壮和可靠。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有如下的能力**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。
- 此外，还可以通过MPU设置存储器regions的其它访问属性，比如，是否缓区，是否缓冲等。

3

了解野指针

上面简单认识了一下MPU的功能，其实它有个重要的功能就是对指针访问的内存具有保护作用。所以，这里让大家认识一下指针和野指针。

> 回顾一下， 什么是指针？指针在内存中实际上是一个无符号整数（unsigned int），但是它的值被赋予特殊的解释：表示变量或函数的地址。所以才被形象地称为“指针”，就好像指向谁家似的。 在使用指针前， 都必须先让它指向有意义的， 并且允许由程序使用的实体——数据和代码。 而所谓“野指针”， 就是指某个指针变量的值因故超出合法的范围， 使其“枪口”乱指。 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条件， 甚至是恶意地破坏等， 都可以制造出野指针。 如果使用野指针去读取或修改内存， 则被读取或修改的位置是不可预料的。 前者导致读回来的都是乱掉的数据， 后者则会破坏未知用途的数据。 这常常导致系统发生莫名其妙的功能紊乱， 严重时会使系统毫无征兆，没有理由地失控、死机。
>
> 野指针就像“肉里的刺， 酱里的蛆”一般： 一个野指针就足以毁掉整个系统， 而且极其隐蔽， 很难通过症状来找出是哪里存在野指针， 甚至都不能判定症状是否因野指针造成（程序大了其它 bug 也很多，并且也能导致相同的症状）。对于通常的单片机系统，是没有任何办法来防止野指针的破坏的， 完全靠程序员的素质和自律。 但智者千虑，必有一失。 尤其是当程序规模变得很大时，复杂度会呈指数上升，千头万绪纠缠不清， 就算是谨慎如诸葛亮，聪明如比尔·盖茨的天才，也不敢保证没有漏网之鱼。

---来自CM3内核翻译作者

4

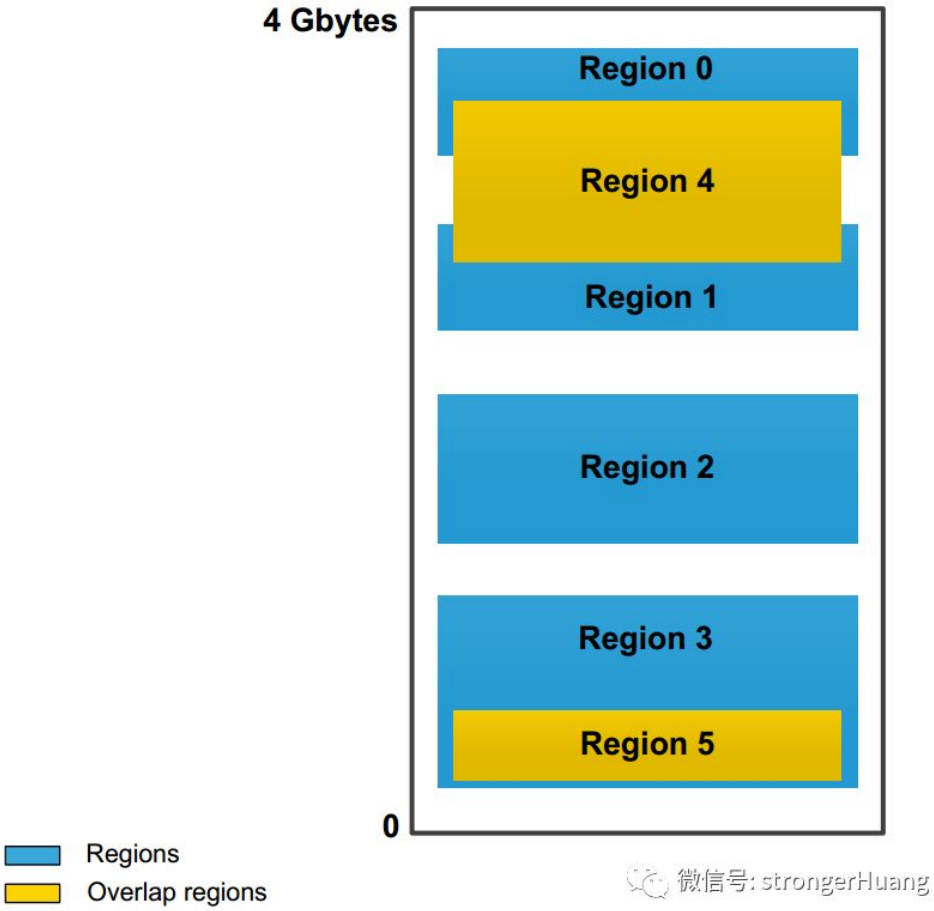
进一步了解MPU

MPU在执行其功能时，是以所谓的“region区域”为单位的。**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（对齐方式，最小容量等）**。

CM3的MPU共支持8个regions，还允许把每个region进一步划分成更小的“子region”。此外，还允许启用一个“背景region”（即没有MPU时的全部地址空间），不过它是只能由特权级享用。在启用MPU后，就不得再访问定义之外的地址区间，也不得访问未经授权的region。否则，将以“访问违例”处理，触发MemManage fault。

MPU定义的regions可以相互交迭。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，则交迭的部分受4号region控制。

MPU可用于保护多达16个内存区域。如果区域至少为256字节，那么这些区域可以有8个子区域。子区域的大小总是相等的，可以通过子区域号启用或禁用。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256字节大小。

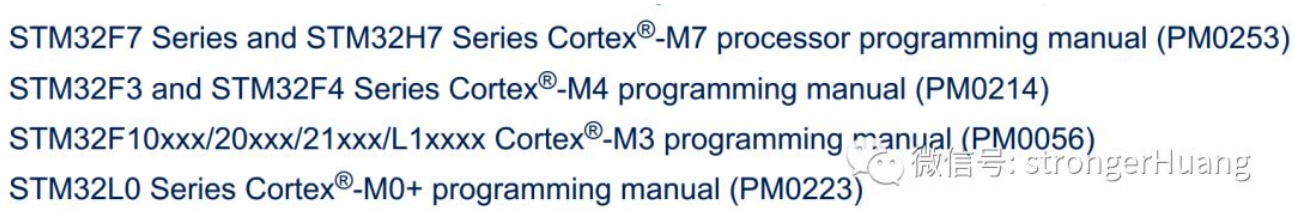


5

MPU学习资料

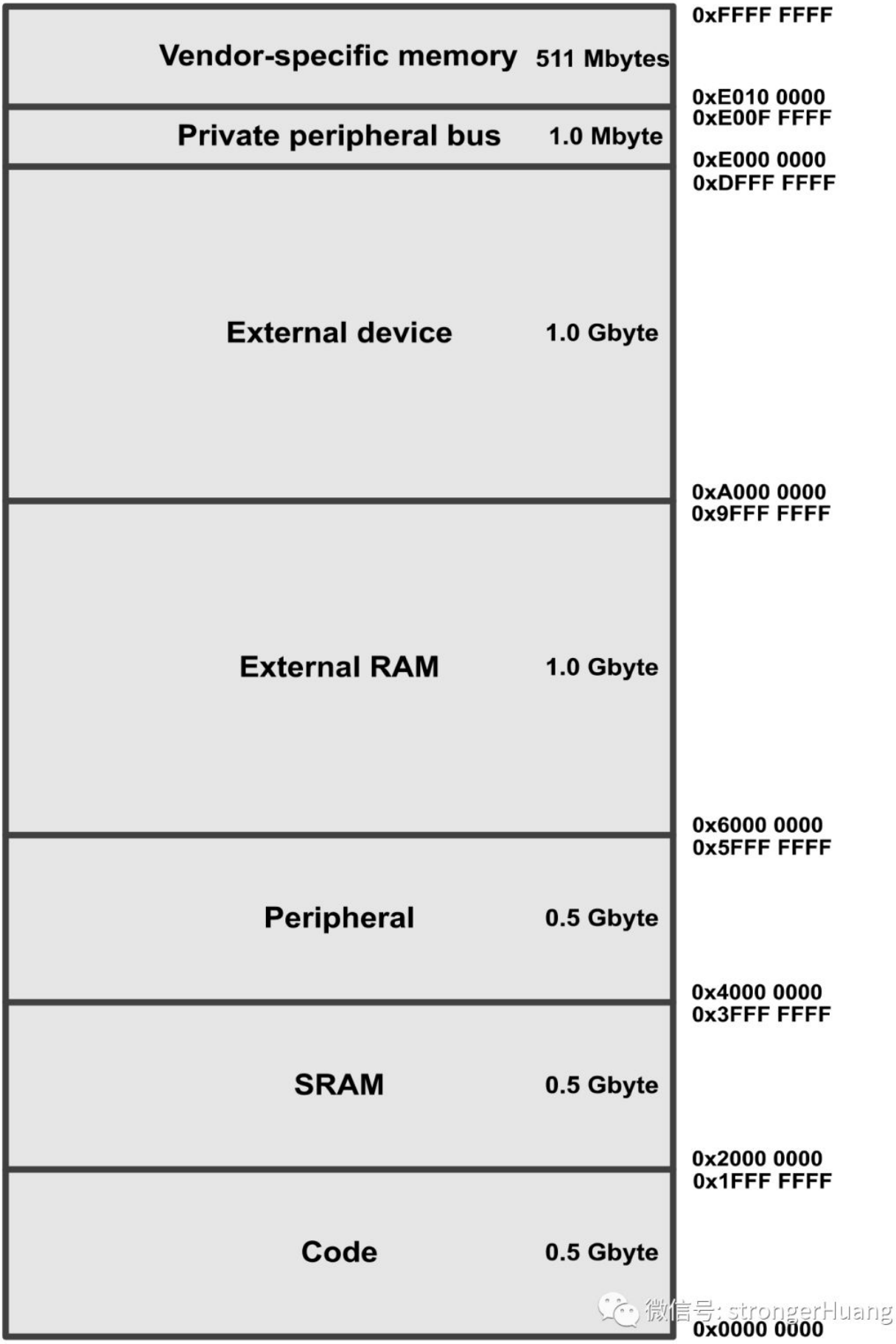
上面只是进一步让大家了解了MPU内存保护单元，对于想要深入理解的朋友就需要参看更多相关资料。

对学习MPU编程，就需要对MPU相关寄存器进行掌握，MPU的寄存器其实相对来说也不多，这里再Cotex-M内核技术参考手册，以及STM32应用笔记Managing memory protection unit (MPU) in STM32 MCUs、编程手册中都有讲述关于MPU的知识。



为方便大家，这里也简单说几点。

1.STM32内存映射



2.MPU 的寄存器组

操作MPU是就如操作普通STM32外设一样，通过访问它的若干寄存器来实现的，MPU寄存器如下表所示。

名字	访问	地址	初值
MPU类型寄存器 MPUTR	RO	0xe000,ed90	A
MPU控制寄存器 MPUCR	RW	0xe000,ed94	0x0000,0000
MPU region号寄存器MPURNR	RW	0xe000,ed98	-
MPU region基址寄存器MPURBAR	RW	0xe000,ed9c	-
MPU region属性及容量寄存器(s) MPURASR	RW	0xed00,eda0	-
MPU region基址寄存器的别名1	D9C的别名	0xed00,eda4	-
MPU region属性及容量寄存器的别名1	DA0的别名	0xed00,eda8	-
MPU region基址寄存器的别名2	D9C的别名	0xed00,edac	-
MPU region属性及容量寄存器的别名2	DA0的别名	0xed00,edb0	-
MPU region基址寄存器的别名3	D9C的别名	0xed00,edb4	-
MPU region属性及容量寄存器的别名3	DA0的别名	0xed00,edb8	-

MPU寄存器看起来比较复杂，那是自然了，毕竟已经上升到存储器管理的高度。但如果我们胸有成竹——已经想好了对存储器如何划分，这就只是一些繁琐和考验细心的体力活。典型情况下，在启用MPU的系统中，都会有下列的regions。

特权级的程序代码（如OS内核和异常服务例程）

用户级的程序代码

特权级程序的数据存储器，位于代码区中（data_stack）

用户级程序的数据存储器，位于代码区中（data_stack）

通用的数据存储器，位于其它存储器区域中（如， SRAM）

系统设备区，只允许特权级访问，如NVIC和MPU的寄存器所有的地址区间

常规外设区，如UART， ADC等

3.Cube HAL配置MPU例子

```
void MPU_RegionConfig(void)
{
  MPU_Region_InitTypeDef MPU_InitStruct;
  /* Disable MPU */
  HAL_MPU_Disable();
  /* Configure RAM region as Region N°0, 8kB of size and R/W region */
  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(&amp;MPU_InitStruct);
  /* Configure FLASH region as REGION N°1, 1MB of size and R/W region */
  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(&amp;MPU_InitStruct);
  /* Configure FMC region as REGION N°2, 0.5GB of size, R/W region */
  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(&amp;MPU_InitStruct);
  /* Enable MPU */
  HAL_MPU_Enable(MPU_PRIVILEGED_DEFAULT);
}
```