针对ARM Cortex M3平台的代码移植

作　者:
ARM中国

ARM Cortex™-M3是一款高性能、低成本、低功耗的32位RISC处理器。它仅执行16位、32位混合的Thumb-2指令，不支持ARM指令集。Cortex-M3处理器集成了一个 ARM v7-M架构的高效哈佛3级流水线ARM内核，支持硬件除法器和快速ISR（中断服务程序）响应。除CPU内核外，Cortex-M3处理器还包括许多其他组件，嵌套向量中断控制器（NVIC）、可选的存储器保护单元（MPU）、计时器、调试访问端口（DAP）以及可选的嵌入式跟踪宏单元（ETM）。同时，Cortex-M3具有固定的存储器映射分配。

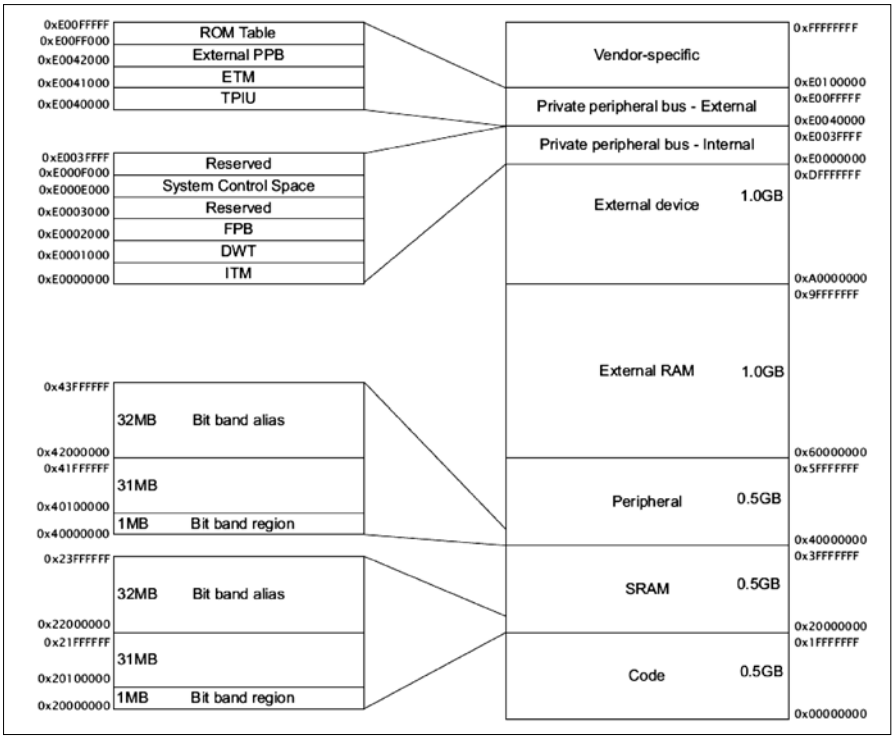
嵌套向量中断控制器（NVIC）

NVIC 支持多达240个外部中断（具体中断数目可由IC vendor决定），256个不同的优先级别，这些优先级别可动态地重新排列优先顺序。它支持电平和脉冲中断源。处理状态会在中断进入时由硬件自动保存，并在中断退出时恢复。同时，NVIC对于末尾连锁（tail-chaining）中断有独特的处理方式，将中断响应时间减到最小。

在Cortex-M3中使用NVIC意味着Cortex-M3的向量表与之前的ARM内核有着显著的区别。与大多数其他ARM内核不同，Cortex-M3向量表中包含异常处理程序和ISR的地址，而不是指令。复位处理程序的初始堆栈指针和地址必须分别位于0x0和0x4。这些值在随后的复位中被加载到适当的CPU寄存器中。

存储器保护单元（MPU）

MPU是Cortex-M3的选配组件。它



▲ 图1　Cortex-M3存储器分配图

可通过执行特权和访问规则为保护存储器的区域提供支持。它支持多达8个不同的区域(region),这些区域又可以分为8个子区域,每个子区域占一个区域大小的八分之一。

调试访问端口(DAP)

调试访问端口使用AHB-AP接口与处理器以及其他外设通信。调试端口支持两种不同的实现:串行JTAG调试端口(SWJ-DP)和串行调试端口(SW-DP)。IP Vendor可以根据需求在这两种调试方式中进行选择。

存储器映射

与之前的大多数ARM内核不同,Cortex-M3处理器拥有固定的存储器映射(Memory Map)。这方便了软件在基于Cortex-M3的不同系统之间轻松移植。Cortex M3地址空间被分为许多不同的段,这在图1和表1中有所说明。

存储区域	说明	通过总线访问
代码	用于代码存储器(flash、ROM 或重新映射的内存)	ICode和DCode
静态随机存取存储器(SRAM)	用于片上 SRAM,带位元处理操作功能	系统
外设	用于普通外设,带位元处理操作功能	系统
外部内存	用于外部存储器	系统
外部设备	外设的存储空间,或共用存储器	系统
私人外设	系统设备的地址空间,如MPU、NVIC、DAP和其他 CoreSight 设备	系统
特定于经销商	用于经销商指定的其他用途	

▲ 表1 Cortex-M3存储器分配图详情

与任何将项目移植到新目标平台的过程一样,用户将代码移植到Cortex M3平台时,通常是从项目的最小版本出发,并逐步建立所有功能模块。您可以使用注释或预处理程序宏来移除暂时并不需要的程序功能段,从而轻松实现这一点。这样还可以在稍后的移植过程中重新逐个、轻松的引入功能,实现对每个与具体平台相关程序功能的彻底测试。

1 一般代码的修改

大多数不依赖硬件平台的代码段通常无需修改即可在Cortex- M3上正确工作。然而,您可能需要修改代码中的某些功能并为新的功能目标进行更新。

1.1 对C代码的修改

将项目从ARM7内核迁移至Cortex-M3时,您必须通过适当的--cpu编译选项将所有的 C 代码重新编译为Thumb-2,这包括任何第三方库。

传统的遗留工程中的Thumb-1代码与Cortex-M3 兼容,完全可以在新的处理器CortexM3上运行。但是,在RVCT 3.0中,连接程序可能会意外地将对象文件与包含一些 ARM 指令的库连接。所以ARM建议用户在移植代码到CortexM3上时,尽量将所有代码重新编译为Thumb-2。如果您需要使用传统的对象或库,则必须手动检查连接映象,并确认其中未包含 ARM 指令。如果您具有针对ARM7的第三方库,则可能需要联系供应商以获取第三方库的Thumb-2版本。

用户可能还需要对源代码本身进行一些小小的更改。很多遗留工程中包含用于更改处理器状态的编译指示("#pragma arm"和"#pragma thumb"),用户必须将其删除。由于嵌入式汇编代码(Inline assembly)不能够被编译为Thumb-2代码,因此用户在移植过程中必须将其改写为C、C++或者嵌入汇编(embedded assembly)代码。

1.2 对汇编代码的修改

移植汇编代码时应特别小心。

用户必须将ARM或CODE32等ARM指令的汇编指示删除,或者将其替换为为THUMB指示。

在汇编程序命令行中插入正确的--cpu汇编选项之后,汇编器可以轻松的处理大多数现有的汇编代码。然而,某些ARM指令集特有的,不受Thumb-2指令集支持的罕见指令可能仍然需要修改。

很多遗留代码中存在CODE16指示,当用户在移植过程中将器直接替换为 THUMB时,汇编器会在不警告的情况下进行汇编,但在行为上CODE16与THUMB可能存在细微的差别。这是因为CODE16根据传统的Thumb-1句法规则进行汇编。例如,在CODE16句法规则下,许多不带后缀S的指令将编码为标记设置变量;而在THUMB指示的句法规则下,必须明确指定后缀S。

汇编程序将根据需要为条件执行的指令插入IT指令。例如,单独的"ADDSNE r0, r0, r1"指令(后面跟着无条件的指令)会变成:"IT NE",后面跟着"ADDS r0, r0, r1"

移植过程中,用户应该留意针对内核和架构一些特殊指令,这些指令包括:

- 协处理器指令
- 对PSR的状态或模式更改及其他访问

- SWP指令，已由LDREX和STREX代替
- 某些寻址模式不受LDM和STM指令的支持
- Thumb–2对某些指令有附加的限制，如带有寄存器偏移量和立即数移位的 LDR /STR 指令。

您可能需要改写代码中的某些部分以适应Cortex–M3编程模型。否则，当发现不兼容问题时，汇编器会产生警告或错误。Cortex–M3的大多数状态和控制寄存器都是存储器映射的，其支持的模式与ARM7TDMI有着显著的区别；遗留工程中任何更改状态或模式的代码都必须修改或者移除。同样，访问协处理器的代码也必须移除。

同时请注意在地址运算中使用PC的值的代码。由于Thumb–2使用混合的16位和32位指令，当用于数据处理操作时，PC的值始终是当前指令的地址加上4。

2 对启动代码的修改

启动代码包含应用程序的复位处理程序，以及在应用程序的主体运行之前设置环境和外设的任何初始化功能。它针对特定的内核和目标。

如果系统比较简单，用户可以在中断向量表中将C库函数进入点（__main()函数 ）指定为复位处理程序，并在用户代码中从main()功能执行附加的初始化就足够了。但是，如果有外设需要关键的初始化，则可能需要编写简短的汇编代码功能以当作初始的复位处理程序，然后再使用 __main()。同时请注意，访问某些设备（如MPU）的代码在写入这些寄存器后可能需要一个或多个内存栅指令，以便确保更改立即生效。如果您的项目之前针对支持MPU或MMU的平台，则必须修改相关的代码。

对于所有的Cortex–M3工程，您必须按CortexM3的固定格式创建新的异常向量表，并分别在0x0和0x4添加复位处理程序的初始堆栈指针和地址。

3 异常处理代码

当移植旧代码到Cortex–M3时，异常处理程序必须修改。

您通常不需要以汇编语言编写的低级别处理程序，因为重新进入是由内核进行处理。如果您的低级别处理程序执行附加工作，则可能需要将部分程序划分为单独的功能，以便由新的处理程序进行调用。请记住使用__irq关键字标记您的IRQ处理程序，以确保编译器可以保持 Cortex–M3 revision 0硬件的堆栈对齐。

Cortex–M3没有专门的FIQ输入。任何在ARM7TDMI项目上产生FIQ中断的外设都必须被设置成高优先级的向量中断（IRQ），或Cortex–M3的NMI信号。用户需要修改代码，以避免这些中断的处理程序使用FIQ banked registers，因为CortexM3中，这些中断只是正常的 IRQ 中断，在它们的处理程序中，这些寄存器目前需要压栈以保存现场。

最后，您必须编写一个新的初始化功能以配置包括中断优先级的 NVIC，以便在输入主要应用程序代码之前启用中断。

在Cortex–M3上，异常优先排序、异常的嵌套和易腐寄存器的保存全部由内核处理，从而提供非常高效的中断处理并最大程度降低中断延迟。这意味着在异常处理程序过程中，中断一直是使能的。此外，如果在中断从异常返回时禁用中断，处理器不会自动将其重新启用。内核无法执行中断的自动启用并从异常返回。如果您在处理程序中临时禁用中断，则首先必须重新启用中断，然后使用一个用于返回的单独指令。因此异常可能会紧接在异常返回之前发生。

根据系统设计，异常模型的这些功能可能会对代码中的临界段造成影响。在临界段的执行期间，需要禁用中断，以便临界段作为不间断的块来执行，如操作系统中的上下文开关程序代码。某个传统代码可能会假设将在中断进入异常处理时将其禁用，并仅在任何临界段完成后由代码明确地将其启用。这些假设在Cortex–M3的新异常模型下不成立，此类代码需改写以考虑这一点。

4 利用 Cortex–M3 的新功能

当原始的项目重新针对新的平台后，您可能需要修改项目以利用Cortex–M3提供的新功能。如果您将代码重新编译为Thumb–2，架构v7–M中可用的大多数新指令都将自动使用。但Cortex–M3中的某些功能可能需要用户手动更改代码。尤其您可能需要：

- 修改初始化代码以启用MPU。
- 利用内核提供的睡眠模式以降低功耗。
- 利用位元处理（bit–banding）操作以提高位元修改的性能（如果有任何数据或外设位于相应的位元处理区域）。