第18章

# ARM7应用程序移植到Cortex-M3

* 简介
* 系统个性
* 汇编源程序
* C源程序
* 预编译的目标文件
* 优化

## 18.1 简介

如果非要找出CM3的降临可以带来的痛苦，也许就是把运行在ARM7TDMI上的代码升级过来所要做的工作了，这种成长的阵痛也是在所难免的。为了降低升级难度专门开出本章，把升级过程中的重点明确地总结一下。

在计划把代码从ARM7移植到CM3时，需要考虑以下的方面：

* + 系统性质
  + 汇编源程序
  + C源程序
  + 优化

总体来说，越是底层的代码，受到的冲击越大。像最底层的硬件控制、任务管理以及异常服务例程，它们与架构的关系最密切。另一方面，因为底层的代码往往大面积地使用汇编，因此面临改写甚至重写的工作量最大。普通的应用程序需要的改动则比较小，而且这时优良的编程习惯经常会大幅度，甚至戏剧般地降低修改工作量（最简单的就是多使用宏定义）。对于与架构无关的纯算法类应用程序，则都无需改动，只要简单地重新编译即可。

## 18.2 系统的个性

想必大家也已经总结出来了，CM3与ARM7相比，还是有很多新的个性的。像固定的存储器映射，中断处理机制，操作模式，系统控制，以及新引入了MPU等。下面我们就一一小结。

### 18.2.1 存储器映射

在不同处理器架构间的差异中，存储器映射算得上是最“外向”型的了。在ARM7中，是由器件厂商自由划分4GB的寻址空间的，再加上厂商还可能玩各种“二次映射”技术，各ARM芯片之间的存储器映射可以是大相径庭的。到了CM3中，把存储器映射被粗线条地标准化了——把4GB空间分成了若干个不同类型的区域，对应的存储器必须对号入座。一般地，通过设置编译和连接选项，可以轻易地适应新的ROM和RAM的映射图。但对于设备驱动程序，则情况比较复杂。如果是不同厂家的芯片，外设寄存器的用法基本上是完全不同的，此时驱动程序必须重写；如果是在同一厂家的ARM7和CM3芯片间移植，则外设寄存器有望相对一致，驱动程序只需部分改动，甚至简单到只修改基地址即可。

许多ARM7芯片会提供存储器的“二次映射”功能，其中一个重要的用途，就是使向量表可以被重映射到SRAM中。而在CM3中，可以通过编程NVIC的寄存器来实现此功能，因此不再需要这些二次映射功能，从而许多芯片可能也去掉了完备的二次映射支持（但是可能会提供一种“硬件控制”的二次映射——上电时，由某些管脚的电平决定把哪里的存储器映射到零地址上，以支持多种引导方式。如STM32就采用了此法，以支持从 Flash/SRAM/原配BootLoader引导——译者注）。

CM3对大端模式的支持方式也与ARM7的不一样。程序代码只需重新编译，但是事先做好的查找表则需要重新编码。（大端编码是多事之地，建议读者少碰它——译者注）。

从ARM720T，以及ARM9等那个年代开始的处理器，为了支持像WinCE这样的操作系统，引入了所谓的“高端向量”功能——允许把向量表重定位到0xFFFF\_0000。CM3并没有打算支持WinCE（实际上最重要的原因是没有配MMU），因此去掉了“高端向量”的支持。

### 18.2.2 中断/异常系统

可能NVIC都快引起大家的审美疲劳了。没错，在CM3中的中断处理已经被彻底改造，因此所有与控制中断有关的代码都需要大面积更新。而且还需要为建立中断优先级和向量表添加全新的代码。

中断返回机制也变了。这影响到了汇编代码。而且如果编译器使用指示字(directive)来支持C程序中断服务程序的话，还需要调整指示字。

过去，对中断的使能和除能是通过修改CPSR的，在CM3中没有CPSR，而是使用PRIMASK或FAULTMASK来实现全局中断的开关。

CM3在响应中断时，启用了一个自动栈操作的机制，因此可以把旧时的入栈和出栈指令化简。然而，旧时的ARM还有所谓的FIQ，并且为FIQ服务例程专开了小灶——独立的4个寄存器（R8-R11），专为FIQ服务例程使用，无需push/pop。FIQ其实极少利用，成了“彩色糖衣包装却没营养的药药”。在CM3中并没有FIQ的概念，因此在移植以前的FIQ服务例程时，在代码上必须把它当作普通的中断服务例程处理——其实因为CM3有自动堆栈操作，普通中断也相当于享有FIQ的小灶待遇。另一方面，通过提升其优先级到最高，可以使它在时间上得到FIQ的待遇。

实现嵌套中断的代码现在可以去掉了，因为CM3的NVIC已经内部实现了中断嵌套。

错误处理机制也大有不同。旧时的ARM只有DAbt, IAbt, Undef这3种异常模式对应错误处理，而到了CM3中，提供了很多fault状态寄存器来确定各种faults，而且还定义了许多新的fault类型，其中最有新意的就是堆栈操作faults、存储器管理faults以及硬fault了。因此，fault服务例程需要重新设计。

### 18.2.3 MPU

MPU是CM3中的新鲜血液，因此需要新的程序代码来使用它。另一方面，因为在ARM7TDMI中没有MPU，因此这方面没有“代码移植”的概念。不过，在ARM720T上是配有MMU的，它的功能与CM3的MPU不一样——事实上，如果代码需要MMU来支持虚拟内存，根本就不能使用CM3。

### 18.2.4 系统控制

系统控制也是移植程序时必须充分重视的关键内容。CM3内建了进入睡眠模式的指令。另一方面，在CM3芯片中的系统控制器也有特殊的设计要求，基本上它们不会与ARM7芯片中的有什么相似之处。因此，要做好思想准备，来重写系统控制相关的代码。

### 18.2.5 操作模式

以前的ARM架构有7种操作模式，在CM3中，它们可以用对应的异常来取代，如表18.1所示：

表8.1 把ARM7TDMI中的操作模式和异常映射到CM3

|  |  |
| --- | --- |
| 在ARM7中的操作模式和异常 | 在CM3中与之等价的操作模式和异常 |
| 监察者(supervisor)（复位后自动进入） | 特权级的线程模式+MSP |
| 监察者（因SWI而进入） | SVC异常 |
| FIQ | 优先级最高的外部中断 |
| IRQ | 外部中断 |
| 指令流产(IAbt) | 总线fault |
| 数据流产(DAbt) | 总线fault |
| 未定义指令 | 用法fault |
| 系统模式 | 特权级的线程模式+PSP |
| 用户 | 用户级的线程模式+PSP |

虽然在CM3中，可以把ARM7的FIQ对应到优先级最高的外中断，从而实现FIQ的时间地位。但是ARM7的“专用寄存器”是R8-R11，而CM3自动入栈的寄存器是R0-R3,R12。因此，旧时FIQ服务例程需要改用R0-R3,R12；如果依然要使用R8-R11，就必须先把它们手工入栈。

**NMI vs. FIQ**

可能有不少人曾想到过用NMI来取代FIQ。的确，在一些场合中，这是可行的。但是，NMI与FIQ有本质的区别，使得很多情况下它们不能互换，这也是我们必须清醒地认识到的。

1. NMI正如其名，是不能被除能的。而ARM7的FIQ则可以通过把CPSR.F置位来除能。因此，在CM3中何时进入NMI完全不可控——有可能在引导期间就进入NMI。而在ARM7中，复位后FIQ是除能的，因此不会意外地进入。
2. CM3的NMI服务例程不得使用SVC，而ARM7的FIQ服务例程则可以使用SWI。另外，在ARM7下，即使是在FIQ服务例程的执行过程中，也可以转而响应其它异常（IRQ除外）。而在CM3下，如果NMI服务例程执行过程中发生fault，则处理器当即被锁定。

## 18.3 汇编源程序

对汇编源程序的移植取决于使用的是ARM状态还是Thumb状态。

### 18.3.1 Thumb状态

如果使用的是thumb汇编源文件，则是幸运的，在大多数情况下代码可以直接拿来用。只有个别的thumb指令在CM3中不可用：

* + 任何试图转入ARM状态的指令（典型就是BLX）
  + 不再支持SWI，而是要使用SVC，而且用法上也有区别

最后，一定要只使用向下生长的满栈，CM3的push和pop就是使用这种模型的——总有程序喜欢玩另类，结果不但移植工作量加重了，也无法使用C语言了。因此可别在这里秀“叛逆精神”啊，否则会自找苦吃的，而且向下生长的满栈本来就是更合理的。

### 18.3.2 ARM状态

如果不幸在汇编源文件中使用了ARM状态，也不要慌，仔细核对下列情况：

* 向量表：在ARM7中，向量表从0地址开始，并且由一系列的跳转指令组成。在CM3中，跳转表给出了MSP的初值以及复位向量地址，接下来的则是各异常服务例程的入口地址。因此这些区别是本质上的不同，向量表必须重写。
* 寄存器初始化：在ARM7中，经常需要把每个模式下的寄存器分别初始化。比如，每个模式（除系统模式外）都有自己的SP、LR和SPSR。CM3去掉了这些繁文缛节，而且也不再需要把处理器的模式换来换去。
* 模式切换与状态切换：在CM3不再保留ARM7中的那些操作模式，也没有Thumb状态，因此相关的代码都可以移除。
* 中断的使能与除能：在ARM7中，中断的使能与除能是通过对CPSR.I来控制的。在CM3中则改用PRIMASK或FAULTMASK。更进一步地，CM3中没有FIQ的概念，因此也没有F位。
* 协处理器访问：CM3不支持协处理器，因此相关的代码无法移植。但是可以通过软件模拟的办法来缓解
* 中断服务例程和中断返回：在ARM7中，中断服务例程的首条指令在向量表中。这条指令，除了FIQ服务例程的外，都必须是一种无条件跳转指令，而CM3中则是直接在向量表中给出ISR的入口地址。中断返回时，ARM7是通过带S后缀的指令手工地调整PC的值来实现；而CM3则把需要返回的地址压入堆栈中，并且通过把某个EXC\_RETURN写入PC来触发中断返回序列。因此，在CM3中，不得使用诸如MOVS或SUBS之类的指令来启动中断返回。由于这些原因，中断服务例程和中断返回的代码需要加以改动。
* 当需要启用中断嵌套时，ARM7的作法通常是先进入系统模式再重新使能IRQ，在CM3中则没有这些操作。
* FIQ服务例程：因为在ARM7中，FIQ有专用的R8-R12；而CM3则自动保存了R0-R3，R12。所以如果必须要移植FIQ服务例程，则需要手工保存R8-R11。或者把本来对R8-R11的使用，改为以R0-R3的使用。
* 软件中断(SWI)服务例程：SWI由SVC取代。不过，定位软件中断指令并萃取系统调用号的作法不同。在CM3中，通过压入栈的返回地址来计算出SVC指令的地址；而在ARM7中，则是通过LR来计算。
* 交换指令（SWP）：在CM3中没有交换指令。如果以前使用SWP来实现信号量，则要改为使用互斥访问来实现，因此需要改动信号量相关的代码。如果以前使用SWP只是为了纯粹地传送数据，则需要使用若干存储器访问指令来实现。
* 对CPSR和SPSR的访问：ARM7中的CPSR在CM3变成了xPSR，而SPSR则被去掉了。对于访问标志的应用程序代码，可以改为对APSR的访问。如果异常服务例程想要访问异常发生之前的xPSR，则要读取压入堆栈中的值——这取代了ARM7中SPSR的功能，因此CM3中不再需要SPSR。
* 条件执行：在ARM7中，大量指令都可以条件执行；而Thumb-2的指令则几乎都不能条件执行。在移植这些代码到CM3中时，对于短小的条件执行段，可以用IF-THEN指令封装；而比较大的则需要使用跳转指令来改建。当使用IT指令时要注意一些小问题。主要就是会增加代码量，有可能使得某些加载/存储指令超出最大可操作的地址范围。
* 使用PC计算当前代码的地址：在ARM7中，读取的PC值“读PC指令的地址+8”。这是由ARM7的三级流水线造成的——当读取PC的指令处于执行阶段时，PC已经自增了两次。同样的事情也发生在CM3中，但是在代码移植到CM3后，因为这些代码将在Thumb下执行，所以PC被加的值变成4。
* 对R13的使用：R13总是32位的。但是在CM3中，末2位被强制为0。因此，如果偶尔遇到使用R13作为基址的场合（强烈反对使用），必须更改代码，因为末2位的信息已经丢失了。

## 18.4 C源程序

好歹也是高级语言，移植C源程序要比移植汇编的轻松很多。在许多情况下，只消重新编译即可。但是对于使用了非主流技巧的C程序（常见于系统程序中），则可能要考虑如下的方面。

* + 内联汇编：如果使用RVDS，则不支持内联汇编，因此使用了内联汇编的C程序需要做出修改。对于RVDS 3.0及更高版本，可以使用嵌入式汇编来替代内联汇编。
  + 中断服务例程：对于使用“\_\_irq”来创建的ARM7中断服务例程，因为CM3使用了新的中断模型，往往可以去掉“\_\_irq”指示字（不过，如果使用RVDS 3.0和RVCT 3.0，则\_\_irq也支持CM3，此时可以保留“\_\_irq”，以强调程序的类型，提高了可读性）。

## 18.5 预编译的目标文件

许多编译器都为函数库和启动代码预先编译出了目标文件。但是因为操作模式和状态模型的不同，它们往往不能用在CM3上——尤其是启动代码。此时，就必须得到它们的源代码，并且移植到CM3上，请参阅你所使用工具链的联机帮助来获取详细信息（事实上，推荐使用的开发工具（KeilMDK/GCC）都已经做好了这些事情——译者注）。

## 18.6 优化

CM3中有许多新特性，加以利用的话常常可以大大提高程序的性能，或者降低对存储器的使用。对于积极向上的我们，一定要挖掘这些特性：

* + 使用32位Thumb-2指令：对于下列的场合：先使用一条16位thumb指令把数据从一个寄存器传送到另一个，再对该数据执行数据处理。有时能使用一条Thumb-2指令来完成（这主要是因为16位Thumb指令不能使用“高寄存器”——译者注），从而使所需的处理时间缩短。
  + 位带操作：如果外设寄存器位于位带区，则可以通过对位带别名区的访问，大大地化简对寄存器位的操作。
  + 乘法与除法：CM3的一个重大革新就是支持除法指令和部分支持64位乘法指令。请善用它们（尤其是除法），可以成十上百倍地提高程序的执行速度。
  + 立即数：有些Thumb-2指令支持长达12位的立即数，因此可以把以前Thumb指令无法加载的立即数使用一条Thumb-2来加载。
  + 跳转：过去单条Thumb指令无法执行的远程跳转，现在可以使用Thumb-2指令实现了。
  + 布尔数据：对于“BOOL”型的变量，可以强制把它们定址到内存的位带别名区。相比于过去使用字来实现BOOL变量，现在只需使用以前1/32的内存空间。
  + IT指令块：有些短距跳转可以使用IT指令取代，这样做消灭了因流水线清洗而引入的等待周期，从而提高了性能。
  + ARM/Thumb状态切换：在大多情况下，可以把大部分代码以Thumb指令编码，一小部分以ARM指令编码。这主要是为了在平时提高代码密度，而在紧急关头下提高性能。在CM3下有了Thumb-2代码，可以在同一模式下解决时间与空间的权衡。这就可以去掉这些状态转换及其所带来的额外负担（overhead），也简化了对工程的管理。