目录

第1章 简介

基于ARM嵌入式处理器的片上系统解决方案可应用于企业应用、汽车系统，家庭网络和无线技术等市场领域。ARM Cortex™系列提供了一个标准的体系结构来满足以上各种技术的不同性能要求，其包含的处理器基于ARMv7 架构的三个分工明确的部分。A部分面向复杂的尖端应用程序，用于运行开放式的复杂操作系统；R部分针对实时系统；M部分为成本控制和微控制器应用提供优化。Cortex-M3 是首款基于ARMv7-M架构的处理器，是专门为了在微控制器，汽车车身系统，工业控制系统和无线网络等对功耗和成本敏感的嵌入式应用领域实现高系统性能而设计的，它大大简化了可编程的复杂性，使ARM架构成为各种应用方案（即使是最简单的方案）的上佳选择。

1.1 通过提高效率来提高性能

处理器可通过两种途径来提高它的性能，一是“work hard”，也就是直接通过提高时钟频率来提高性能，这种情况以高功耗作为代价，并增加了设计的复杂性。另一种是“work smart”，在低时钟频率的情况下提高运算效率，使处理器可以凭借简单的低功耗设计来完成与第一种情况同样的功能。Cortex-M3 处理器的核心是基于哈佛架构的 3 级流水线内核，该内核集成了分支预测，单周期乘法，硬件除法等众多功能强大的特性，使其在Dhrystone benchmark上具有出色的表现（1.25 DMIPS/MHz）。根据Dhrystone benchmark的测评结果，采用新的Thumb®-2 指令集架构的Cortex-M3 处理器，与执行Thumb指令的ARM7TDMI-S®处理器相比，每兆赫的效率提高了 70%，与执行ARM指令的ARM7TDMI-S处理器相比，效率提高了 35%。

1.2 快速有效的应用程序开发源于简易的使用方法

缩短上市时间与降低开发成本是选择微控制器的关键标准，而快速和简易的软件开发能力是实现这些要求的关键。Cortex-M3 处理器专门针对快速和简单的编程而设计，用户无需深厚的架构知识或编写任何汇编代码就可以建立简单的应用程序。Cortex-M3 处理器带有一个简化的基于栈的编程模型，该模型与传统的 ARM 架构兼容，同时与传统的 8 位、16 位架构所使用的系统相似，它简化了 8 位、16 位到 32 位的转换过程。此外，使用基于硬件的中断机制意味着编写中断服务程序（handlers）不再重要。在不需要汇编代码寄存器操作的情况下，启动代码得到了大大的简化。

在位字段处理、硬件除法和 If/Then 指令的协助下，Thumb-2 指令集架构（Instruction Set Architecture－ISA）底层的关键特性使 C 代码的执行变得更加自然。在开发方面，Thumb-2 指令自动优化了性能和代码密度，在无需交互使用 ARM 代码和 Thumb 代码的情况下加快了开发的进程，简化了编译目标的长期维护和支持工作。如此一来，用户不但可以继续使用 C 代码，而且还免去了建立预编译目标代码库的麻烦，代码在更大程度上获得了重复利用。

1.3 针对敏感市场降低成本和功耗

成本是采用高性能微控制器永恒的屏障。由于先进的制造工艺相当昂贵，只有降低芯片的尺寸才有可能从本质上降低成本。为了减小系统区域，Cortex-M3 处理器采用了至今为止最小的 ARM 内核，该内核的核心部分（0.18um G）的门数仅为 33000 个，它把紧密相连的

系统部件有效地结合在一起。通过采用非对齐数据存储技术、原子位操作和 Thumb-2 指令集，存储容量的需求得到最小化，其中 Thumb-2 指令集对指令存储容量的要求比 ARM 指令减少超过 25%。

为了迎合对节能要求日益增长的大型家电和无线网络市场，Cortex-M3 处理器支持扩展时钟门控和内置睡眠模式。当采用ARM Metro™标准单元库和TSMC 0.13G制造工艺时，处理器运行在 50MHz的目标频率下的功耗仅为 4.5mW，芯片封装面积只有 0.33mm^2。

1.4 集成的调试和跟踪功能推进上市的步伐

嵌入式系统通常不具备图形用户界面，软件调试也因此成了程序员的一大难题。传统上，在线仿真器（ICE）单元作为插件使用，通过大家熟悉的 PC 界面向系统提供窗口。然而，随着系统体积的变小及其复杂性的增加，物理附加类似的调试单元已经再难成为可行的方案。Cortex-M3 处理器通过其集成部件在硬件的本身实现了各种调试技术，使调试在具备跟踪和分析、断点、观察点和代码修补功能的同时，速度也获得了有效的提高，促使产品可以更快地投入市场。此外，处理器还通过一个传统的 JTAG 端口或一个适用于低管脚数封装（LPC 封装）器件的 2 管脚串行线调试（Serial Wire Debug－SWD）端口赋予系统高度的可视性。

1.5 从ARM7™升级为Cortex-M3 可获取更佳的性能和功效

在过去十年中，ARM7 系列处理器被广泛应用于众多领域。之后，Cortex-M3 在 ARM7 的基础上开发成功，为基于 ARM7 处理器系统的升级开辟了通道。它的中心内核效率更高，编程模型更简单，它具有出色的确定中断行为，其集成外设以低成本提供了更强大的性能。

表 1-1 ARM7TDMI-S 和 Cortex-M3 的比较（采用 100MHz 频率和 TSMC 0.18G 制程）

特性	ARM7TDMI-S	Cortex-M3
架构	ARMv4T（冯·诺依曼）	ARMv7-M（哈佛）
ISA 支持	Thumb/ARM	Thumb / Thumb-2
流水线	3 级	3 级+分支预测
中断	FIQ / IRQ	NMI + 1 到 240 个物理中断
中断延迟	24－42 个时钟周期	12 个时钟周期
休眠模式	无	内置
存储器保护	无	8 段存储器保护单元
Dhrystone	0.95 DMIPS/MHz（ARM 模式）	1.25 DMIPS/MHz
功耗	0.28mW/MHz	0.19mW/MHz
面积	0.62mm2（仅内核）	0.86mm2（内核+外设）*

*不包含可选系统外设（MPU 和 ETM）或者集成的部件

图 1-1 ARM7TDMI-S (ARM) 和 Cortex-M3 (Thumb-2)的性能对比

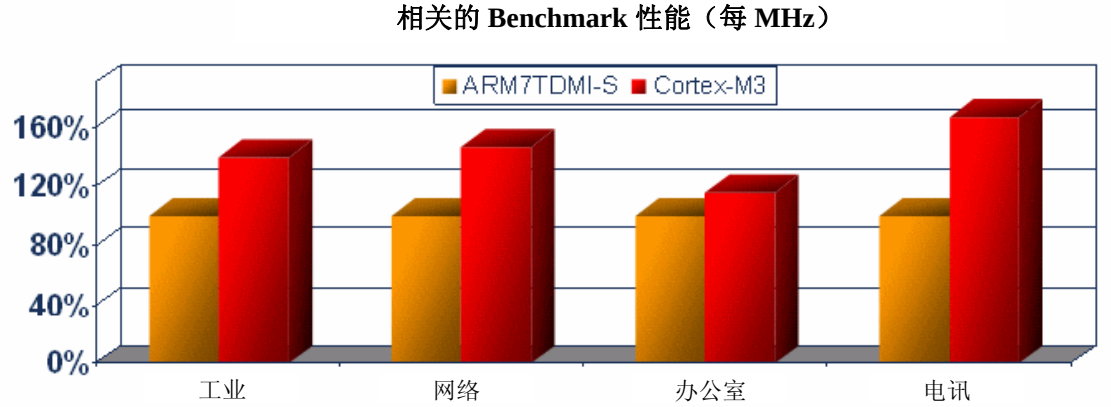
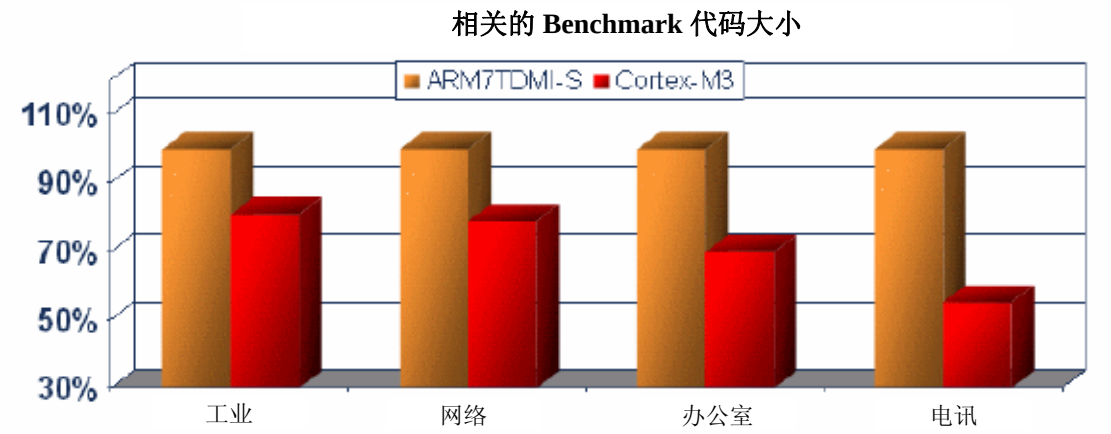


图 1-2 ARM7TDMI-S (ARM) 和 Cortex-M3 (Thumb-2)的代码大小



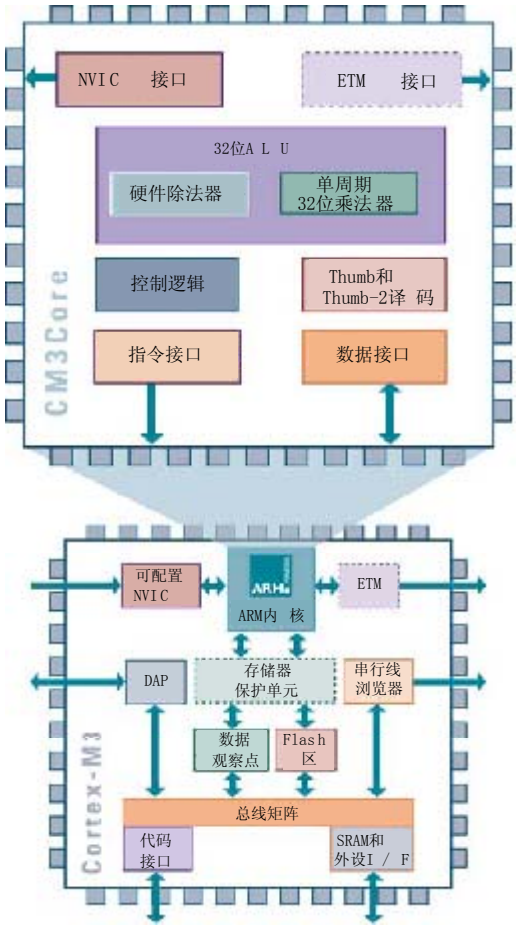
第2章 Cortex-M3 处理器的架构和特性

基于 ARMv7 架构的 Cortex-M3 处理器带有一个分级结构。它集成了名为 CM3Core 的中心处理器内核和先进的系统外设，实现了内置的中断控制、存储器保护以及系统的调试和跟踪功能。这些外设可进行高度配置，允许 Cortex-M3 处理器处理大范围的应用并更贴近系统的需求。目前 Cortex-M3 内核和集成部件（图 2-1）已进行了专门的设计，用于实现最小存储容量、减少管脚数目和降低功耗。

2.1 Cortex-M3 内核

Cortex-M3 中央内核基于哈佛架构，指令和数据各使用一条总线（图 2-1）。与 Cortex-M3 不同，ARM7 系列处理器使用冯•诺依曼（Von Neumann）架构，指令和数据共用信号总线以及存储器。由于指令和数据可以从存储器中同时读取，所以 Cortex-M3 处理器对多个操作并行执行，加快了应用程序的执行速度。

图 2-1 Cortex-M3 处理器



内核流水线分 3 个阶段：取指、译码和执行。当遇到分支指令时，译码阶段也包含预测的指令取指，这提高了执行的速度。处理器在译码阶段期间自行对分支目的地指令进行取指。在稍后的执行过程中，处理完分支指令后便知道下一条要执行的指令。如果分支不跳转，那么紧跟着的下一条指令随时可供使用。如果分支跳转，那么在跳转的同时分支指令可供使用，空闲时间限制为一个周期。

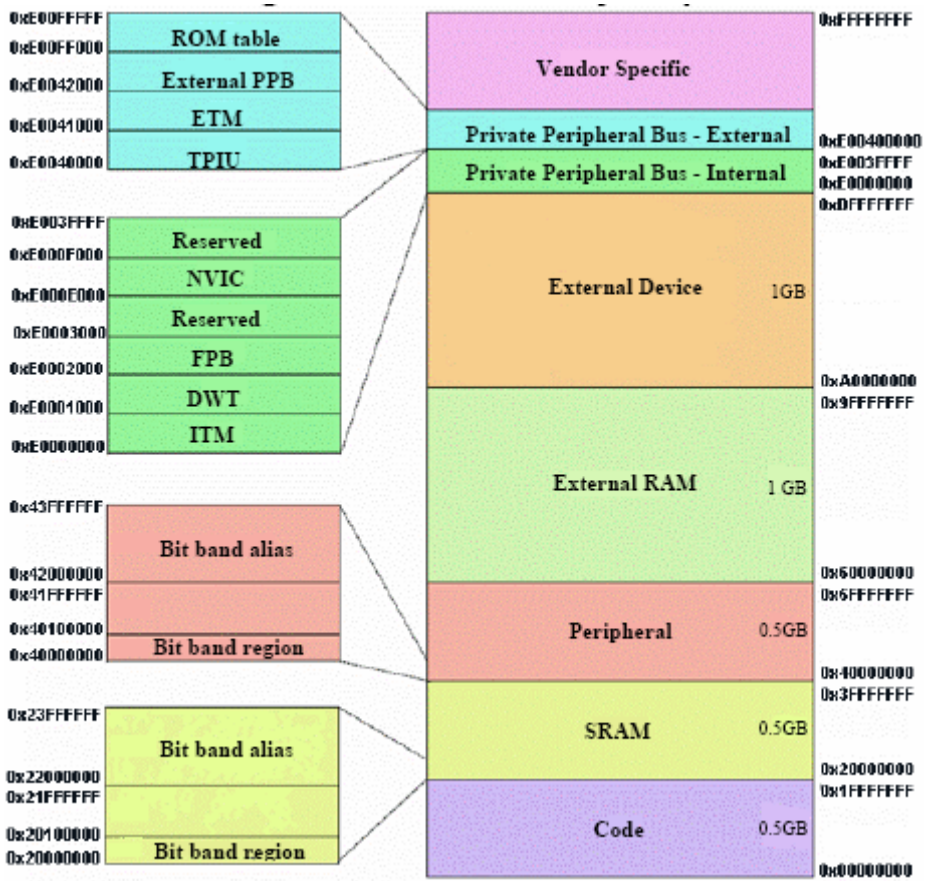
Cortex-M3 内核包含一个适用于传统 Thumb 和新型 Thumb-2 指令的译码器、一个支持硬件乘法和硬件除法的先进 ALU、控制逻辑和用于连接处理器其他部件的接口。

Cortex-M3 处理器是一个 32 位处理器，带有 32 位宽的数据路径，寄存器库和存储器接口。其中有 13 个通用寄存器，两个堆栈指针，一个链接寄存器，一个程序计数器和一系列包含编程状态寄存器的特殊寄存器。

Cortex-M3 处理器支持两种工作模式（线程（Thread）和处理器（Handler））和两个等级的访问形式（有特权或无特权），在不牺牲应用程序安全的前提下实现了对复杂的开放式系统的执行。无特权代码的执行限制或拒绝对某些资源的访问，如某个指令或指定的存储器位置。Thread 是常用的工作模式，它同时支持享有特权的代码以及没有特权的代码。当异常发生时，进入 Handler 模式，在该模式中所有代码都享有特权。此外，所有操作均根据以下两种工作状态进行分类，Thumb 代表常规执行操作，Debug 代表调试操作。

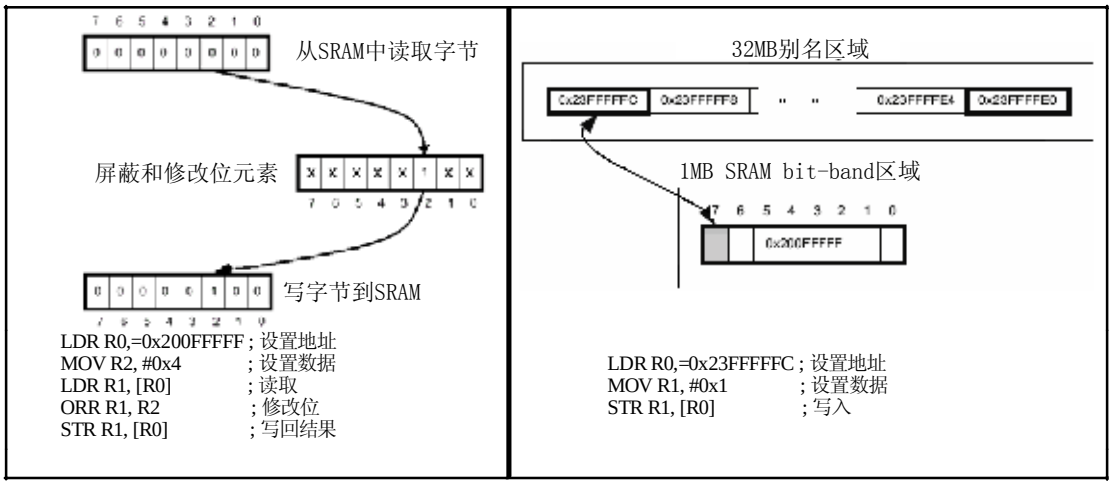
Cortex-M3 处理器是一个存储器映射系统，为高达 4GB 的可寻址存储空间提供简单和固定的存储器映射，同时，这些空间为代码（代码空间）、SRAM（存储空间），外部存储器/器件和内部/外部外设提供预定义的专用地址。另外，还有一个特殊区域专门供厂家使用。

图 2-2 存储器映射



借助bit-banding技术（图 2-3），Cortex-M3 处理器可以在简单系统中直接对数据的单个位进行访问。存储器映射包含两个位于SRAM的大小均为1MB的bit-band区域和映射到32MB别名区域的外设空间。在别名区域中，某个地址上的加载/存储操作将直接转化为对被该地址别名的位的操作。对别名区域中的某个地址进行写操作，如果使其最低有效位置位，那么bit-band位为 1，如果使其最低有效位清零，那么bit-band位为零。读别名后的地址将直接返回适当的bit-band位中的值。除此之外，该操作为原子位操作，其他总线活动不能对其中断。

图 2-3 传统的位处理方法和 Cortex-M3 bit-banding 的比较



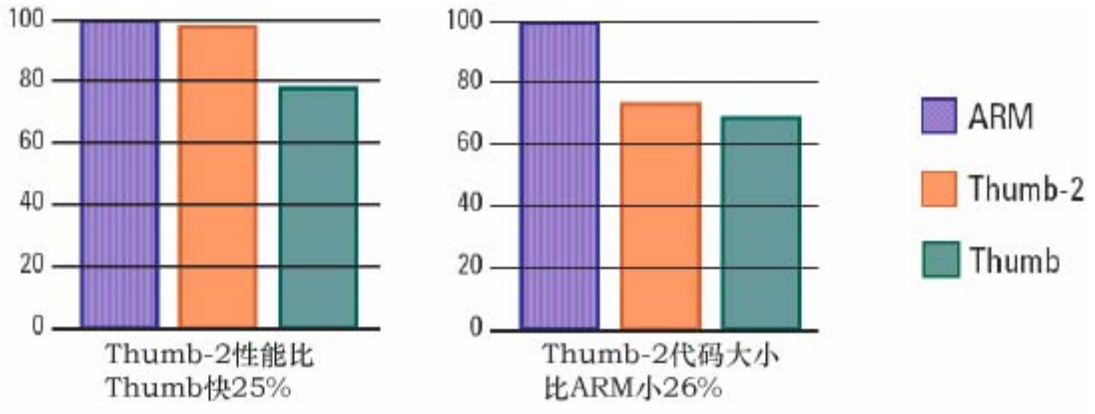
基于传统 ARM7 处理器的系统只支持访问对齐的数据，只有沿着对齐的字边界才可以对数据进行访问和存储。Cortex-M3 处理器采用非对齐数据访问方式，使非对齐数据可以在单核访问中进行传输。当使用非对齐传输时，这些传输将转换为多个对齐传输，但这一过程不为程序员所见。

Cortex-M3 处理器除了支持单周期 32 位乘法操作之外，还支持带符号的和不带符号的除法操作，这些操作使用 SDIV 和 UDIV 指令，根据操作数大小的不同在 2 到 12 个周期内完成。如果被除数和除数大小接近，那么除法操作可以更快地完成。Cortex-M3 处理器凭借着这些在数学能力方面的改进，成为了众多高数字处理强度应用（如传感器读取和取值或硬件在环仿真系统）所亲昧的理想选择。

2.2 Thumb-2 指令集架构

ARMv7-M是ARMv7 架构的微控制器部分，它和早期的ARM架构不同，它在早期的ARM架构中只单独支持Thumb-2 指令。Thumb-2 技术是 16 位和 32 位指令的结合，实现了 32 位ARM指令性能，匹配原始的 16 位Thumb指令集并与之后向兼容。图 2-4显示了预测的Dhrystone benchmark结果，由结果可见，Thumb-2 技术确实达到了预期的目标。

图 2-4 与 ARM、Thumb 以及 Thum-2 相关的 Dhrystone 性能和代码大小



在基于 ARM7 处理器的系统中，处理器内核会根据特定的应用切换到 Thumb 状态（以获取高代码密度）或 ARM 状态（以获取出色的性能）。然而，在 Cortex-M3 处理器中无需交互使用指令，16 位指令和 32 位指令共存于同一模式，复杂性大幅下降，代码密度和性能

均得到提高。由于 Thumb-2 指令是 16 位 Thumb 指令的扩展集，所以 Cortex-M3 处理器可以执行之前所写的任何 Thumb 代码。得益于 Thumb-2 指令，Cortex-M3 处理器同时兼容于其他 ARM Cortex 处理器的家族成员。

Thumb-2 指令集用于多种不同应用，使紧凑代码的编写更加简单快捷。BFI 和 BFC 指令为位字段指令，在网络信息包处理等应用中可大派用场。SBFX 和 UBFX 指令改进了从寄存器插入或提取多个位的能力，这一能力在汽车应用中的表现相当出色。RBIT 指令的作用是将一个字中的位反转，在 DFT 等 DSP 运算法则的应用中非常有用。表分支指令 TBB 和 TBH 用于平衡高性能和代码的紧凑性。Thumb-2 指令集还引入了一个新的 If-Then 结构，意味着可以有多达 4 个后续指令进行条件执行。

2.3 嵌套向量中断控制器（NVIC）

NVIC 是 Cortex-M3 处理器中一个完整的部分，它可以进行高度配置，为处理器提供出色的中断处理能力。在 NVIC 的标准执行中，它提供了一个非屏蔽中断（NMI）和 32 个通用物理中断，这些中断带有 8 级的抢占优先权。NVIC 可以通过综合选择配置为 1 到 240 个物理中断中的任何一个，并带有多达 256 个优先级。

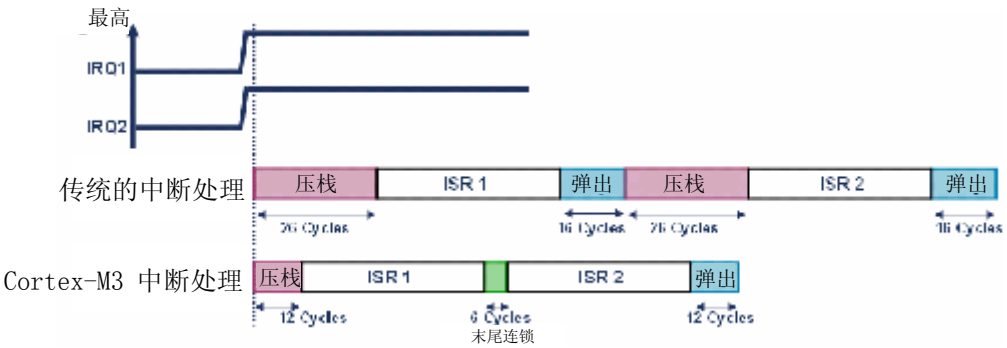
Cortex-M3 处理器使用一个可以重复定位的向量表，表中包含了将要执行的函数的地址，可供具体的中断处理器使用。中断被接受之后，处理器通过指令总线接口从向量表中获取地址。向量表复位时指向零，编程控制寄存器可以使向量表重新定位。

为了减少门计数并提高系统的灵活性，Cortex-M3 已从 ARM7 处理器的分组映像寄存器异常模型升级到了基于堆栈的异常模型。当异常发生时，编程计数器、编程状态寄存器、链接寄存器和 R0－R3、R12 等通用寄存器将被压进堆栈。在数据总线对寄存器压栈的同时，指令总线从向量表中识别出异常向量，并获取异常代码的第一条指令。一旦压栈和取指完成，中断服务程序或故障处理程序就开始执行，随后寄存器自动恢复，中断了的程序也因此恢复正常的执行。由于可以在硬件中处理堆栈操作，Cortex-M3 处理器免去了在传统的 C 语言中断服务程序中为了完成堆栈处理所要编写的汇编程序包，这使应用程序的开发变得更加简单。

NVIC 支持中断嵌套（压栈），允许通过提高中断的优先级对中断进行提前处理。它还支持中断的动态优先权重置。优先权级别可以在运行期间通过软件进行修改。正在处理的中断会防止被进一步激活，直到中断服务程序完成，所以在改变它们的优先级的同时，也避免了意外重新进入中断的风险。

在背对背中断情况中，传统的系统将重复状态保存和状态恢复的过程两次，导致了延迟的增加。Cortex-M3 处理器使用末尾连锁（tail-chaining）技术简化了激活的和未决的中断之间的移动。末尾连锁技术把需要用时 30 个时钟周期才能完成的连续的堆栈弹出和压入操作替换为 6 个周期就能完成的指令取指，实现了延迟的降低。处理器状态在进入中断时自动保存，在中断退出时自动恢复，比软件执行用时更少，大大提高了频率为 100MHz 的子系统的性能。

图 2-5 NVIC 中的末尾连锁(Tail chaining)技术



NVIC 还采用了支持内置睡眠模式的 Cortex-M3 处理器的电源管理方案。立即睡眠模式（Sleep Now mode）被等待中断（WFI）或等待事件（WFE）其中一个指令调用，这些指令可以使内核立即进入低功耗模式，异常被挂起。退出时睡眠（Sleep On Exit）模式在系统退出最低优先级的中断服务程序时使其进入低功耗模式。内核保持睡眠状态直到遇上另一个异常。由于只有一个中断可以退出该模式，所以系统状态不会被恢复。系统控制寄存器中的 SLEEPDEEP 位如果被置位，那么该位可以用来控制内核以及其他系统部件，以获得最理想的节电方案。

NVIC 还集成了一个递减计数的 24 位系统嘀哒（SysTick）定时器，它定时产生中断，提供理想的时钟来驱动实时操作系统或其他预定的任务。

2.4 存储器保护单元（MPU）

MPU 是 Cortex-M3 处理器中一个可选的部分，它通过保护用户应用程序中操作系统所使用的重要数据，分离处理任务（禁止访问各自的数据），禁止访问存储器区域，将存储器区域定义为只读，以及对有可能破坏系统的未知的存储器访问进行检测等手段来改善嵌入式系统的可靠性。

MPU 使应用程序可以拆分为多个进程。每个进程不仅有指定的存储器（代码、数据、栈和堆）和器件，而且还可以访问共享的存储器和器件。MPU 还会增强用户和特权访问规则。这包括以正确的优先级别执行代码以及加强享有特权的代码和用户代码对存储器和器件的使用权的控制。

MPU 将存储器分成不同的区域，并通过防止无授权的访问对存储器实施保护。MPU 支持多达 8 个区域，每个区域又可以分为 8 个子区域。所支持的区域大小从 32 字节开始，以 2 为倍数递增，最大可达到 4GB 可寻址空间。每个区域都对应一个区域号码（从 0 开始的索引），用于对区域进行寻址。另外，也可以为享有特权的地址定义一个默认的背景存储器映射。对未在 MPU 区域中定义的或在区域设置中被禁止的存储器位置进行访问将会导致存储器管理故障（Memory Management Fault）异常的产生。

区域的保护是根据规则来执行的，这些规则以处理的类型（读、写和执行）和执行访问的代码的优先级为基础进行制定。每个区域都包含一组位影响访问的允许类型，以及一组位影响所允许的总线操作。MPU 还支持重叠的区域（覆盖同一地址的区域）。由于区域大小是乘 2 所得的结果，所以重叠意味着一个区域有可能完全包含在另一个区域里面。因此，有可能出现多个区域包含在单个区域中以及嵌套重叠的情况。当寻址重叠区域中的位置时，返回的将是拥有最高区域号码的区域。

2.5 调试和跟踪

对 Cortex-M3 处理器系统的调试访问是通过调试访问端口（Debug Access Port）来实现的。该端口可以作为串行线调试端口（SW-DP）（构成一个两脚（时钟和数据）接口）或串行线 JTAG 调试端口（SWJ-DP）（使能 JTAG 或 SW 协议）使用。SWJ-DP 在上电复位时默认为 JTAG 模式，并且可以通过外部调试硬件所提供的控制序列进行协议的切换。

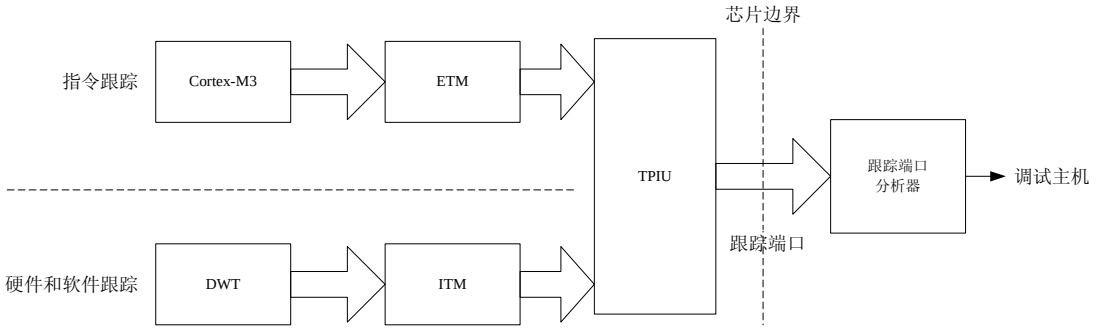
调试操作可以通过断点、观察点、出错条件或外部调试请求等各种事件进行触发。当调试事件发生时，Cortex-M3 处理器可以进入挂起模式或者调试监控模式。在挂起模式期间，处理器将完全停止程序的执行。挂起模式支持单步操作。中断可以暂停，也可以在单步运行期间进行调用，如果对其屏蔽，外部中断将在逐步运行期间被忽略。在调试监控模式中，处理器通过执行异常处理程序来完成各种调试任务，同时允许享有更高优先权的异常发生。该模式同样支持单步操作。

Flash 块和断点（FPB）单元执行 6 个程序断点和两个常量数据取指断点，或者执行块操作指令或位于代码存储空间和系统存储空间之间的常量数据。该单元包含 6 个指令比较器，用于匹配代码空间的指令取指。通过向处理器返回一个断点指令，每个比较器都可以把代码重新映射到系统空间的某个区域或执行一个硬件断点。这个单元还包含两个常量比较器，用于匹配从代码空间加载的常量以及将代码重新映射到系统空间的某一个区域。

数据观察点和跟踪（DWT）单元包含 4 个比较器，每一个比较器都可以配置为硬件观察点。当比较器配置为观察点使用时，它即可以比较数据地址，也可以比较编程计数器。DWT 比较器还可以配置用来触发 PC 采样事件和数据地址采样事件，以及通过配置使嵌入式跟踪宏单元（ETM）发出指令跟踪流中的触发数据包。

ETM 是设计用于单独支持指令跟踪的可选部件，其作用是确保在对区域的影响最小的情况下实现程序执行的重建。ETM 使指令的跟踪具有高性能和实时性，数据通过压缩处理器内核的跟踪信息进行传输可以最小化带宽的需求。

图 2-6 Cortex-M3 跟踪系统



Cortex-M3 处理器采用带 DWT 和 ITM（测量跟踪宏单元）的数据跟踪技术。DWT 提供指令执行统计并产生观察点事件来调用调试或触发指定系统事件上的 ETM。ITM 是由应用程序驱动的跟踪资源，支持跟踪 OS 和应用程序事件的 printf 类型调试。它接受 DWT 的硬件跟踪数据包以及处理器内核的软件跟踪激励，并使用时间戳来发送诊断系统信息。跟踪端口接口单元（Trace Port Interface Unit-TPIU）接收 ETM 和 ITM 的跟踪信息，然后将其合并、格式化并通过串行线浏览器（Serial Wire Viewer-SWV）发送到外部跟踪分析器单元。通过单管脚导出数据流，SWV 支持简单和具有成本效率的系统事件压型。曼切斯特编码和 UART 都是 SWV 支持的格式。

2.6 总线矩阵和接口

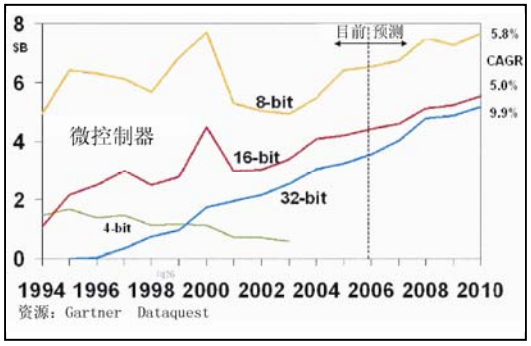
Cortex-M3 处理器总线矩阵把处理器和调试接口连接到外部总线；也就是把基于 32 位AMBA® AHB-Lite的ICode、DCode和系统接口连接到基于 32 位AMBA APB™的专用外设总线（Private Peripheral Bus-PPB）。总线矩阵也采用非对齐数据访问方式以及bit banding技术。

32 位 ICode 接口用于获取代码空间中的指令，只有 CM3Core 可以对其访问。所有取指的宽度都是一个字，每个字里面的指令数目取决于所执行代码的类型及其在存储器中的对齐方式。32 位 DCode 接口用于访问代码存储空间中的数据，CM3Core 和 DAP 都可以对其访问。32 位系统接口分别获取和访问系统存储空间中的指令和数据，与 DCode 相似，可以被 CM3Core 和 DAP 访问。PPB 可以访问 Cortex-M3 处理器系统外部的部件。

第3章 下一代 MCU 以 8 位 MCU 的成本提供 32 位的性能

微控制器单元（MCU）是一块高度集成的单芯片，它集成了处理器、非易失性存储器（用于编程）、易失性存储器（用于存储数据）、I/O 端口、定时器和多种外设，它的应用领域相当宽广，从低端的通用应用延伸至高性能的数字信号控制。另外，面对激烈的市场竞争，是否能在提供性能优越、外设齐备的产品来满足广大用户群的同时，保持低廉的成本，使产品得以推广和应用已成为最大的挑战。

图 3-1 32 位 MCU 市场的增长



MCU 市场被 8 位和 16 位 MCU 长期占据，为用户提供低成本的解决方案，例如，购买 8 位的 MCU 只需 40 美分。随着下一代应用方案对性能以及功能要求的提高，MCU 市场正向 32 位架构加速前进，届时将以极具竞争力的系统成本推出性能更高、使用更简单的 32 位 MCU。

由于许多 MCU 都实现了工业标准的接口和外设，因此在大多数情况下，器件核心部分的处理器将成为区分 MCU 产品的最重要标准。此外，ARM7 和 ARM9™处理器系列已经在 MCU 领域取得了巨大的成功，再通过解读 Cortex-M3 处理器，想必用户将确信从 8 位、16 位升级到 32 位架构是明智的决定。

由低位升级到 32 位架构的主要动机之一是获得更出色的性能。不过性能是相对而言的，例如，高端媒体处理为了满足带宽的要求，它所需的是非常高的系统时钟频率。而在 MCU 中，评估性能的标准是 flash 存储器指令的快速执行、有效位的操作、廉价的调试和跟踪技术、以及强大的中断结构（该中断对于实时应用来说至关重要）。Cortex-M3 处理器把结构特性和内置的外设部件结合在一起，为 MCU 设计者提供了低成本和高性能的解决方案，在简化开发过程的同时大大缩短了产品投放市场的时间。

3.1 有效使用内存可降低成本

事实上，在所有嵌入式设计的材料单中，存储器都占据了主导的地位，有效使用代码存储器和数据存储器成为降低成本的关键。Cortex-M3 处理器的 Thumb-2 技术和总线接口支持更高的代码密度和更有效的指令取址，bit handling 技术增强了数据处理的能力。

Thumb-2 指令是原始 16 位 Thumb 指令和新 32 位指令的结合，它提供与 32 位 ARM 指令集同样的功能。Cortex-M3 处理器的 ICode 接口对指令进行取指，每个取指的宽度为一个字。16 位指令在存储器中以半字对齐，取指时一次获取两个指令。对于字对齐的 32 位指令，取指在单个周期内完成；对于半字对齐的 32 位指令，取指最多在两个周期内完成，前提是访问零等待存储器。这样便可以灵活地使用指令，并实现低取指延迟。

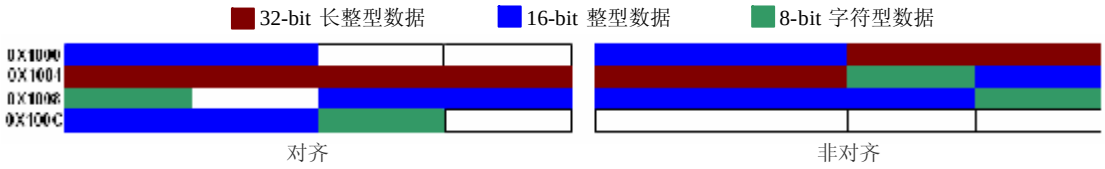
虽然嵌入式 flash 存储器在众多嵌入式应用中颇受推崇，但 flash（flash 工作在 30~40MHz 时）与处理器之间在速度上的差距也越来越大。为了消除这一差距，系统必须在访问 flash 时提供高效的指令和数据获取操作。Flash 存储器接口一般通过取指或者文字池取值来一次

获取多个 16 位的值。MOVW 和 MOVT 指令能够更有效地插入 16 位常量；使用这两个指令可以使数据立即可用，而无需像文字池那样需要执行一次加载操作。

为了有效地利用片上 RAM，许多应用方案采用打包成一个字节大小的信号量（即数据的每一个位）。对信号量进行原子处理的传统方法是读字节，屏蔽/修改位，然后再读回字节。Cortex-M3 使用 bit-banding 技术对这些过程进行了简化。1MB 大小的存储地址区域以别名标记为 32MB 位特定的空间。当对该 32MB 区域内的地址执行读或写操作时，它将自动对别名标记的地址实行位复位或位清零操作。通过该机制，多操作任务简化成了单核的写操作任务。

为了有效地利用SRAM，Cortex-M3 处理器还采用了非对齐数据访问技术。图 3-2是对齐和非对齐数据访问的存储空间分配图。每个格子代表一个字节长度的各种类型数据（每种类型数据要求的字节长度不同）。白色格子代表空的没有使用的存储空间。通过把尚未使用的存储空间整理成连续的空间，无形上减少了对SRAM的需求。

图 3-2 对齐和非对齐数据访问的存储空间分配



3.2 低成本的调试和跟踪技术

传统的系统调试一般由 JTAG 接口来执行，通常需要 5 个管脚，这对于总共才有 10 个管脚的低成本系统来说无疑是太奢侈了。Cortex-M3 处理器采用支持串行线调试（Serial Wire Debug）技术的调试访问端口（Debug Access Port-DAP），调试只需两个管脚便可完成，而且表现与 JTAG 接口同样出色。此外，把可选的 ETM 部件添加到系统有利于实现出色的指令跟踪功能，同时把成本的影响降至最低。

通过把 ROM 代码和数据独立地重新映射到 SRAM 区域上，Flash Patch 技术针对大多数只有 ROM 的微控制器实现了对只读代码的调试功能。这使得器件和系统工程师可以在运行期间对代码中的错误进行修补，从而免去了成本不菲的芯片重制过程。

3.3 低延迟中断处理机制

典型的 MCU 系统是中断强化的系统，但他们并没有为了对这些中断进行独立的服务而奢侈地采用昂贵的和智能的外设。在这种情况下，处理器本身得牺牲一定的性能来换取有效的中断响应和中断处理机制。Cortex-M3 内置的 NVIC 可实现硬件中断处理以及非常低的延迟，这有利于把对处理器性能的影响降至最小。NVIC 和处理内核的深度集成加快了中断服务程序（Interrupt Service Routines-ISR）的执行速度，同时也减少了进入中断所需的周期数，减幅达 70%。这一过程通过寄存器硬件压栈以及退出和重新开始多个 load-store 指令的执行来完成。

3.4 业内新的突破

ARM 架构成为 32 位 MCU 的标准已是不争的事实，它通常被称作下一代的 8051。Cortex-M3 处理器凭借其卓越的性能和低廉的成本，通常可以媲美高端的 8 位 MCU，现在

它的应用范围进一步扩大，使更多用户升级到 32 位系统成为现实。此外，初始微控制器器件在 Cortex-M3 处理器上整合了 flash、SRAM 和多个外设，缔造了$1 的价格神话，提供了无与伦比的竞争力。

第4章 可靠安全的汽车和工业控制

现今的汽车系统相当复杂，一些高端的汽车含有的微控制器已达上百个之多。这些系统的应用范围覆盖了从高性能的巡航系统到成本敏感的电子车窗控制和油压传感器功能等众多领域。不言而喻，如此复杂的可互操作系统需要极高的可靠性。而且在工业控制过程中，同样还要考虑关键控制操作的安全问题。是否有一款处理器可以满足这些要求，Cortex-M3处理器就是答案。

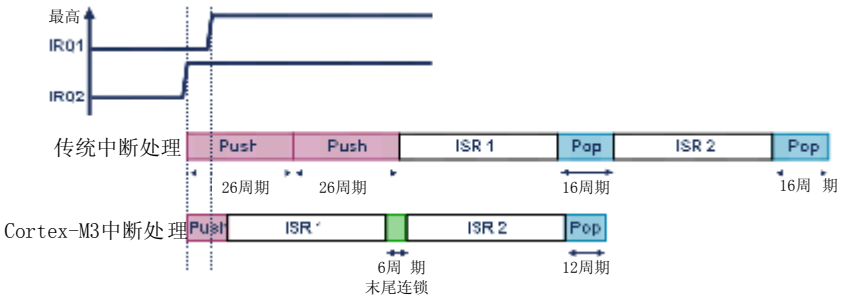
4.1 使用确定的中断处理来预测汽车的响应

汽车系统是由中断驱动的，这些中断包括刹车和急转弯等各种常规操作。对这类操作快速反应和预测有助于开启防抱死（ABS）系统和使用自动稳定控制技术，从而达到安全行驶的目的。一般来说，这些系统都非常复杂，各种不同的子系统为汽车的安全行驶随时待命。

NVIC 的 tail-chaining（末尾连锁）技术支持背对背中断，但有时优先级更高的中断有可能在当前中断的压栈或状态恢复阶段产生。在传统的基于 ARM7 处理器的系统中，这些阶段必须在开始处理未决的中断前完成。另一方面，Cortex-M3 支持后来和占先机制，因此对各种可能事件可提供确定性的响应。

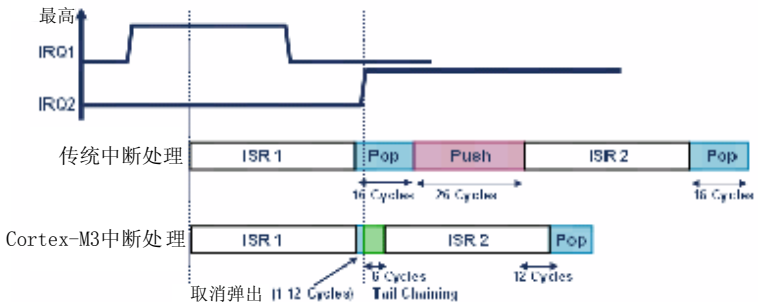
如果一个具有更高优先级的中断在上一个中断执行压栈期间到达，那么NVIC马上获取一个新的向量地址来处理该未决的中断，如图 4-1所示。

图 4-1 NVIC 对迟来的具有更高优先级的中断做出的响应



同样，如果在压栈期间发生异常，那么 NVIC 将丢弃压栈而马上处理新的中断。由于在转换到第二个中断前放弃了状态恢复以及保存阶段，所以 NVIC 以确定性的方式实现了更低的延迟。

图 4-2 NVIC 占先出栈



非屏蔽中断（Non-Maskable Interrupt-NMI）的确定性可以通过把关键的中断资源设置为非屏蔽形式而进一步提高，这在带有需要以特定的时间间隔进行可靠处理的看门狗定时器的系统中是一个非常重要的特性。

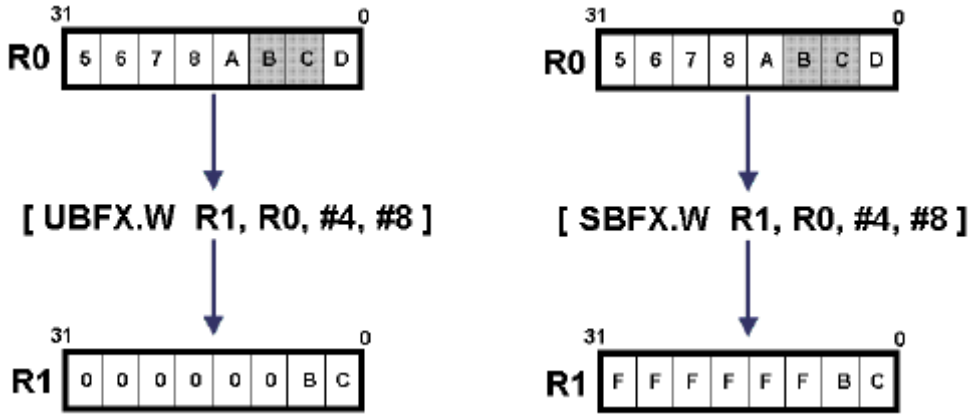
4.2 使用精细的存储器保护来获得可靠的软件集成

对基于汽车标准 API 的应用软件进行重复使用在下一代汽车系统中将成为必然的趋势。为了使产品更快地投入市场，汽车系统的设计者希望从不同的供应商那里选择软件模型，然后把这些模型重复应用到各种车辆。为此，我们需要建立一个机制来使各个模块分隔开，以便把干扰降至最小。Cortex-M3 处理器中的 MPU 提供可以细分到 32 字节的区域，实现了对单独任务的有效划分。如此一来，系统就可以把很多不同的小软件程序进行分隔，使它们避免保护机制的共享。

4.3 加快位提取的速度来获得有效的 I/O 数据处理

汽车和工业应用的特点在于它们需要处理大量的通用 I/O 数据。传感器接口或与其类似的外部器件对数据通常以 8 位或 16 位的形式读取，然后在处理所读的数据后提取所需的数据位。这些处理过程一般通过微代码来完成，所需时间为几个周期。此外，Thumb-2 UBFX（零扩展）和 SBFX（符号扩展）位字段指令可以使位级的提取在一个周期内完成。

图 4-3 使用 UBFX 和 SBFX 指令进行位提取



4.4 保护操作提供更安全的工业环境

在受保护的环境中，工业处理通常采用实时操作系统来执行控制。而且鉴于有多种不同的应用同时运行，一个保护机制被建立用来划分应用的安全级别。

为了安全操作，MPU 加强了优先权和访问规则。它在任务接任务（task-by-task）的基础上通过分离代码、数据和堆栈来实现安全的优先级，同时通过控制存储地址的读、写和执行来实现访问的限制。它还可以通过检测堆栈破坏等未知的、有害的存储器访问事件，和对这些问题进行安全和清洁的处理来避免灾难性的故障。

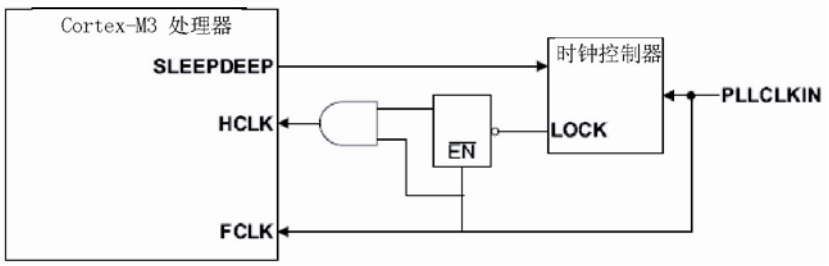
第5章 针对无线网络实现了更低的功耗

在世界日趋紧密相连的今天，无线网络无处不在。各种无线标准如 Wi-Fi、WiMAX、Bluetooth 和 Zigbee 纷纷面世，成为推动连接技术发展的新鲜血液，先后实现了跨互联网连接技术以及以类似方式相连的器件之间的连接技术。这些技术之间的共同点无一例外就是对低功耗的需求日益增长。例如，我们希望基于 ZigBee 的无线器件在使用一对 AA 电池供电的情况下可以维持工作超过 10 年。

5.1 时钟门控和内置睡眠模式可以降低功耗

时钟门控技术用于关闭系统中的某些部分来有效地禁止它们。Cortex-M3 处理器广泛地使用这些门控时钟来禁止不用的功能和不用的功能模块的输入，因此，只有使用中的逻辑才会消耗动态电源。

图 5-1 深度睡眠模式下的电源控制



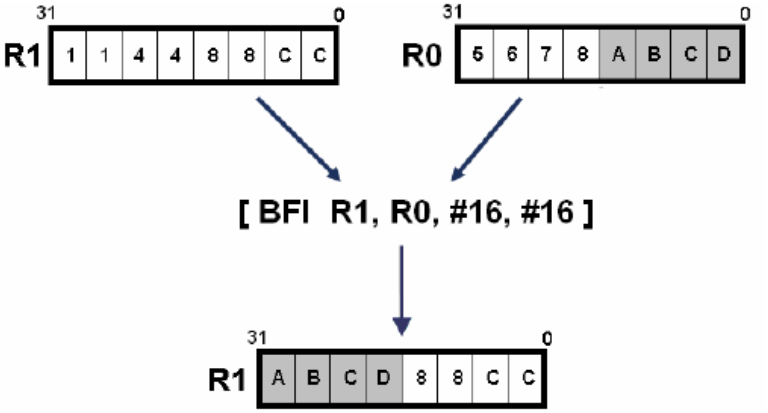
除了在处理器的设计中广泛使用时钟门控技术外，Cortex-M3 处理器还引入了一系列可以使处理器进入低功耗状态的睡眠模式，实现了有效地对整个处理器（中断控制器除外）进行时钟门控。睡眠模式通过WFI（等待中断）和WFE（等待事件）指令执行。此外，Cortex-M3处理器使用SLEEPDEEP信号更大幅度地降低了系统级功耗。如果处于立即睡眠（Sleep-now）或退出时睡眠（Sleep-on-exit）模式，该信号将在系统控制寄存器中的SLEEPDEEP位置位时发出。该信号发送到时钟管理器，对处理器和包括锁相环（PLL）在内的系统部件进行门控，功耗进一步下降。图 5-1展示了如何在低功耗模式下使用SLEEPDEEP信号来使时钟控制器停止，从而达到减小功耗的目的。当退出低功耗模式时，LOCK信号指出PLL处于稳定状态，并且可以安全使能Cortex-M3 时钟，确保了处理器不会在时钟信号稳定之前重新启动。为了检测中断，处理器必须接收低功耗状态下的自由运行（free-running）FCLK信号。

5.2 通过灵活的工作方式来增加处于睡眠模式的时间

高功耗通常是因为处理器为了达到一定的性能要求而在快速的时钟频率下工作产生的。处理器也可以通过灵活的工作方式来降低时钟频率。Cortex-M3 处理器的工作效率达到 1.25 DMIPS/MHz，使系统可以更快完成计算强度大的任务，因此，系统可以有更多时间处于低功耗的睡眠模式。

对于网络应用中的信息包处理任务来说，位处理技术非常重要。例如，位字段修改指令可以访问经过打包的数据结构，如网络信息包的包头。Cortex-M3 处理器中的位处理指令 BFI 和 BFC 可以减少总线活动以及对数据结构解包的需求，因此加快了网络处理的速度。

图 5-2 使用 BFI 指令插入任意数目的相邻位



5.3 真正实现省电

Cortex-M3 处理器如果不带可选部件，那么当目标频率为 100MHz，采用 TSMC 0.18G 的制造工艺和 ARM SAGE-X 标准单元库时，它的功耗仅为 0.24mW/MHz。在相同的频率下，Cortex-M3 处理器与 ARM7TDMI-S 处理器相比不但功耗降低了 30％以上，而且性能也翻了一番。

表 5-1 性能和功耗的比较（频率 100MHz，采用 TSMC 0.18 制程）

	CM3Core	Cortex-M3	ARM7TDMI-S (ARM)	ARM7TDMI-S (Thumb)
mW/MHz	0.19	0.24	0.28	0.28
DMIPS/MHz	1.25	1.25	0.93	0.74
DMIPS/mW	6.57	5.21	3.32	2.64

第6章 更快地投入市场

嵌入式系统设计人员都面临着同一挑战，那就是产品的开发速度问题，这个问题源于设计本身的复杂程度以及所使用软件工具的质量。在解决这一问题的过程中，早期系统的设计和软件被普遍采用。Cortex-M3 处理器可被集成系统部件和广泛的软件工具所支持，它为新系统以及那些升级到 ARMv7 架构的系统提供了一套完整的解决方案。

6.1 简单、可配置的硬件设计和调试

Cortex-M3 处理器可以进行高度配置和快速有效地适应系统的要求。通过把可选的 MPU 和 ETM 添加到系统以及在合成时移除 DWT 和 FPB，设计人员可以快速建立复杂的系统。在简单的操作系统中，NVIC 可以只带一个中断，而在汽车应用等中断强化的系统中，NVIC 可以支持多达 240 个物理中断并提供 256 个优先级别。在要求不同的处理可以安全操作的系统中，MPU 被用来加强处理的分离以及特权访问模式的使用。

随着花在应用验证上的时间日益增多，片上调试和跟踪对于产品的准时交付已经变得毫无意义。Cortex-M3 处理器的集成调试功能可以实现快速验证，而无需使用 ICE 元件。系统可以通过 JTAG 端口或两脚串行线（Serial Wire Debug）端口进行观察。可选 ETM 具有出色的指令跟踪功能，而 FPB 和 DWT 针对调试实现断点和硬件观察点的使用。

6.2 简易的应用程序开发

Cortex-M3 处理器包含许多可以实现快速软件开发的特性，使开发人员无需编写任何汇编代码或对处理器及其寄存器集有深刻的认识。Cortex-M3 处理器带有一个被简化和更易于理解的基于栈的编程模型，以及一个用于处理中断和异常的基于向量的中断机制。此外，Thumb-2 指令可实现有效编码而无需交错使用 ARM/Thumb 指令。简单的线性 4G 地址空间不包含数据页或代码页。该架构提供了灵活的寄存器机制，使得编译非常友好。该机制允许任何寄存器之间的单周期乘法操作和把任意寄存器作为数据结构/数据指针来使用。

在主 C 程序被调用之前和之后以及在启动引导代码期间，传统的 ISR 需要一个汇编程序包来处理堆栈操作。Cortex-M3 处理器的所有堆栈操作都在硬件中完成，因此不需要汇编程序，开发人员可以只用 C 语言编程，而不用了解处理器和所有寄存器组具体如何运作。在许多操作系统中，硬件定时器被用来生成中断，以便 OS 可以执行任务管理。这对于确保在系统上可以运行多个任务以及避免出现单个任务占用所有资源的情况来说非常必要。SysTick 定时器专门用于实现这一功能，它使 OS 在基于 Cortex-M3 处理器的 MCU 器件之间的移植变得更加简单，移植时无需对 OS 的系统定时器代码进行修改。

RealView® DEVELOP和CREATE系列工具完全支持Cortex-M3 处理器。凭借RealView SoC Designer技术，系统硬件可以在基于Cortex-M3 处理器的系统上建立原型，软件可以及早进行开发和调试。处理器内核以及外设均使用精确到周期（cycle-accurate）的模型，因此器件驱动程序和硬件的设计可以同时进行。这样的话，软件的开发便可以在距离硬件设计完成很早之前开始，从而赢得更理想的上市时间。

使用 Thumb-2 ISA，RealView 开发套件（RealView Development Suite－RVDS）实现了一个简化的编译流程，免去了析构、编译和交错使用 ARM 和 Thumb 程序的需要。早前使用 Thumb 编写的应用程序不必经过修改就可以在 Cortex-M3 处理器上运行，因为 Thumb-2

ISA 包含了所有 Thumb 指令。此外，ARM 汇编代码通过 ARM Unified Assembler 结构可以轻易地移植到 Thumb-2 指令。如果源代码是用高级语言（如 C）编写的，那么可以使用 RealView 编译工具或 GNU 编译器等第三方工具将其重新编译到 Thumb-2 代码。

RealView 微控制器开发套件（RealView Microcontroller Developer Kit）为所有基于 ARM 处理器（包括 Cortex-M3 处理器）的微控制器提供了完整的软件开发环境。这里随便提一些第三方开发工具，包括 IAR、Green Hills 和 Lauterbach 的编译器、仿真器和调试器等。

第7章 总结

Cortex-M3 是首款基于 ARMv7-M 架构的 ARM 处理器。中心 Cortex-M3 内核使用 3 级流水线哈佛架构，运用分支预测、单周期乘法和硬件除法功能实现了出色的效率（1.25 DMIPS/MHz）。Thumb-2 指令集结合非对齐数据存储和原子位处理等特性，轻易以 8 位、16 位器件所需的存储空间就实现了 32 位性能。

凭借灵活的集成硬件配置，快速的系统调试和简易的软件编程，基于 Cortex-M3 处理器的设计得以更快地投入市场。为了在中断强化的汽车应用中实现可靠的操作，内置的嵌套向量中断控制器（Nested Vectored Interrupt Controller-NVIC）通过末尾连锁（tail-chaining）技术提供了确定的和低延迟的中断处理并可以通过设置带有多达 240 个中断。对于工业控制应用，可选存储器保护单元（MPU）通过使用特权访问模式和分离应用中的处理进程来实现安全操作。Flash 修补和断点（Fllash Patch and Breakpoint-unit）单元、数据观察点和跟踪（Data Watchpoint and Trace-DWT）单元、测量跟踪宏单元（Instrumentation Trace Macrocell-ITM）和可选嵌入式跟踪宏单元（Embedded Trace Macrocell- ETM™）为深度嵌入式器件提供了廉价的调试和跟踪技术。扩展时钟门控技术和内置睡眠模式为低功耗的无线设计铺路。

Cortex-M3 处理器是专门为那些对成本和功耗非常敏感但同时对性能要求又相当高的应用而设计的。凭借代码大小和中断延迟的优化、集成的系统部件、灵活的配置、简单的高级语言编程和强大的软件系统，Cortex-M3 处理器将成为广大系统（从复杂片上系统到低端的微控制器）的理想解决方案。