2

闪存的应用

Gary Forni, Collin Ong, Christine Rice, Ken McKee, and Ronald J. Bauer

# 2.1 简介

消费电子行业正在从成熟的家用电子和电器市场的主导地位迅速过渡到迅速崛起的移动电子市场。蜂窝电话、掌上电脑和数字音乐播放器是填充消费者口袋的一些最明显的电子小工具的例子。闪存通过为定义其功能的程序代码以及必须携带的越来越多的用户数据提供存储，使这些设备得以诞生。闪存为移动应用提供了理想的属性组合：低功耗、非易失性、速度和小尺寸。闪存是一种关键的使能技术，因为其他的替代技术会违反移动设备的主要设计标准，从而造成了折衷。例如，使用基于磁盘的存储就不能实现小型化和坚固性。

蜂窝电话为Flash在移动应用中的作用提供了一个有启发性的案例研究（图2.1）。早期的手机使用只读存储器（ROM）设备来存储他们的程序代码，这对于他们有限的功能和硬接线的模拟协议来说是一个足够的选择。然而，使用可擦写可编程ROM（EPROM）进行开发的设计人员很快就转向了闪存设备，以加快原型设计周期。生产工程师也可以通过使用闪存看到代码版本库存控制的好处。但是，在生产型手机中转向使用Flash的驱动力来自于数字蜂窝协议。Flash的系统内可重编程性是需要的，以支持世界各地实施的各种数字协议和规格的频繁更新。随着标准的稳定，制造商希望通过将他们的参数数据存储从电可擦除可编程的ROM（EEPROM）转移到闪存，并尽量减少静态随机存取存储器来巩固他们的存储器子系统。通过直接从闪存中执行，将ROM（EEPROM）转移到闪存，并尽量减少静态随机存取存储器（SRAM）的要求。(通过直接从闪存执行，最大限度地减少静态随机存取存储器（SRAM）的要求。

*以闪存为重点的非易失性存储器技术*。J. E. Brewer和M. Gill编辑，版权所有©2008年美国电气和电子工程师协会。

****

图2.1.闪存使移动设备成为我们口袋里必不可少的一部分内容。(由*摩托罗拉公司提供。*)

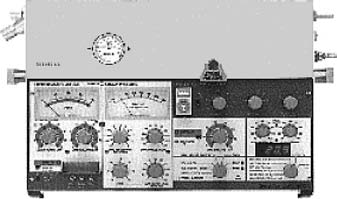


图2.2.闪存也是工业和嵌入式设备的一个重要组成部分。

为了实现这一目标，闪存需要同时处理代码和数据，这些功能由升级后的闪存软件媒体管理器提供。闪存处理代码和数据的能力提供了实现新功能所需的基础设施，如语音记录、信息传递和电子邮件。随着Flash设备容量的增加，更多复杂的应用出现在手机上，如互联网浏览、语音识别和视频。因此，从一开始仅仅是代码存储，Flash已经成为手机新功能和应用的驱动力。

然而，便携式应用并不是唯一采用闪存的应用。闪存已经成为嵌入式设计的标准系统存储器，因为它为工程设计提供了好处，可以在开发过程中快速更新代码，也可以通过ROM版本库存管理和现场更新为生产和支持提供好处（图2.2）。因此，闪存现在被用于各种不同的应用，如在网络路由器中存储地址表，在医疗监测设备中捕捉生物识别数据。

# 2.1.1 存储设备的频谱

系统设计者在一个新项目开始时面临两个熟悉的决定。他们必须根据应用的需要和要求选择一个处理器和一个存储器。处理器和内存共同决定了系统的核心，而所选择的内存类型则进一步决定了系统的架构。

表2.1.常见内存技术的优势和劣势

|  |  |  |
| --- | --- | --- |
| 存储器类型 | 优势 | 弱点 |
| 身份证 | 成本，非挥发性 | 不可写入，不可擦除 |
| EPROM | 可擦除，非挥发性 | 非系统内可写 |
| 内存 | 快速读/写，可改变字节数 | 成本、波动性 |
| EEPROM | 非易失性，可改变字节数 | 低密度，成本 |
| 磁盘 | 高容量、低成本、非易失性 | 访问时间、可靠性、楼面成本、存储空间 |
|  |  | 和下载的执行模式，而不是 |

闪存 成本、非易失性、高密度、

可擦写、快速读取

可改变的字节

不可以改变字节，写的时候不可以单周期。

由于有大量的存储器技术类型可供选择--随机存取存储器（RAM）、动态RAM（DRAM）、SRAM、铁电RAM（FRAM）、硬盘驱动器（HDD）、ROM、EPROM，当然还有Flash--如何选择？所有的存储器都是优点和缺点的折衷，而手头的具体应用要求将决定必要的权衡。表2.1总结了目前普遍使用的存储器类型的优点和缺点。

# 2.1.2 从EPROM演变而来

在20世纪70年代和80年代前半期，EPROM作为控制处理器的主要指令存储手段，统治了电子设计界。它们是非易失性的，这意味着当它们的电源被关闭时，它们的内容不会丢失，而且密度相对较高，这意味着它们可以容纳很多。你甚至可以通过把它们放入紫外线（UV）灯箱中20分钟来擦除它们，或者在紧要关头，把它们放在阳光下几天。然后你可以在几分钟内对它们进行重新编程，把它们放回插座中，然后用你的新代码运行。资深的系统设计人员可能会回忆起那些通过轻触拨动开关来加载内存的日子，或者也许他们足够幸运，可以加载成堆的打孔卡。无论哪种方式，他们在一天结束时都会失去他们的程序代码和数据，如果他们足够幸运，没有在断电时失去一切的话。对他们来说，设计一个使用16KB（千字节）非易失性、可擦除存储器的系统似乎是美好的生活。

EPROM的工作原理是在晶体管的ﬂoating（或绝缘）栅极上捕获电子，从而存储信息。制造EPROM的故障模式之一是电子从ﬂoating栅极上泄漏。如果这些电子泄漏，即使是缓慢地泄漏，也会影响到所存储的信息。对于从事闪存的工程师来说，这显然是一个必须解决的问题。

然而，一些研究该问题的工程师认为他们可以利用泄漏作为一种优势。你应该用紫外光来擦除EPROM，紫外光可以用几个额外的电子伏特的能量将电子从ﬂoating gate上踢下来，但是如果泄漏也将电子踢下来，那么为什么不利用这个错误来擦除芯片呢？东芝的一些其他工程师也意识到了同样的问题，但英特尔的工程师是第一个将这一想法付诸实践的人，"闪存EPROM"，即所谓的 "闪存"，因为它们可以在没有紫外线的情况下通过电子方式快速擦除，于1988年投入生产。

从紫外光擦除的EPROM到电擦除的Flash存储器的变化产生了一个立竿见影的后果：芯片封装顶部让紫外光照射进来的窗口消失了。但除了更快、更方便的擦除之外，闪存似乎与普通的EPROM没有什么不同。事实上，这一变化的全部内容需要12年的时间来发展，今天，闪存行业仍然没有用完的功能可以增加。

由于其根源是EPROM的延伸，闪存很容易被采用为更容易使用的EPROM，但同时，这些根源带来了系统结构、设计和制造技术的遗产，减缓了更适合闪存的系统内存使用模型的发展。嵌入式设计中最普遍的使用模式可以分为三组：代码存储、数据存储和代码+数据存储。

本章将讨论这三种使用模式，它们与闪存应用的关系，以及闪存设备和系统设计的相关属性。

# 2.1.3 NOR和NAND

两种主要的闪存技术是NOR和NAND，每一种都有优势和劣势。NOR型闪存首先作为EPROM的替代品进入市场，并在几乎所有的计算机和嵌入式设计的各种应用中的代码和代码+数据存储方面保持主导地位。NAND型闪存出现得较晚，在可移动卡领域主要是一种数据存储存储器。

英特尔、ST Micro和大多数Spansion闪存都是基于NOR技术，而东芝和三星是NAND闪存的主要供应商。

***2.1.3.1 NOR型闪存。***NOR型闪存的优势在于快速的随机访问读取和可靠的存储。

代码的执行通常需要快速的随机存取读取，因为代码在分支时从一个内存位置跳到另一个位置。出于这个原因，NOR闪存架构的高速随机存取能力是代码存储应用的理想选择。

存储在闪存中直接执行的代码必须完全可靠，因为任何错误都会产生系统故障。NOR闪存保证100%的良好位，消除了对纠错代码或控制器的任何需要。保证的 "良好位 "通过消除系统中的纠错需要而简化了实施。

这两种属性使NOR闪存非常适合于代码和数据的组合应用（代码+数据）。代码可以根据需要随机访问，而数据可以从同一设备中可靠地存储和检索。这种存储器结构以最低的元件数量提供最大的功能。

NOR闪存最常见的用途是在嵌入式应用和移动电话中用于代码和数据存储。然而，NOR闪存在任何需要快速、可靠的随机访问读取的应用中都能很好地工作，如代码或代码+数据的组合存储。此外，16位或8位宽的并行数据总线允许快速将代码和/或数据传输到NOR闪存中或从NOR闪存中取出。

***2.1.3.2 NAND闪存。***NAND型闪存的优势在于快速写入和高密度阵列，以牺牲随机访问读取能力为代价降低了芯片成本。

NAND非常适用于仅有数据的应用，如音频存储、视频存储或数据记录，这些应用需要快速的数据写入，并且数据将被连续访问。由于其快速写入性能，它在这些类型的应用中效果最好。

NAND闪存的随机访问读取性能较差，但顺序读取性能良好。由于NAND的晶体管在其阵列中的排列方式，NAND的数据是按顺序访问的。一个页面中的第一个访问是缓慢的，因为系统必须等待读取电流通过一列闪存单元传播。在按顺序读取数据的应用中，如音频和视频播放或数据记录，这是一个好处。然而，如果需要随机访问，就像代码执行那样，性能就会很差。

基于NAND的存储器在给定的光刻技术下具有较小的单元尺寸，但需要更复杂的电路来实现。NAND闪存通常不能保证100%的良好位，但通过纠错和单元冗余来解决这个问题[纠错代码（ECC）被用来确保从系统中删除坏块的错误]。冗余的存储单元和所需的纠错增加了介质或系统的复杂性和成本。

# 2.1.4 闪存使用模式的演变

两种闪存使用模式，即代码和数据存储，从一开始就很明显，而且都是从以前的内存技术使用方式演变而来的。直到最近，第三种使用模式，即代码+数据，才专门围绕着闪存的优势发展起来。

***2.1.4.1 代码使用模式。***历史上，最基本的系统结构模型是将中央处理单元（CPU）的指令存储在一个非易失性存储器中，如ROM或EPROM，它直接连接到CPU的存储器总线上进行指令获取。系统设计者在试图满足其他存储器子系统的要求时，将这种模式向不同方向发展，如性能和非易失性存储。

随机存取存储器增加了更高速度的代码访问，以提高性能，并提供了刮板空间。各种类型的RAM设备由于其快速的访问时间而在代码执行方面表现出色，但由于它们在断电时失去了内容，因此在系统中必须有另一个重复的存储器来在停机期间存储代码和数据。这个问题可以通过直接执行存储在ROM中的代码来解决，用较少的RAM作为数据的临时工作副本，但是较慢的ROM访问时间会降低系统的性能，而且不可更改的内容会在工程中造成不便，在制造中造成库存问题，以及在现场的可升级性问题。

用EPROM代替ROM，可以在工程中和产品出厂后更新代码，但仍然需要将设备拉到紫外光箱中擦除。无论是ROM还是EPROM，都没有规定在系统断电时持续存储数据，因此可以添加另一个存储器，即EEPROM，用于非易失性参数存储。

加入EEPROM是为了支持小型系统参数和设置的非易失性存储。磁盘被添加以支持不断增长的软件代码和数据大小。这种内存技术的扩散使系统变得复杂，并增加了内存子系统的冗余度。现在，系统必须包括许多内存设备：用于持久性代码/数据存储的磁盘，用于CPU启动代码的ROM/EPROM，用于执行下载代码和工作数据的RAM，也许还有一个用于配置参数存储的EEPROM。

这种模式的一个极端例子是标准的个人电脑系统结构，它利用ROM、EPROM或Flash设备作为基本输入/输出系统（BIOS），DRAM作为主代码执行和临时数据存储，EEPROM/CMOS（互补金属氧化物半导体）作为配置参数，磁盘作为非挥发性代码和数据存储。这种结构是不理想的，因为它在磁盘和RAM之间复制代码和数据时重复了存储空间，并为几乎只在启动时使用的BIOS使用单独的存储器。

在个人电脑或更大规模的系统中，这种复杂的重复记忆可能是可行的，但许多嵌入式或便携式应用不能支持所有这些冗余设备的尺寸、成本和功耗。一个更典型的嵌入式系统设计将包含一个用于代码存储的闪存设备，用于刮板和代码执行的SRAM，以及一个用于小参数存储的EEPROM。一些系统还可能使用电池支持的SRAM进行半永久性存储。

在这个权衡的泥潭的尽头是闪存。由于访问时间快，从闪存中直接执行代码是可能的。它的电气可更新性使工程师和终端用户都能方便地进行更新，并解决了EPROM和ROM的库存和代码修订管理问题。当由软件数据管理器管理时，它的字节级写入可以实现EEPROM式的数据存储。

***2.1.4.2 数据使用模型。***从另一个方向来看，闪存也被誉为基于磁盘的存储设备的继任者，用于所有使用软盘或硬盘的应用程序，但在现实世界中的成功非常有限。

设计师在这些类型的应用中对基于磁盘的存储的热情一直是喜忧参半。磁盘存储非常便宜，但笨重、耗电，最糟糕的是，对冲击和振动很敏感。闪存，以其小尺寸、非易失性数据保留和固态可靠性，被视为这些存储问题的解决方案。然而，由于每兆字节的闪存成本比磁盘高，它在这个市场上的渗透一直受到阻碍。尽管如此，随着便携式数字采集应用的日益普及，如个人数字助理、数码相机和数字音频记录器，闪存在数据存储方面的应用一直在增加。这种数据存储的采用大多是在可移动的闪存卡中，而不是在嵌入式设备中。

***2.1.4.3 代码+数据使用模式。***由于这些能力，闪存已经在很大程度上取代了其他内存设备在嵌入式和便携式应用中的代码存储，并在一定程度上取代了数据存储。闪存的使用模式主要是根据应用是以代码还是以数据为中心来区分。然而，除了在传统使用模式中取代这些传统存储器的功能外，闪存最终发展到了它自己的使用模式，即单芯片代码+数据存储。

# 2.1.5 了解Flash属性

虽然它们是作为EPROM的延伸而发展起来的，但随着时间的推移，闪存已经获得了自己的一套属性，改变了它们在系统应用中的使用方式。(见表2.2)了解这些属性是掌握Flash使用模式的功能和限制的重要第一步。本节将探讨最重要的Flash属性以及它们如何影响系统设计。首先，将讨论设备级的特性，然后讨论Flash软件和包装在构成一个完整系统中日益增长的作用。

表2.2.重要的闪光灯属性和效果的总结

属性 相关度量 它在应用中的作用

随机存取读取时间

页面模式界面

突发模式接口

纳秒（ns）；毫秒（ms）

每页的字数，通常为4或8个字

页面访问时间（ns） 每突发字数，通常为4、8、16字或连续字数 最大零等待状态时钟频率（MHz）

代码执行（如果在纳秒范围内，否则，只有数据）。

更快的代码执行，支持更高性能的页模式CPU

更快的代码执行，在零等待状态下支持最高性能的突发模式CPU

总线宽度位 ，通常为8或16位，与CPU总线宽度相匹配，以获得更高的性能。

每周期的数据吞吐量

密度Megabits (Mb) 更大的代码或数据尺寸导致更多的

复杂的应用

写入状态机

支持的命令 将系统软件从低级别的

闪存管理使写到闪存的程序

阵列阻断块的大小，单位是千比特（kb）。

通常为8-64kb

模块化的代码/数据更新，而不需要冒整个阵列内容的风险，使写入闪存成为常规。

不对称的阻断

区块大小、数量、排列

根据基本代码或代码+数据的应用模式，有效地确定块的大小（启动、参数、主）。

对称的阻断

电压供应

块大小（kb） 易于将多个设备集成到闪存阵列中，使总的闪存存储容量大于单个设备的大小。

读取电源，伏特(V) 与系统电源电压 相匹配；较低的

电源电压导致更长的电池寿命

程序/擦除电源，伏特（V）。

如果它与系统电源电压相匹配，那么它就会启用对闪存的常规写入，从而实现系统内的代码、数据更新。

I/O电源，伏特(V) 与逻辑运行电压的 接口

低于Flash电源电压，导致更低的功率消耗

表2.2.*延续*

属性相关指标它在应用中实现的功能 循环 k循环，通常为100+k 当足够多时，写入成为常规

循环可用，使频繁的

系统内更新和代码+数据存储

可暂停的模式（编程、擦除、擦除时暂停编程，等等）。

暂停时可进行的操作（读取、编程，等等）。

在需要时必须暂停，以便在需要时灵活地服务于中断或捕获数据；启用边读边写的软件解决方案 在需要的时间执行需要的操作；启用边读边写的软件解决方案。

暂停延迟（ms） 确定的暂停延迟允许

软件边读边写，在实时系统中工作

分区的数量 和大小

分区

在其他分区的程序/擦除过程中可使用的功能

决定了代码和数据之间的内存分配

如果程序、擦除或其他闪存操作可以在不同的分区中同时进行，则增加了系统功能的灵活性；增加代码执行和数据存储的吞吐量。

写保护

硬件或软件控制

硬件控制增加了软件之外的额外保护

块状锁定个人

或组锁/解锁

单个区块的锁定控制消除了其他区块不必要的暴露，提供了更多的安全性

锁定变化的延迟低延迟允许锁定变化到

实时发生

硬件联锁提供了额外的保护层

为代码或引导代码

软件 提供的管理功能

将系统软件与低级别的闪存管理隔离开来，并提供更高级别的接口，因此开发工作可以集中在应用上，而不是基础设施上。

***2.1.5.1 设备属性。***闪存拥有自己的一套属性，确定了产品的类别，每个规格的产品以稍微不同的方式实现这些属性。因此，了解每个属性对于衡量一个特定的闪存设备在应用中的实用性非常重要。

访问时间和读取接口。像所有的存储器一样，闪存首先是通过访问时间来衡量的。访问时间是指从给定一个地址位置到闪存设备从该位置输出存储数据之间的延迟。这个属性对系统设计有明显的影响，因为它与代码和数据检索的系统性能有关。对于代码，它限制了CPU可以运行的时钟频率。对于数据来说，它决定了数据检索的性能。

工艺几何的减少加快了闪存的访问时间，但与此同时，降低电源电压的趋势已经抵消了性能上的好处。因此，现在提高闪存读取性能的主要方法是改变架构，纳入改进的读取接口，如页面模式或突发模式读取，以满足最新的低电压和高性能CPU。

替代的读取接口的操作与DRAM上常见的同类接口类似。页模式是一种异步读取模式，在这种模式下，闪存内部一次读取一个页面（通常是四个或八个字）到一个页面缓冲器中，这样，当低阶地址线发生变化时，它们可以非常迅速地依次输出。突发模式是一种读取模式，它向CPU同步输出连续的数据突发。突发长度根据处理器的要求而变化。突发模式是目前普遍使用的最快的接口在闪存设备上。

总线宽度。总线宽度是指一个闪存设备在同一时间提供的比特数。总线宽度通常是8位（×8，"按8"）或16位（×16，"按16"）。一些闪存器件的总线宽度可以通过输入引脚来选择。总线宽度是影响性能的一个因素，因为更宽的总线可以在一个周期内传输更多的数据。通常情况下，系统设计者会选择一个与系统CPU的总线宽度相匹配的闪存设备。由于CPU已经从8位和16位转移到32位和64位总线，闪存设备通常被搭配在一起以提供所需的总线宽度。

密度。闪存密度是存储器所存储的数据量的同义词，通常以兆比特（Mb）为单位。最早的NOR闪存是256kb（千比特），后来在英特尔StrataFlash产品系列中增长到了512Mb。NAND闪存设备也同样从16Mb增长到8Gb（千兆位）。密度与应用直接相关，因为它需要的代码、数据或代码+数据的存储量必须由一个或多个闪存设备来满足。

写入状态机。第一批闪存设备沿用了EPROM控制写操作的先例。系统有责任对电压和信号进行控制，以正确执行编程或擦除操作。在数字系统的控制下执行这个基本的模拟过程是有问题的，而且涉及到难以控制的超时和写入脉冲。这些算法的实施所产生的变化导致了这些早期闪存设备的可靠性降低。此外，这些算法的时间要求意味着系统的CPU必须完全致力于这项任务，从而使系统的运行陷入停顿。所有这些都意味着系统内的写操作并没有被系统设计者常规使用。解决方案是将程序/擦除算法的控制引入到闪存设备内部，通过在片上写状态机中实现这些算法，适当地控制所需算法。

写入状态机作为一个标准功能，增加了设备的可靠性，并允许CPU使用标准总线周期写入闪存。增加的易用性和可靠性是使系统内写入闪存成为常规设计做法的第一步。

阵列阻断。虽然大多数NOR型闪存一直是以单字节或单字为基础进行编程的，但第一代闪存设备有一个单一的可擦除存储空间，就像它们的EPROM前辈一样。

1一些闪存设备需要以页为单位进行编程。虽然这通常与512字节左右的较小块大小有关，但它增加了更新单个字节或字所需的开销和风险。

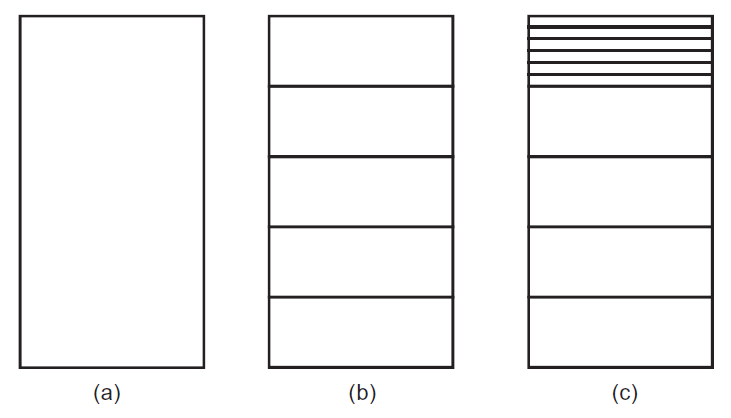


图2.3.各种Flash封锁结构。(*a*)批量擦除，(*b*)对称，和(*c*)不对称。

这就是为什么它们有时被称为*批量擦除*闪存器件[图2.3(*a*)]。为了更新批量擦除式闪存，整个存储器阵列必须被擦除，然后对整个阵列进行重新编程。这种方法违背了稳健的系统设计惯例，因为如果在擦除之后，但在重新编程完成之前，这个过程被中断，系统就会失去功能。系统级的安全防护措施，如使用重复的闪存设备，允许变通，但却是多余的和不合适的。

虽然EPROM有理由同时擦除整个存储器（它们的窗口将整个存储器阵列暴露在擦除的紫外光下），但闪存技术没有窗口，因此必须进行全阵列擦除。这导致了另一项重大的结构创新：将存储器阵列划分为独立的可擦除块[图2.3(*b*, *c*)]。阻断的最早应用之一是在 "引导块 "结构中。一个小的引导块从阵列的其他部分分割出来，并被单独锁定，这样系统就可以把基本的开机引导代码放在引导块中，并锁定它以防止意外擦除。现在，即使固件以某种方式被擦除，系统也能自己启动，足以到软盘上重新启动更新。引导块结构，其大小块的组合，至今仍是最受欢迎的闪存设备之一。

以更小的块擦除闪存，使代码和数据存储的管理更容易和更安全。大多数人都想知道为什么块的大小没有完全减少到单字节/字擦除的理想状态。原因是，块越小，晶体管和芯片面积的惩罚就越大，这就增加了成本。虽然较小的区块更容易使用，擦除速度也更快，但就芯片尺寸而言，它们的成本更高，因此，每个封锁方案都必须平衡其区块尺寸与设备成本和目标应用的需求。

不对称与对称。闪存阵列被划分为块的方式使其对特定类型的存储进行优化。由于数据倾向于更频繁地变化，较小的块更适合存储数据，因为它们的擦除时间更快。另一方面，代码的变化并不频繁，较大的块可以用来存储代码，以节省芯片的尺寸和成本。

两种常见的分割阵列的方法是对称的和非对称的。对称分块[图2.3(*b*)]将内存阵列分成大小相等的块。通常选择的块大小是64 kB，英特尔FlashFile架构使用的就是这个块。对称分块往往适合于代码或数据的存储，但对于在同一设备中混合使用代码+数据来说，并不是特别优化，因为大块大小的擦除相对较慢。

非对称阻塞[图2.3(*c*)]将阵列分为不同大小的块，小块用于数据，大块用于代码，根据代码+数据的要求优化块的大小。不对称分块的一个副作用是需要单独的顶部和底部的Flash设备。因为有些微处理器在内存图的顶部寻找启动代码，而其他的则在底部寻找，所以需要镜像的内存图。一个为代码+数据优化的非对称块方案的例子是英特尔的Advanced+Boot Block架构，它的特点是8 kB块用于小数据参数存储，64 kB块用于代码和大数据存储。

今天，不断提高的闪存擦除性能和日益复杂的软硬件数据管理器正在减少对小块和不对称块方案的需求。此外，随着嵌入式设备通过网络、互联网和无线链路的连接，它们存储的数据量不断增加，将它们的存储需求推向更大的块大小。

电压供应。当闪存首次出现在系统设计中时，它们是ROM或EPROM的直接替代品，因此它们大多存储稳定的、不变的代码，只对制造工艺有改进的可能性。明显的下一步是提供系统内的更新，即用一个全新的代码映像来重塑系统闪存设备的能力，类似于今天个人电脑的BIOS芯片的更新方式。然而，为了实现这一目标，系统需要为闪存芯片的编程电压引脚*V*pp 提供12V的电源，以及为芯片的工作电压引脚*V*cc 提供5V的电源。这种双电源要求增加了实现系统内更新的复杂性。由于大多数系统不会让12-V电源持续供电，这一要求意味着对闪存的写入被认为是一个偶然的事件，而不是系统操作的常规部分。

虽然12-V差值仍然是NOR型闪存单元（最适合代码存储和代码+数据应用的类型）的内部要求，但现在该要求由器件本身的内部电荷泵来满足，它可以为闪存编程和擦除操作提高系统级输入电压。今天，典型的NOR型闪存芯片可以用单一的系统电源电压进行读取、编程和擦除。一些闪存器件仍然允许临时使用更有效的外部高电压电源，以加快制造过程中的编程。

目前对闪存工作电压的设计要求是在标准系统电压下提供读取、编程和擦除能力。这个系统电压已经从5V下降到3.3V及以下。由于无线电话等移动应用是闪存的最大采用者之一，闪存的工作电压正在积极地降低到2.5、1.8V，甚至更低，以增加电池寿命。

循环。闪存设备通常有一个最大擦除周期的规定。擦除周期数是指一个块被擦除的次数。对于大多数闪存设备的周期是以块为单位计算的，一个块的擦除并不影响其他块的可靠性或性能。

循环次数是相关的，因为器件的性能规格，如块擦除时间或字节编程时间，只能保证达到最大擦除循环规格。这是因为擦除闪存块时，会对存储在芯片中的信息的ﬂoating gate的绝缘氧化层造成磨损。一个广泛存在的对循环规格的误解是，一旦达到最大循环规格，设备将失效或停止工作。事实并非如此；在大多数情况下，设备将在超过最大循环规格后继续运行。该规格标志着制造商可以保证设备性能规格的擦除量。

自从闪存问世以来，擦除循环的规格有了很大的改善。第一批批量擦除的闪存设备的额定周期为1000次，但10000次的规格很快就很普遍了。在闪存设备中增加了嵌入式写状态机，对擦除过程进行了更严格的控制，使100,000周期成为可能。今天，典型的设备的额定值是100,000到1,000,000次，但是改进的介质管理技术（见第2.4.1.7节）已经减少了典型数据存储过程中的擦除次数，所以100,000次对大多数应用来说是足够的。

暂停。直接从闪存中执行系统代码并期望写入同一闪存设备的第一个问题是*边写边读*问题。这个问题指的是，标准的单分区闪存设备在写操作（编程或擦除）进行时不能被读取。因为内部写状态机正在操纵内部控制信号和电压来对闪存阵列的单元进行写操作，这些相同的闪存单元不能同时被读取。因为系统通常是这样设计的：CPU的指令是从闪存中获取的，这就造成了如下的情况。CPU从哪里获得它所需要的指令来控制闪存的写入操作，而这些指令在闪存写入操作开始后就会变得不可用？这个问题在代码+数据的应用中更为严重，因为向Flash写数据比修改代码要频繁得多。

处理边读边写问题的标准方法是将写入闪存所需的最小代码复制到RAM中，并在写入操作中从RAM中执行。然而，这种方法的缺陷是在闪存写入操作过程中发生中断的情况下，无法从闪存中执行额外的代码。为了缓解这一限制，在闪存写入状态机中加入了写入暂停功能。

写入暂停允许CPU中断正在进行的编程或擦除操作，从闪存阵列中读取，然后恢复暂停的操作。最近的闪存设备支持擦除暂停读取、程序暂停读取、擦除暂停编程，甚至是嵌套的擦除暂停到程序暂停读取。当从RAM中执行Flash写代码时，中断处理程序可以轮询中断，并暂停写操作，以便执行代码，为中断服务。这种中断处理的方法有时被称为*软件边读边写*。

暂停会在暂停延迟和写性能方面产生一些性能损失。暂停延迟是写状态机暂停正在进行的写操作并将闪存阵列返回到读模式所需的时间。

应根据系统的最大中断服务延迟来评估闪存设备的最大暂停延迟规格，以确保兼容性。写入性能会受到暂停的影响，因为编程或擦除在暂停期间是暂停的。在一个经常中断闪存写入操作的系统中，数据存储的吞吐量将被降低。

双分区。最近的一个架构发展旨在通过减少对暂停的需求来缓解与编程或擦除暂停相关的性能损害，但代价是增加硅面积。双分区闪存设备将闪存分为两个分区，可以在一定程度上相互独立操作。例如，CPU可以在一个分区开始擦除或编程操作，同时在另一个分区执行代码。这消除了从RAM中执行闪存写入代码的需要，因为该代码可以从与被写入的分区相反的地方运行。一般的使用模式是将代码放在较大的分区中，数据放在较小的分区中。然而，需要仔细规划代码和数据的位置，以确保所需的代码在需要时总是可以从其中一个分区获得。另外，双分区设备也不能完全消除对暂存器的需求。如果需要从一个已经忙于写操作的分区中读出一些东西，那么仍然需要暂停。

在未来，闪存设备可能被进一步细分为更多的部分。多分区闪存将能够支持每个分区的独立编程、擦除或读取操作。这个概念可以通过允许每个区块的自主操作来达到其逻辑极限。

设计师们现在对如何处理边读边写有了许多选择。可以容忍一些中断延迟、数据吞吐量和总线利用率相对较低的系统可以使用标准的单分区闪存来节省元件成本。对中断延迟容忍度较低或具有较高吞吐量和总线利用率的高性能系统可以考虑使用双分区闪存器件，以减少对性能下降因素的暴露。

写入保护。随着闪存设备在系统中被写入，由于意外写入闪存而导致的数据损坏的机会变得有可能。这些事故的一些来源是嘈杂的系统或错误的软件。为了帮助减少闪存对这些事故的脆弱性，已经开发了两种通用的写保护方法。

一种方法是以硬件为中心的，包括在预计不对闪存进行写入时关闭*V*pp 编程/擦除电压源引脚。由于闪存的内容在没有编程电压源的情况下不能被改变，闪存被屏蔽了，不会被无意中写入。

另一种方法是以软件为中心的，包括延长启动改变闪存阵列内容的操作所需的命令序列的长度。通过延长握手序列，由随机系统噪声产生的任何写入序列将更不可能启动一个编程或擦除操作。

区块锁定。随着Flash成为嵌入式系统设计中代码和数据的中心存储库 随着Flash成为嵌入式系统设计中代码和数据的中心存储库，人们对存储在Flash设备中的信息的安全性的担忧正在增加。

除了广泛的写保护方案外，还需要额外的保护，因为闪存有可能被CPU上运行的任何代码编程或擦除。因此，设计者希望有能力用各种安全级别来保护这些信息。区块锁定是添加到闪存设备中的一个功能，以满足这一需求。

最早的区块锁定方法之一是英特尔第一款启动块闪存设备上的硬件控制启动块。这种方法使用一个硬件引脚来控制闪存中第一个块的锁定状态，其中包含了CPU的启动代码，其目的是为了在闪存的其他部分被破坏时保护关键的启动和重新编程代码。然而，随着今天复杂的应用程序将更多的代码和更多的数据存入闪存，需要在锁定方案中增加灵活性。

最新的锁定方法在更先进的闪存设备上可用，有时被称为即时独立块锁定，它以两个级别的保护独立地保护每个块，并且可以在没有延迟的情况下进行更改。第一级保护对于保护用于数据存储的块是很有用的。这个级别可以防止对锁定的块进行编程或擦除。由于数据的写入相对频繁，这个级别允许只用软件解锁和重锁，以便在需要访问区块时，实时快速地改变锁定。第二级保护，称为 "锁定"，对于保护用于代码存储的块很有用。由于代码的更新并不频繁，但对系统运行却更为关键，因此锁定包含了硬件控制。一旦一个块通过软件命令进入锁定状态，为更新而解锁需要改变硬件输入引脚。这一额外步骤为存储在闪存中的代码段提供了额外的保护。

***2.1.5.2 软件。一个关键的角色。***虽然在物理上不是Flash设备的一部分，但软件是每个使用Flash的系统的重要组成部分。因此，本节将把它作为Flash的一个属性。事实上，随着应用程序以越来越复杂的方式使用Flash，越来越复杂的软件是实现这些应用程序的关键，在未来，曾经在软件中实现的功能可能会在Flash硬件本身中实现。

软件的需求各不相同，从最基本的将Flash设备置于适当模式的算法，到最复杂的媒体管理器，协助管理Flash设备并提供完整的文件系统功能。一个系统将需要部分或全部的功能，这取决于它的具体要求和需要。本节将分别探讨Flash软件的这些方面。

为什么闪存需要软件？闪存有许多独特的特性，需要用软件来编程或擦除闪存设备上的信息。在任何需要向闪存设备写入信息的系统中，软件是必不可少的。所有闪存的固有特性，无论其结构或包装如何，都是如此。闪存与其他存储设备不同，它提供字节可编程性、块擦除和非挥发性。

一个闪存设备在 "读 "的状态下开机，需要一系列的命令来使它进入另一个状态，如 "编程 "或 "擦除"。闪存中的一个位可以从擦除状态（一般为 "1"）变为编程状态（一般为 "0"），但需要进行块擦除，将 "0 "变为 "1"。

必须使用特殊软件来修改存储在闪存设备上的小块信息。此外，在发生性能下降之前，闪存对一个块的擦除次数有限制。这通常超过100,000次擦除，但必须在系统中加以考虑，以确保最佳操作。

闪存的基本软件。基本的Flash算法允许用户执行一些功能，如对一个字节或一系列字节进行编程，擦除一个块或一系列块，读取设备ID，或读取部件的状态。如果没有这些算法，用户只能从闪存设备中读取信息（代码或数据），因为它的电源已经处于读取状态。这些算法中的每一个都是针对正在使用的设备的，尽管大多数制造商倾向于在他们的产品系列中使用许多相同的程序（例如，所有英特尔设备的程序例程通常是相同的）。

至少，要将信息编程到Flash设备，读取、编程、擦除和系统软件必须使用状态算法。例如，要把一个人的电话和地址信息写入闪存设备，必须先调用一个程序例程，向闪存设备发送程序命令，然后把信息写入闪存中的所需位置。在每个程序命令之间必须调用状态例程（向Flash设备发送 "状态 "命令），以确保数据被正确写入设备。一旦数据被写入，必须向设备发送读命令，以便再次从设备中读取信息。如果最初闪存设备中没有足够的可用空间来写入这些信息，必须先调用 "擦除 "命令来擦除一个闪存块。图2.4是一个调用顺序的例子。

这些基本的Flash算法只是向Flash设备发送一个命令，使其进入适当的状态。然而，这些基本算法对于系统中的闪存设备的运行是至关重要的。

Main()

{

if (!(space\_available)) {

Erase\_block (block\_to\_erase);

}

Program (beginning\_address, data, number\_bytes\_to\_program);

Read();

}

Program (beginning\_address, data, number\_bytes\_to\_program)

{

Flash\_address = beginning address;

while (number\_of\_bytes\_to\_program)

{

Send\_command (Program\_command, Flash\_address, data\_byte);

while (Flash\_status = = not\_done) {

wait;

}

number\_of\_bytes\_to\_program--;

Flash\_address++;

}

}

图2.4.基本算法调用序列示例。

更高

级别

软件

为

闪存。

习惯于使用其他

允许直接覆盖内存位置的内存设备，如RAM或EEPROM，在使用闪存时可能会感到奇怪。他们通常习惯于将一块数据放在内存地图的特定地址上，并能在同一位置修改这块数据。然而，当使用闪存时，就地修改是不现实的，因为整个块的内容必须被备份到一个临时的位置，然后整个块被擦除以准备重写的位置。这是不现实的，也是对资源的浪费。相反，更高级别的软件（通常称为媒体管理器）对于优化闪存的使用是必要的。

媒体管理器所做的最重要的事情是管理信息的存储和回收旧的、无效的数据所使用的空间。如果没有这个功能，每一个存储在设备上的信息都必须使用一个闪存块（大小为8到128千字节），无论其大小如何。这通常是非常浪费的，因为大多数存储在闪存中的信息都比8到128千字节小得多。一个强大的媒体管理器可以通过几种不同的方式使这个块的大小对系统透明，但大多数使用两种方法中的一种--虚拟阻断或链接列表方法。

虚拟阻断。 在虚拟块中，系统通过在查找表中查找参数号来访问数据，查找表会返回闪存中每块信息所存储的物理位置。当使用虚拟块时，所有的信息都是以碎片的形式存储的，可以放入一个或多个虚拟块。例如，一块大于一个虚拟块的信息将跨越多个虚拟块。这些信息在闪存中可能是连续的，也可能不是连续的，所有的虚拟块都是一样的大小。见图2.5，它说明了虚拟块的使用。

链接表。当使用一个链接列表时，系统使用一个查找表，但这个表只指出第一条信息的位置。每条信息都包括一个头，它指向下一条信息的位置。每块内存的大小不一定相同，其中一块可能是

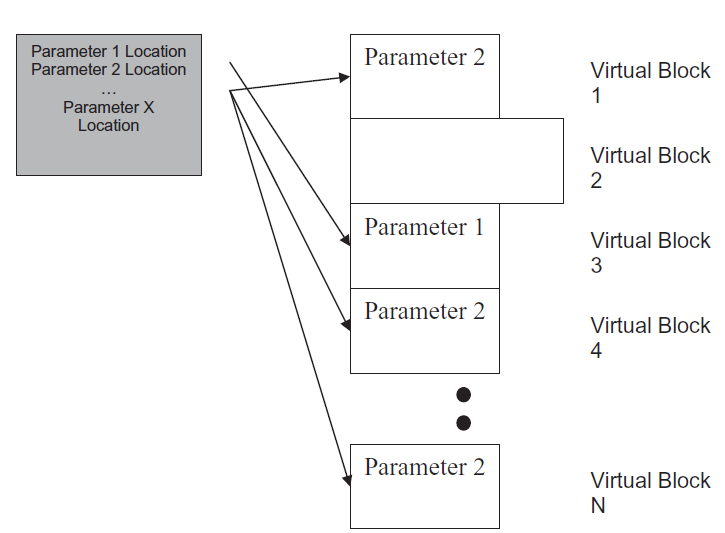
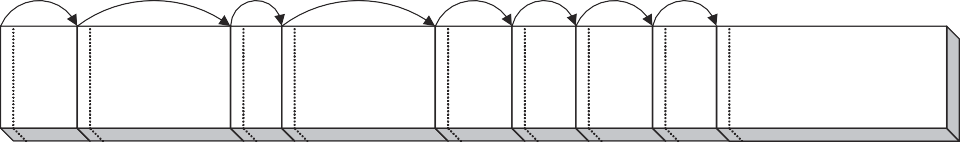




图2.5.虚拟阻断媒体管理器的例子。



1

2

3

4

5

6

7

8

9

图2.6.链接列表媒体管理器的例子。

跨越多个Flash擦写块。见图2.6，它说明了链接列表的使用。

高级媒体管理功能。介质管理器还可以实现高级功能，如再生或磨损平衡，这些功能超出了标准闪存驱动器的能力。

回收是媒体管理器的一个基本功能，因为它通过擦除块使部分闪存再次可用。当信息在闪存中被更新时，旧的信息不会被立即擦除（因为你只能擦除一个闪存块），而是被标记为 "无效"。当一个块中有足够多的位被标记为无效时，媒体管理器将回收该块。这种管理数据存储的方法允许在需要回收和擦除之前，在一个块中存储和更新许多数据，这增加了闪存设备的有效块循环。回收闪存块的必要步骤包括将有效信息从要回收的块中复制到新的块中，更新用于将参数数字转换到内存中的物理位置的表，最后擦除旧块。在这一过程中，必须特别注意断电恢复问题。如果在回收过程中系统断电，如果不使用适当的断电机制，就有可能丢失信息。

磨损平衡在任何将信息写入闪存设备的系统中都是有用的。如果一个媒体管理器没有意识到磨损平衡的需要，一个块可能会被专门用于一个特定的信息。这可能会提前耗尽闪存设备，特别是当某条信息被快速更新时。一个强大的媒体管理器会对每个区块被擦除的次数进行统计，并会偶尔移动数据，使正在快速更新的数据被移动到一个没有被擦除多次的区块。一般来说，磨损平衡算法将确保被擦除次数最多的区块和被擦除次数最少的区块的擦除次数保持接近。这就延长了闪存达到其循环规格的时间。

Flash归档系统。最后，最复杂的Flash软件形式是一个文件系统。编码系统不仅使Flash的大型擦除块对系统透明，而且允许系统通过文件名访问数据，并写入、修改和删除这些文件。一般来说，一次可以打开多个文件。闪存系统可以使用标准化的转换协议，如FTL（闪存转换层），将磁盘式闪存分配表（FAT）的文件扇区转换为线性闪存中的位置，或者闪存系统可以使用专有方法来创建这种转换。通过一个固件升级系统，对存储在闪存上的文件的请求在系统看来就像文件存储在一个标准的磁盘驱动器上一样。编码系统除了具有文件功能外，还包括媒体管理器的所有功能。

Flash软件的未来。闪存软件继续扩大闪存的使用范围并为系统增加价值。通过允许同时执行代码和存储数据，闪存软件已经帮助手机原始设备制造商（OEM）减少组件数量。这样一来，手机现在重量更轻，体积更小，而且电池寿命更长。此外，同步代码执行和数据存储允许在手持式个人电脑（PC）、微型笔记本、智能手机、网络电话和机顶盒中实现同样的组件数量减少。

用于手机的Flash软件的进步将有助于互联网的浏览和最终的空中代码更新。机顶盒将受益于Flash软件的改进，这将实现快速、可靠的代码更新，而不需要多余的内存。最后，所有使用闪存进行文件存储的应用程序都可以受益于闪存软件，不仅用于文件存储，而且还用于启动代码。

随着复杂的软件媒体管理器的使用变得普遍，某些通常在软件中实现的核心功能可以通过将其功能拉到Flash硬件本身来加速。例如，软体媒体管理器经常进行 "发现 "操作，在闪存内容中搜索特定的数据。这包括通过内存总线从闪存中读取数据，将其与CPU中的目标数据进行比较，然后循环操作直到找到数据。一个更快、更有效的方法是将其纳入闪存芯片中，这样可以节省总线和CPU带宽。

***2.1.5.3 包装的作用不断扩大。***多年来，封装在闪存市场上是一个事后的想法。闪存设备的硅片才是最重要的，而硅片被放在一个有足够引脚的封装中，以提供所有必要的信号。尖端技术，如DIP（双列封装）和PLCC（塑料引线芯片载体）是很好的解决方案，无论是在无处不在的机器编程器中，还是在原型开发板的插座中。此外，它们为调试提供了方便的引线。时至今日，日本的手机整机重量为60克，比过去的大引脚数DIP器件大不了多少。

原始设备制造商开发小型产品的持续压力转化为对硅供应商的另一种压力，即推出更小、更轻、更薄的封装。

闪存的典型封装。在闪存的早期，器件通常采用双列直插（DIP）和塑料引线芯片载体（PLCC）封装。DIP是最后一种印刷电路板（PCB）通孔封装，而PLCC则开始了表面安装技术（SMT）的实践。后来，这些类型的封装被称为小轮廓封装（SOP）的表面贴装封装系列所取代，最近又出现了球栅阵列（BGA）型封装。图2.7中显示了一些最常见的闪存封装。

多年来，SOP作为一个成熟的行业标准，已经获得了很大的发展，主要是由于大大减少了设备所消耗的电路板空间。SOP系列通过在相同的封装中提供针脚兼容的密度来提供灵活性，使其更容易实现扩展代码，而不需要重新设计PCB。SOP系列的鸥翼式引线和双线设计与传统封装相比，SOP系列有更多的改进。



图2.7.典型的Flash存储器封装。

例如，与其他PLCC封装相比，SOP系列提供了更好的铅检测、电路板布线和焊点。

SOP系列包括薄型小轮廓封装（TSOP）、收缩型小轮廓封装（SSOP）和塑料小轮廓封装（PSOP）。TSOP的引线间距为0.5毫米，最大封装高度为1.2毫米，而SSOP和PSOP的引线间距分别为0.8和1.27毫米，最大封装高度为1.9和2.95毫米。

TSOP是一个多供应商的工业标准小尺寸封装。TSOP提供了最简单的闪存密度升级途径。TSOP支持多种密度，对于以尺寸和便携性为主要设计考虑的复杂功能，如手机、PC BIOS、网络、国际个人电脑存储卡协会（PCMCIA）卡、大容量存储、传真/调制解调器、街机游戏和打印机，是理想的选择。

SSOP坚固耐用，易于使用，并支持中高密度。它特别适用于那些必须承受极端温度和要求高可靠性的应用。它最适合于要求最严格的应用，如汽车、蜂窝基站和工业嵌入式应用。

PSOP具有与SSOP相同的高可靠性特征，支持中低密度的产品。它特别适用于必须承受极端温度的应用。它最适合用于医疗和工业嵌入式应用、打印机服务器和传真/调制解调器的办公室设置，以及电信交换机和网络。

芯片规模和球栅阵列封装。基于PCMCIA卡和其他小尺寸系统的闪存应用，如计算机和手持通信（移动电话、个人通信服务（PCS）、传呼机），每天都在变得更加紧凑。此外，较大的应用，如电话交换系统和PC网络集线器，正试图在同样大小的盒子里增加更多的功能和内存密度。这两个领域都在推动对一种新型的超小型封装的需求：芯片级封装（CSP）。一个不超过芯片尺寸20%的封装属于CSP的范畴。

不断缩小的消费产品（如手机）的市场正出现爆炸性增长和激烈的竞争。几乎所有的手持式消费产品似乎正在以非常快的速度缩小。

通往亚手持式产品的竞赛正在进行，而竞赛的评判标准是毫米、克和额外功能。同时，强大的竞争力量正在给制造商带来压力，要求他们降低这些产品的成本。

新设计或重新设计的消费产品几乎每天都在推出，而产品设计周期正在缩短，以满足这些加速推出的周期。在过去，积极进取的公司可能会同时推出秋季和春季的产品，但在这种新的环境下，每个季度都推出新的产品是很常见的。整个芯片级封装解决方案市场正在看到这种环境的涟漪效应。供应商正看到组件的阶梯式需求增长。制造商和供应商的机会是巨大的，但为实现这一新的消费产品浪潮而供应元件的挑战也是巨大的。

此外，从传统的引线式封装到BGA类型的封装已经发生了巨大的转变。这种转变的原因之一是由于缺乏共面性和弯曲的引线问题以及BGA封装的自对准特性，抵消了微小的元件放置误差，从而提高了PCB组装的产量。

有几种类型的BGA封装可以满足各种应用的需要。这些封装的结构从Tessera公司开发的μBGA封装（使用弹性体和聚酰亚胺胶带插接，硅芯片的背面暴露在顶部）到传统的塑料模制设备（使用层压板或聚酰亚胺插接）不等。这些包装也有不同的间距（球之间的距离）。最常见的间距是1.0、0.8、0.75和0.5毫米。一些公司还将集成度提高到一个新的水平，将一个以上的硅片结合到一个封装中，通常称为堆叠式CSP封装。例如，闪存正在与逻辑和SRAM结合在一个封装中。

# 2.2 代码存储

代码存储是闪存的第一个用途，并且仍然是闪存的一个大客户。代码存储的应用范围很广，因为所有类型的电子设备--从个人电脑、移动电话、激光打印机、网络系统、全球定位系统（GPS）接收器、生物医学设备、邮政仪表到大型机器，如汽车、飞机和农业机械--都使用闪存进行代码存储。几乎所有类型的基于微处理器的系统都使用闪存。在这些不同的应用中，出现了两种基本的使用模式，涵盖了大多数的代码存储应用：原地执行（XIP）和存储和下载（SnD）。大多数应用程序使用SnD或XIP，尽管一些设计已经发展到使用两者的组合。

在本节中，我们对这两种使用模式进行了区分，讨论了每种使用模式的好处和设计上的考虑，并提供了这些使用模式在今天的应用中如何实现的例子。在本节的最后部分，我们描述了一些由于闪存技术而产生的未来应用。

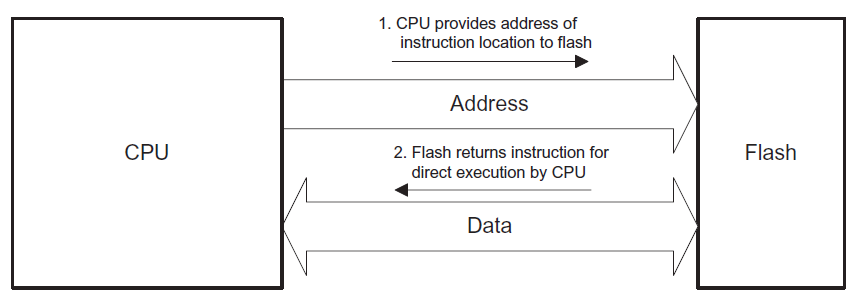


图2.8.XIP模型-CPU直接从闪存子系统执行程序代码。

# 2.2.1 就地执行

就地执行，或称XIP，是指可执行代码存储在闪存中的使用模式，系统微处理器直接从闪存子系统中获取指令进行执行。XIP的例子包括工业控制和生物医学设备中的嵌入代码存储，个人电脑中的BIOS代码存储，以及汽车中的发动机和传动系统控制（图2.8）。在这些应用实例中，有几个市场要求是关键的考虑因素，可以从XIP使用模型中受益。虽然最重要的属性和最不重要的属性的相对排名在不同的应用中可能有所不同，但每一个都是满足成本、性能和可靠性系统要求的关键。

***2.2.1.1 XIP应用的相关Flash属性。***XIP应用关心的主要闪存属性是读取性能、总线宽度和代码完整性。虽然在这些类型的应用中使用了其他闪存属性，但它们与核心的XIP功能没有直接关系。例如，一个闪存设备的系统电压写入能力、写入状态机和阻塞都能实现代码更新，但相对于核心的读取功能，这些更新并不频繁。

读取性能。XIP要求的主要属性是读取访问时间或读取性能。为了让系统的微处理器直接从闪存中执行代码，内存必须足够快地输出所要求的代码，以保持指令队列的完整，而不造成影响系统性能的延迟。通常情况下，闪存的读取性能与工作电压直接相关；也就是说，工作电压越高，读取时间就越快。一个完全额定的5-V异步闪存设备的典型访问时间是60至65纳秒。这足以支持零等待状态的读取性能，其时钟速度可达15-MHz。

一般来说，有两个因素驱动基于微处理器的系统的性能：时钟速度和工作电压。有趣的是，微处理器的性能通常与工作电压成反比，因为电压越低，微处理器内的晶体管开关速度就越快，因为逻辑电平之间的电压差较小。随着微处理器性能的提高，存储器子系统所需的读取性能也随之提高。然而，在闪存中发生的效果正好相反；工作电压越低，

闪存的读取性能就越慢。这种微处理器和闪存的性能与工作电压的对比给XIP模型带来了固有的性能限制。当闪存的原始速度不再跟上微处理器的性能时，XIP模型作为一个完整的内存子系统就变得不那么可行了，至少在使用异步闪存器件时是这样。解决方案是使用页面模式或同步突发读取界面。这是一个重要的发展，值得在第2.2.1.3节中深入讨论。2.2.1.3.

总线宽度。在为XIP应用选择闪存时，总线宽度是另一个关键属性，因为存储设备的总线宽度应与CPU的总线宽度相匹配，以获得最佳性能。闪存设备一般有8位或16位的宽度，多个设备可以平行排列以建立所需的总线宽度。交织是另一种设计技术，有时用于提高存储器的性能，以乒乓的方式交替从两个器件中读取。然而，由于页/突发模式的闪存设备，这种技术的使用正在减少，它将同一想法的优化版本直接集成到芯片中。

程序代码的完整性。代码存储的另一个非常重要的属性是代码安全。在今天的市场上，几乎所有的系统都需要绝对的代码完整性。不能有一个比特出现错误，否则系统可能无法启动。虽然错误检测和纠正可以在SnD使用模式中实现（如硬盘驱动器），但XIP必须保证零位错误，以确保可靠运行。此外，许多闪存设备有写保护功能，以防止意外写入内存图的代码存储部分。这些硬件控制输入必须由外部驱动以允许代码更新。最后，一些闪存器件有专门的程序电压输入（*V*pp ，与*V*cc 输入分开），可以切换到地，以防止非故意的写操作改变存储器阵列内容。

NOR型闪存非常适合于代码存储，因为它同时满足了读取性能和安全要求。NOR型闪存具有快速的随机访问，这意味着每一个新的地址，甚至是不连续的地址都会在最大访问时间内产生数据。NOR也有极高的可靠性，不需要错误检测和纠正，并且有10年或更长时间的数据保留。

相比之下，NAND型闪存并不适合用于代码存储，原因有二。首先，NAND的随机存取速度非常慢，大约为5到10μs（与NOR的60ns相比）。 其次，NAND可能需要错误检测和纠正，因为一些比特可能会随机改变状态。由于代码存储需要绝对的位完整性，NAND型闪存实际上最适合用于数据存储。

***2.2.1.2 用于XIP的Flash的优点和局限性。在***XIP中使用Flash的优点是干净的内存模型。代码存储在同一个地方，更新到同一个地方，并从同一个地方执行，消除了冗余的内存，如果代码存储和执行是在不同的内存中进行的，就像存储和下载应用那样。通过减少内存冗余，XIP模型减少了系统中需要的RAM数量。

此外，在系统开始运行之前，代码不需要从非易失性存储器（闪存或磁盘）复制到RAM。因此，系统可以更快地启动和运行，这正是用户所希望的。

闪存对于XIP的限制是速度和大小。正如第2.1.5.1节所讨论的，在XIP环境中，Flash的读取性能会直接限制CPU和系统的性能。然而，现在有突发和页面模式的读取来解决这个限制。由于最新的同步闪存设备能够在66MHz下进行零等待状态的操作，很少有系统会受到闪存读取性能的限制。随着软件开发商增加越来越多的功能，代码大小也在不断增加。使用XIP，所有的代码都需要包含在Flash中。一些最新的闪存设备，如英特尔StrataFlash手机存储器，在一个芯片中包含64Mbytes，所以大量的代码可以存储在一个非常小的电路板空间中用于XIP。

***2.2.1.3 高级读取接口。页和突发模式。***传统的闪存采用了类似于SRAM的异步接口。在这种情况下，CPU向闪存提供一个地址，然后在一段时间后（称为访问时间），闪存将数据返回给处理器。通常情况下，5-V闪存设备的访问时间从55到120纳秒不等；而3-V元件的访问时间则从75到150纳秒不等。低密度元件的访问时间更快。在16-Mbit的密度下，3-V的存取时间在75到150 ns之间。在这种访问速度下，大多数高性能的CPU必须等待数据。例如，在33MHz下运行，一个使用80ns闪存的典型系统将不得不插入两个等待状态。在这两个周期中，处理器处于空闲状态，等待信息。

异步读架构非常适合非突发处理器，因为非突发处理器一次检索一个数据。然而，一个高性能的突发CPU在从内存子系统获取信息时，可以访问一个 "组 "的数据。这种类型的内存访问，主要用于代码XIP应用，对闪存提出了很高的读取性能要求。当突发处理器与异步存储器一起使用时，系统性能会进一步下降。在一个突发周期中，初始内存访问至少需要两个时钟周期，包括一个地址阶段和一个数据阶段。然而，在处理器读取第一条数据后，数据可以在每个时钟沿被访问。再次使用一个80ns的异步闪存组件，一个典型的系统将有一个2-3-3-3的等待状态程序（见图2.9）。

为了缓解使用异步闪存设备时逻辑和内存之间的性能差距，高性能系统不得不依靠复杂和冗余的内存配置，如内存交织和代码阴影来克服闪存的读取性能缺陷。由于这些技术增加了系统成本并降低了系统的可靠性，因此它们不是很好的解决方案。

显然，性能上的差距必须通过一个干净的架构解决方案来解决。粗暴的解决方案，即加快异步闪存的性能，是不可能的，因为用于使CPU更快的标准技术，如降低工作电压和增加晶体管密度，并不适用于闪存。与逻辑电路不同，较低的工作电压和较高的密度往往会对闪存的读取性能产生负面影响。一部分闪存的传感逻辑由模拟电路组成，它不能有效地随电压变化而变化。

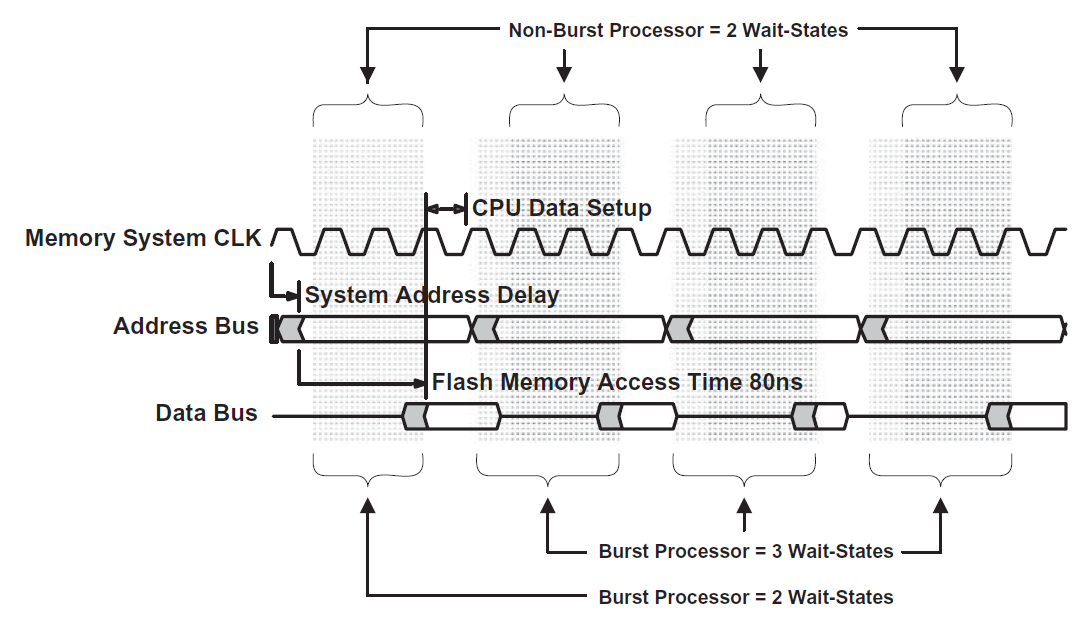


图2.9.突发事件与非突发事件处理器的等待状态示例。

使得在低工作电压的趋势下，很难大幅提高访问时间。密度对闪存设备的读取性能也有不利影响。随着闪存密度的增加，内部电容的增加减缓了随机访问的速度。

与其用蛮力解决这个问题，不如通过改变闪存的读取架构来解决XIP性能瓶颈问题。其他存储器技术是如何处理读取性能问题的？

为了跟上处理器的性能，其他存储器技术，如ROM、DRAM和SRAM已经实现了接口，通过利用处理器的突发协议来提高读取传输率。例如，为了解决计算机行业面临的性能问题，DRAM已经从快速页面模式发展到扩展数据输出（EDO），再到同步到直接DRAM接口。同样，随着时间的推移，ROM设备也发生了类似的转变。ROM制造商提供异步、页面模式和同步架构以满足不同的性能需求。今天，所有出货的ROM组件中有50%支持页面模式接口，而新的同步ROM设备正在被引入。所有这些创新都使用突发协议的可预测和顺序模式来帮助跟上处理器的性能。闪存正在遵循类似的接口和架构演变，以解决这些高性能的应用需求。

与页模式ROM一样，异步页模式闪存为突发CPU提供了一个高性能的非时钟内存解决方案。这种架构的随机访问速度与传统的异步闪存相当。然而，异步页模式以非常高的速度提供对后续信息的访问。通常情况下，页面访问时间等于输出缓冲器的速度--对于闪存来说是20到30纳秒。

同步闪存接口为突发CPU提供了最高性能的内存解决方案。通过查找数据的下一个字，同时输出目前的数据，

闪存的同步接口提供了足够的内存吞吐量来实现零等待状态的性能。这使得突发访问时间与异步页模式的访问时间相比进一步减少。

同步突发模式的闪存接口与同步DRAM（SDRAM）和伪静态RAM（PSRAM）非常相似，在某些情况下，可以通过PSRAM端口直接连接到内存总线。为了在同步模式下控制闪存设备，增加了两个新信号：地址有效和时钟。地址有效引脚通知闪存设备，地址总线上有一个新的地址，并开始一个新的突发周期。时钟信号使闪存设备与CPU同步。这些信号直接连接到处理器的存储器接口单元，以尽量减少系统胶水逻辑和系统运行延迟。

***2.2.1.4 未来的读取接口。***为了继续提高闪存的性能，将需要新的接口，这些接口的版本正在RAM领域出现，如双数据速率（DDR）DRAM和直接Rambus DRAM（RDRAM）。DDR接口类似于同步突发模式机制，但它在系统时钟的上升沿和下降沿输出数据，这在给定的总线频率下有效地使数据传输率翻倍。随着系统总线频率开始超过133-MHz的范围，接口将需要演变为源同步实现。在高频率下需要采用源同步方法，以尽量减少系统飞行时间（信号从一个组件到另一个组件所需的时间）。在这种接口下，用于锁存数据的信号（选通器）与数据一起被设置给接收者。有了源同步接口，可以实现约400MHz的频率。然而，这些增强型接口的实现取决于使用闪存的嵌入式处理器是否采用了高性能接口。处理器和存储器都需要支持优化的互连，以提高整个系统的性能。

***2.2.1.5 选择正确的接口。***应用需求正在发生变化，未来的闪存架构也会发生变化。异步闪存与非突发处理器配合良好。异步页面模式和同步突发模式的闪存设备有助于解决系统设计者今天在用突发处理器直接从闪存执行代码时面临的闪存性能问题。异步页面模式闪存是非时钟内存子系统和慢速运行频率的理想选择。同步突发模式的闪存为高CPU工作频率提供了高性能的解决方案。这三种闪存架构都有重要的特点，有助于满足不同应用的性能需求（见图2.10）。为了在未来推动闪存的性能，需要采用其他接口。

# 2.2.2 存储和下载

存储和下载（SnD）是指程序代码存储在非易失性介质中的使用模式，如Flash、ROM或磁盘，并在开机后下载到系统RAM中执行（图2.11）。由于非易失性存储器解决方案的读取时间通常比易失性存储器慢，如

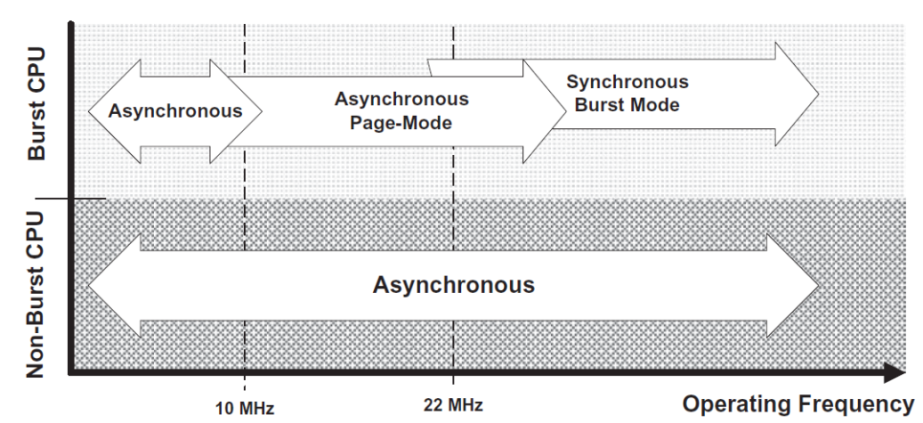


图2.10.每个总线频率的突发和非突发处理器的内存解决方案

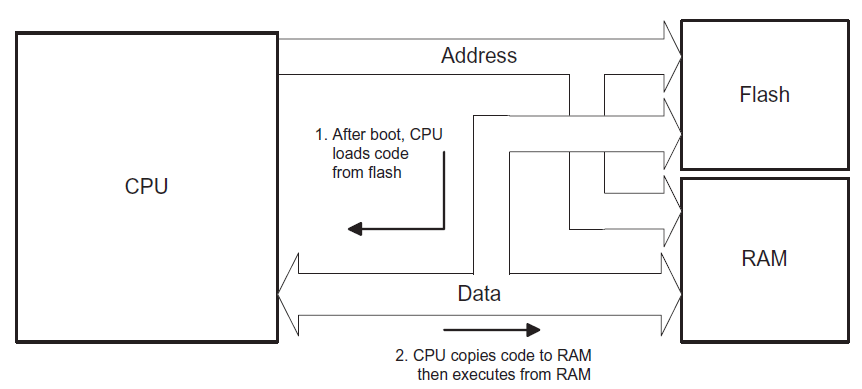


图2.11.存储和下载模型-程序代码被复制到系统的DRAM中，以便更快地执行

与DRAM或SRAM相比，SnD模型非常适用于需要更快的读取性能的系统。使用这种方法的应用包括数据通信网络、交换中心、网络路由器中的启动代码和操作系统代码，以及存储在电信交换机中的代码，仅举几例。一些应用选择SnD是为了满足其系统运行或历史的特殊需要。一个众所周知的例子是个人电脑的架构，它将BIOS代码存储在闪存中，将代码存储在硬盘上，但将其全部下载到RAM中执行，因为个人电脑的操作模式希望从RAM中执行，并要求有随机写入内存的能力。推动内存模型走向SnD的其他系统需求的例子是无线本地环路基站，它需要从其无线数据链路接收空中的代码更新。由于这样的基站必须运行代码来维护其无线传输协议以接收新的代码更新，它不能在向Flash写新的代码图像的同时从Flash运行该代码。因此，SnD是一个很好的时机。

***2.2.2.1 SnD的相关属性。***存储和下载的使用模式对闪存的要求很少。大多数内置在闪存设备中的功能和属性都集中在实现写入和可更新性上，而这些在SnD应用中并没有大量使用。

密度。闪存必须足够大，以存储系统中所需要的所有代码。从非常小的（8Mb）到非常大的（1GB）的存储器都可以满足所有的需求，而且，由于设备不需要对SnD进行线性映射，多个设备很容易被配置以提供额外的空间。然而，电路板空间和成本往往促使SnD应用倾向于最高密度的元件。

访问时间。根据系统的要求，访问时间对SnD应用来说可能重要，也可能不重要。一方面，由于代码将从RAM中运行，访问时间不是执行速度的限制因素。但另一方面，整个代码映像将在启动时从Flash中下载，所以更快的访问时间将加速需要快速启动时间的应用的启动过程。例如，使用SnD的数字广播视频摄像机将需要尽可能快地下载和运行代码，否则一个重要的新闻事件可能会被等待摄像机启动的摄影师所错过。

代码的完整性。虽然代码存储的完整性总是很重要的，但下载的代码在放入RAM执行之前总是可以通过纠错算法来运行；所以这实际上不是一个关键要求。因此，无论是NAND（带纠错）还是更可靠的NOR，都可以用来满足系统要求。然而，如果在硬件中加入纠错，会增加系统的成本，如果在软件中加入纠错，则会减慢启动时间。

# 2.2.3 就地执行与存储和下载的对比

XIP的主要好处包括成本、省电和减少组件。通过直接从闪存设备上执行代码，需要的DRAM更少。相反，SnD的论点是基于性能的；通过将程序代码从存储介质（Flash）下载到DRAM中执行，整个系统的性能提高了（假设DRAM的访问时间比Flash快）。 然而，SnD提出了一个冗余的内存模型，它有独立的内存子系统，Flash用于存储，DRAM用于执行。尽管许多XIP系统都有一定数量的DRAM，但在SnD模型中需要更多的DRAM，这就增加了成本。成本/性能优势是一种权衡，必须根据每个应用程序的性能要求和成本限制进行优化。

# 2.2.4 未来的代码存储应用

闪存技术的基本优势（包括非易失性、低功耗和高可靠性）使得功能丰富的小尺寸电池供电的应用成为可能。虽然蜂窝电话和手持GPS接收器没有闪存，但由于使用了闪存，两者都实现了更小的外形尺寸，更多的功能和更长的电池寿命。一些新的应用正在出现，它们也将利用闪存提供的技术优势。例如，闪存正被越来越多地用于汽车应用中。汽车中的硅含量正在迅速增长。虽然发动机和传动系统控制的程序代码存储多年来一直依赖闪存，但汽车应用中增长最快的部分是仪表盘。特别是，驾驶员信息系统正迅速成为大多数豪华汽车的标准。汽车电子供应商正在将蜂窝电话、汽车电子系统和汽车电子系统整合在一起。

在一个单一的紧凑型仪表盘系统中，有音响、GPS导航和无线数字数据传输。闪存是这些驾驶信息系统中程序代码和地图/路线数据存储的理想存储方案。由于其价格较高，豪华车更能够吸收新的驾驶信息系统的价位提升。随着时间的推移，随着这些系统数量的增加，价格的增加应随着规模经济的发展而减少，最终它们将成为低端汽车的选项，也许会像今天的汽车音响一样成为标准设备。

# 2.3 数据存储

所有的应用都需要存储一定量的数据，但今天的许多应用都需要大量的数据存储。这些应用可以分为两大类。第一类包括专门存储数据的设备，用于执行其功能。一个例子是网络路由器，它存储大量的地址表，并使用它们来发送网络数据包到正确的目的地。第二类设备的唯一目的是捕捉和存储来自外部世界的数据。一个例子是数字音频记录器，它存储音频样本供以后播放。在这两种情况下，数据存储介质必须是非挥发性的。

# 2.3.1 为什么使用闪存来存储数据？

在数字媒体的热潮中，属于第二类的应用正在迅速增加。此外，它们往往是便携式的，如数码相机或音频记录器。然而，在便携式或手持式设备中，除了振动、冲击、可靠性、功率和散热要求外，还有物理尺寸和重量的限制。这种类型的环境是使用闪存而不是旋转磁性介质进行存储的理想场所。

传统的硬盘驱动器比闪存使用更多的电力，产生更多的热量。此外，旋转介质更容易受到振动和冲击的损害，这使得它们在便携式或手持式系统中的可靠性降低。诸如移动电话和手持式组织者的应用使用闪存而不是硬盘驱动器，因为否则它们会很大、很重、很脆弱，而且电池寿命很短。另一个问题是磁盘驱动器的最小容量。硬盘驱动器的容量在不断增加，但却没有现成的小容量。常见的磁盘驱动器尺寸在千兆字节范围内，而闪存卡在一到数百兆字节范围内。闪存可能更适合应用的需要，只提供必要的容量。如果你的应用只需要2或100MB，那么这个容量的闪存卡可能比硬盘驱动器更合适、更可用。

# 2.3.2 架构决策

当设计一个利用闪存进行数据存储的系统时，首先必须决定闪存应该是可移动的还是不可移动的（常驻）。此外，必须在NOR型和NAND型闪存之间做出决定，因为两者在数据存储方面都有优势和劣势。

表2.3.区分嵌入式或可移动闪存的主要标准

嵌入式可移动闪存卡

* 希望将代码和数据结合在同一闪存芯片中
* 数据或代码经常被改变
* 由于物理空间的限制，无法增加可移动卡的插槽
* 需要有最低限度的现成的存储空间
* 系统不能容忍存储空间的意外移除（如可能发生在可移动卡上）。
* 存在重量限制。
* 数据的大小超过了嵌入式闪存中可以合理地本地存储的大小。
* 在系统之间移动数据是必要的。
* 终端用户需要在存储量方面具有灵活性。

***2.3.2.1 可移动与嵌入式。***不可拆卸的闪存存储是指永久嵌入系统中的闪存设备或设备组。(见表2.3。）嵌入式闪存的一个关键好处是可以实现代码+数据。由于大多数需要存储数据的系统也需要将代码存储在某个地方，将两者放在同一个芯片中是有意义的。例如，数码相机的代码存储可以有一个更大的闪存芯片，并在没有卡的情况下使用额外的空间来存储图片。这对终端用户来说，可以方便地进行快速的、计划外的快照。该设备可能无法承担必要的物理空间和重量来增加一个可移动卡槽。

可移动的闪存被放置在一张卡或其他携带装置上，这样它就可以从系统中移除，以便将数据传输到另一个系统，或与另一张卡交换以获得额外的存储。需要存储大量数据的应用，由于成本或尺寸的限制，无法在嵌入式闪存中存储，可能需要可移动闪存介质。使用多个可移动的闪存卡可以使系统用户拥有无限的存储容量。闪存卡允许向前兼容，因为它们提供了一个标准的系统接口。用户可以通过购买额外的卡或购买容量更大的闪存卡来增加其存储容量。一个常见的闪存卡应用是数码相机（图2.12）或录音机。这些都是闪存卡应用的好例子，因为在录制音频或视频时，用户不太可能有网络连接。

***2.3.2.2 用于数据存储的NOR或NAND。***与代码应用不同，数据存储应用不需要直接将存储介质（本例中为闪存）与CPU连接。因此，闪存的接口可以被抽象为高于处理器的硬件信号时间要求。CPU使用标准协议请求一个特定的数据，并以标准格式接收所需的数据。常用的抽象方法之一是先进技术附件（ATA）接口，这将在第2.3.4.1节进一步讨论。通过隐藏闪存设备的规格，这种抽象实现了两件事：使用闪存设备类型的灵活性和在传递给CPU之前对检索到的数据进行操作。

在抽象化的接口背后，存储介质可以是任何东西。例如，ATA接口被用作硬盘、软盘、有线网络、闪存、CD-ROM，甚至是无线网络收发器的接口。这一特性可以用来增加提供实际存储的设备类型的灵活性，因此可以使用NOR或NAND闪存。一个特定的应用要求可能需要NOR（快速随机存取，位可靠性）或NAND（快速顺序存取）的特定特性之一，但除此之外，设计接口以支持两种类型的闪存将合理地增加供应来源。

抽象的接口还允许存储子系统在将数据返回给CPU之前对检索的数据进行按摩。可以利用这种自由的两种方式是纠错和压缩。然而，压缩和纠错都会减慢数据的检索速度，因为算法必须在返回数据之前对其进行解压和修正。

纠错使用数学算法来纠正因闪存单元的退化或其他方式而改变的数据。一般来说，这些算法需要额外的比特与数据一起存储，以提供足够的冗余度来纠正错误。从统计学上看，闪存中的错误越多，就需要更多的额外比特来纠正它们。虽然这降低了有效的存储容量，但错误纠正可以掩盖闪存设备的缺陷，使其有可能容忍闪存设备的可靠性降低。在闪存方面，这有助于NAND闪存，因为NAND设备不能保证100%的良好位，允许使用低成本的设备。NOR闪存设备不能从纠错中获益，因为它们不需要纠错，因为目前市场上所有的NOR设备都能保证100%的良好比特。在非100%的NAND器件的低成本（部分被错误纠正所需的额外比特存储所抵消）和NOR闪存器件的完全可用性之间进行权衡。

压缩使用数学和符号算法，通过编码模式和重复，用比实际数据更少的比特表示数字数据。通过使用一个抽象的接口，数据可以在被存储到闪存之前被压缩，然后，当它被访问时，在被返回到CPU之前被解压缩。这使得一个有效的存储设备比实际的物理设备更大。如果NOR和NAND设备都能提供100%的良好比特，那么压缩应该同样有助于这两种设备。如果在NAND设备上使用纠错，额外的纠错位可能会抵消压缩带来的容量增益。

数据存储应用的底线是，抽象化的接口模糊了NOR和NAND之间的差异。它还可以实现额外的设计技术，如校正和压缩，在选择闪存技术之前，应仔细考虑其与应用性能要求之间的权衡。



图2.12.索尼Cybershot810万像素数码相机，带可移动闪存。(*索尼公司提供*。)

# 2.3.3 嵌入式闪存存储

尽管嵌入式闪存阵列也可用于代码或代码+数据的应用，但*嵌入式*一词最常指的是用于数据存储的阵列。

NOR和NAND闪存都可用于嵌入式存储。如果闪存要做代码存储（例如，启动代码），那么NOR的存储完整性和快速随机访问使它成为正确的选择。另一方面，NAND的快速编程和高密度使它更适合于纯数据存储。

对于需要捕获和访问数据但不需要在系统间传输数据的应用来说，嵌入式Flash是一个很好的解决方案。例如，传真机在发送或打印传真之前，可以使用嵌入式Flash来存储传入和输出的传真。在这种情况下，闪存提供了一个很好的解决方案，因为它的非挥发性可以防止在断电的情况下丢失传真。此外，它提供了固态的可靠性，没有移动部件。由于数据只通过电话线或纸张输出来传输，因此没有必要使用可移动存储。

在其他情况下，应用程序的物理尺寸可能排除了使用可移动闪存卡。例如，最小和最轻的MP3播放器只采用嵌入式闪存。

***2.3.3.1 嵌入式闪存的相关属性。***嵌入闪存的数据存储的相关属性与其他使用模式不同。因此，需求集中在将大量的存储放在一个小的物理空间里，以及长期安全存储和更新数据的能力。

密度

和

包装。大多数考虑添加嵌入式的应用

闪存需要一个相对较大的存储量，并希望在尽可能小的电路板空间内完成。密度变得很重要，因为更高密度的闪存芯片可以在相同的空间内容纳更多的存储，因为一般来说，新的芯片在相同的硅片面积上可以容纳两倍的存储。因此，对于阵列中给定数量的芯片，升级到更高密度的设备可以使存储容量增加一倍。出于这个原因，设计应始终为未来的升级做好计划，在相同的布局中，通过路由所需的额外地址线来支持更大的设备。

随着应用的物理尺寸的缩小，包装也变得很重要。应该考虑两个因素：包装尺寸和路由效率。显而易见，包装的尺寸直接关系到在一个特定的空间里可以装多少容量。然而，路由效率决定了这些封装在电路板上可以放置得有多紧密，并且仍然可以获得所有的信号路由。一些封装提供了特殊的引脚布局，使阵列中的多个器件能够紧凑地布线。

对称块。由于嵌入式闪存可能包含多个连接在一起的闪存设备，所有相同大小的对称块使内存管理更容易。不对称的设备可能在内存地图的中间引入不同的块大小，使管理更加复杂。因此，对称块状的闪存是嵌入式配置的首选。

循环。用于数据存储的嵌入式闪存与仅有代码的闪存相比，可能会产生更多的擦除周期，因为有可能出现持续的周转。

数据。例如，在数码相机中，当用户拍照、下载和擦除时，图片被不断地存储和删除。然而，在许多情况下，嵌入式闪存将通过软件数据管理器进行管理，如果设计得当，可以大大减少擦除周期数，因为在需要擦除之前，可以在一个块中添加多块数据。

区块锁定。与代码不同，损坏的存储数据一般不会对系统运行造成致命影响，但数据丢失仍然会使应用程序的功能失效。例如，一个将网络地址表存储在嵌入式闪存中的网络路由器，如果该表丢失，仍然可以启动，但它将无法在网络中执行其指定功能。由于这个原因，块锁定功能可以用来保护存储在嵌入式闪存中的数据不被损坏或意外删除。

软件。软件数据管理器在抽象闪存接口的细节方面很有帮助，这样系统应用就可以使用更高层次的调用将数据存储到闪存中，使闪存存储文件或模拟磁盘。数据管理器还可以实现复杂的功能，如磨损均衡，以提高嵌入式闪存的性能。

# 2.3.4 可移动媒体

从外部世界捕获或存储数据的应用正在迅速增长，这一点从消费电子零售商货架上众多的数码相机、数码录音机和MP3播放器可以看出。这种使用模式最突出的特点是，终端用户所需的存储量确实无法量化，因为要采集的图片数量是无限的。因此，需要有可移动的媒体，以便终端用户可以决定多少存储量是合适的。然而，这一要求并不包括对内置闪存芯片的需求。这些设备中的大多数都有CPU，需要代码存储闪存，它可以扩展为代码+数据存储，以提供一定量的内置数据空间。

下文将讨论对可移动Flash很重要的问题以及市场上的一些选择。

***2.3.4.1 可移动闪存的相关属性。***为系统设计选择一种可移动的闪存介质，会带来一些尚未讨论的新属性，但请记住，这些卡内是闪存设备，受到与分立元件相同的限制。下面几节将专门讨论与可移动闪存卡有关的属性。

NOR和NAND。可移动的闪存卡是闪存应用的一个很好的例子，它可以是NOR型或NAND型闪存。基于NOR的闪存最适合于代码或代码+数据的存储，而NAND存储器则最适合于只存储数据。

闪存卡的性能是一个重要的考虑因素。性能可以通过几个不同的指标来衡量。NOR和NAND存储器在随机存取读取性能、顺序存取读取性能、写入性能和功耗方面各有优缺点。

根据应用中最重要的性能指标，将有助于确定哪种可移动闪存介质最适合。

接口。线性和ATA。存在两种类型的闪存卡接口：线性和ATA。线性闪存在其内存地图中直接寻址，它的寻址方式与内存组件本身基本相同。例如，系统将通过进入第85个（55个十六进制）内存位置来访问第85个字/字节。闪存卡中不需要转换电路，这就降低了卡的制造成本。另外，线性闪存允许通过16位宽的数据总线进行快速数据传输。代码可以通过线性PCMCIA卡就地执行（XIP）。

像Flash Translation Layer（FTL）这样的软件可以用来管理线性闪存卡。FTL模拟旋转的磁性介质（扇区模拟）。FTL最适合模拟基于扇区的可移动驱动器。

另一种接口方法是使用ATA（高级技术附件）/集成设备电子（IDE）接口连接闪存卡。即将讨论的Com-pactFlash和SmartMedia技术就采用了这种方法。ATA（IDE）接口通过使用简化集成的工业标准接口，减少了开发时间。然而，必须使用一个控制器，这增加了成本。CompactFlash卡中有一个控制器，增加了卡的成本，而SmartMedia卡需要在读卡器中安装一个控制器。最终的结果是更简单的集成，但这要付出一定的代价。此外，这些卡不能直接从卡中执行代码；代码必须在执行前下载到内存中。

可移动卡的形式因素。一个重要的考虑因素是闪存卡的形式因素。有多个供应商提供了多种形式的因素（图2.13）。每个特定的标准都倾向于NOR型闪存或NAND型闪存。PCMCIA闪存卡、CompactFlash、SmartMedia、SD存储卡和多媒体卡都是基于公认标准的不同形式因素。每一种都有自己的尺寸、重量和其他优点或弱点。每个特定应用的需求将决定哪种闪存卡的形式因素最适合。

插入方法。 插入方法可能是选择闪存卡的一个考虑因素。PCMCIA卡、CompactFlash卡、SD存储卡和SmartMedia卡都是纵向插入系统侧面的一个小槽中，就像把软盘插入PC一样。这个插槽只需要与卡的宽度和厚度相同。系统外部所需的表面积非常小。



图2.13.各种卡的形式因素的例子。

数据交换。为某一特定应用选择闪存卡的另一个考虑因素是它与世界其他地区的连接方式。从PCMCIA卡到CompactFlash和SmartMedia的适配器很容易找到（图2.14）。这些无源适配器卡使得通过PCMCIA卡槽与任何闪存卡之间的数据传输变得很容易。此外，还有一些适配器可以让智能媒体在标准软盘驱动器或USB驱动器中被读取。

***2.3.4.2 主要卡片标准。***在市场上有几种相互竞争的标准。每种标准都有其优势和劣势，这将决定哪种标准是特定系统设计的正确选择。

PCMCIA又名PC卡。基于PCMCIA的闪存卡有68针的连接器

在两行34个引脚中。这些闪存卡通常以线性闪存或通过ATA（IDE）接口访问。线性闪存通常以NOR闪存为基础，而ATA接口卡通常以NAND存储器为基础。PCMCIA卡是一个被广泛接受的开放标准，可以与几乎所有笔记本电脑和许多其他系统兼容。PCMCIA与其他标准组织合作，使其很容易建立与其他闪存卡标准的适配器。因此，从PCMCIA闪存卡到CompactFlash和SmartMedia，都有现成的适配器。更多的信息可以在[http://www.](http://www/) pcmcia.org找到。

PCMCIA卡（图2.15）的一个优势是其物理尺寸。较大的卡可以容纳更多的闪存设备，这意味着它们具有更大的存储能力。然而，在小型手持设备中，较大的尺寸可能是一个障碍。

SD存储卡。SD（安全数字）卡是一种通用的扩展卡标准，包括内存和I/O支持。它在很大程度上

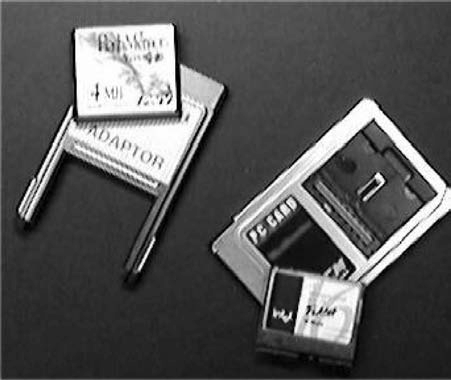


图2.14.闪存卡适配器到PCMCIA。

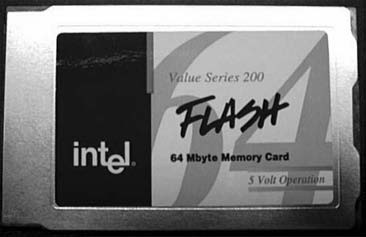


图2.15.带有线性闪存的PCMCIA卡。

比PCMCIA卡小。与PCMCIA不同，这种标准没有任何可以弯曲的针脚，而是在卡上有九个暴露的痕迹。SD卡已成为许多外形较小的手持设备的流行标准，并已成为移动电话市场上可移动存储器的事实标准。除了3g的SD存储卡外，还有一种缩小的标准可供选择。这种更小的版本，即MiniSD，重量只有1克。

紧凑型闪存。CompactFlash是基于一个开放的标准，在电气上符合PCMCIA ATA接口标准。这些卡通常以NAND闪存为基础。它们有一个板载的控制器来管理纠错和控制接口。该控制器增加了额外的成本，但让闪存卡在系统中显示为一个ATA（IDE）驱动器。这使设计变得简单，并能快速进入市场。这些卡的快速连续写入与小的外形尺寸相结合，使这些设备非常适用于数码相机。更多信息可以在互联网上的CompactFlash协会的网站上找到，网址是：http://www.compactﬂash.org。

SmartMedia（SSFDC）。SmartMedia或固态软盘卡（SSFDC）是另一种针对手持系统的可移动闪存卡标准。 SmartMedia卡是非常轻的设备。它们每个大约2克。SSFDC卡使用NAND存储器作为其存储介质。SmartMedia卡的设计只允许使用一个芯片，因此卡的容量仅限于一个芯片的容量。智能媒体主机接口芯片也使用ATA接口标准，允许以简单的设计快速进入市场；然而，其代价是系统成本的增加。NAND的快速连续写入与小的外形尺寸相结合，使这些设备非常适用于数码相机。更多信息可以在互联网上的SSFDC论坛主页上找到[：http://www.ssfdc.or.jp/english/。](http://www.ssfdc.or.jp/english/)

多媒体卡（MMC）。1997年11月推出的MultiMediaCard是一种极小的闪存卡，针对空间有限的应用，如移动电话、MP3音乐播放器和数码摄像机。它的重量不到2克，大小与邮票相当（高32毫米，宽24毫米，厚1.4毫米）。卡的接口是一个7针的串行接口，这使得卡的插座小而简单，但排除了MMC的代码执行。关于MMC的更多信息可在[http://www.sandisk.com/。](http://www.sandisk.com/)

# 2.4 代码+数据存储

代码+数据是唯一一个围绕着闪存的优势而发展的闪存使用模式。(见图2.16。）随着时间的推移，大多数被添加到Flash的功能都可以被看作是代码+数据使用模式的发展步骤。闪存架构的每一个进步都成为代码+数据整合的下一个步骤的基石。当早期的闪存设备被开发出来时，该技术的支持者主张它是内存整合之路的最终目的地，因为它具有系统内更新、快速读取和低成本的能力。然而，在这之前，需要在闪存架构和技术方面取得一些进展，以解决在单一闪存组件中存储代码和数据的问题。这些进展中没有一项可以单独实现代码+数据，但每一项都是实现今天的代码+数据的重要基石。对闪存技术和架构的重大发展的研究将阐明单芯片代码+数据所涉及的问题，并说明今天的闪存组件是如何以这种方式使用的。

随着当今快速变化的环境和快速的开发周期，诸如可更新代码等系统功能已成为强制性的。此外，由于其在各种特性方面的优势，闪存是作为集成存储器功能接收者的合理选择。闪存是非易失性的，可电动更新，读取速度快，密度高，而且成本相对较低。由于其他存储器类型不能提供这种功能组合，这些特性使得将闪存作为系统存储器组件之一几乎是强制性的。

如果可以假设系统中存在Flash，那么什么可以被消除呢？(见图2.17)直接从Flash执行代码，通过快速突发或页面读取模式，消除了在不同存储器中存储代码和执行代码的需要，减少了系统RAM的需求。EEPROM的参数存储功能和RAM数据存储可以通过使用软件数据管理器整合到Flash中。这就从系统中消除了EEPROM，留下了两个存储器组件，即Flash和RAM。RAM的大小可以大大减少，因为它不需要存储执行的代码，也不需要持久的数据。最终的结果是降低了元件数量，产生了更低的成本、功耗和电路板空间。

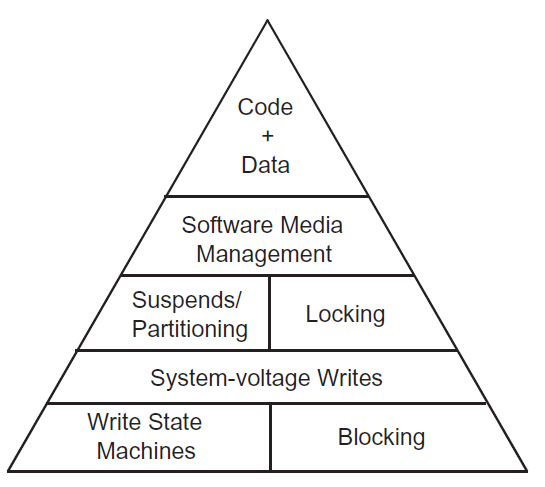


图2.16.导致代码+数据的Flash发展。

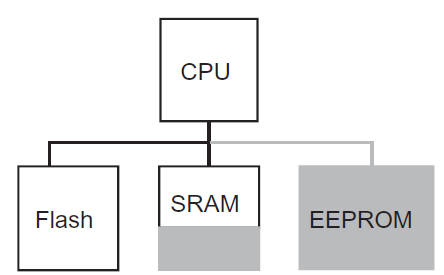


图2.17.存储器子系统的方框图，显示了可以集成到闪存中的存储器。

代码+数据系统架构的目标是整合各种类型的存储器组件。存储器子系统的集成是一个明显的设计趋势，通过将EEPROM功能与NOR型闪存设备相结合，实现了减少元件数量、成本、功耗和电路板空间的直接好处。这些器件具有多种特性，使它们成为结合代码执行和数据存储的明显选择，特别是在没有传统架构负担的嵌入式设计中。今天一些采用代码+数据架构的重要应用是无线电话和手持电脑。今天的闪存设备已经发展出支持集成代码+数据架构所需的功能。

不足为奇的是，那些最关注尺寸、功率和成本的应用正在积极地追求代码+数据设计技术。最明显的例子是移动电话和其他便携式无线设备。一部手机如果随处携带并不断打开，是最有用的，所以手机制造商已经抓住了代码+数据，作为一种在最小化尺寸和功率的同时最大化功能的方法。

# 2.4.1 代码+数据的相关属性

不出所料，代码+数据应用需要结合代码和数据应用的需求。具体来说，这种使用模式需要允许在代码和数据之间有效地分配闪存空间的属性，闪存写入操作的常规可用性，以及保护存储的代码和数据不被破坏的方法（见表2.4）。

***2.4.1.1 阵列分块。***将闪存阵列划分为区块，可以更新代码和数据，而不会因为擦除整个阵列而产生丢失一切的风险，从而使系统设计更加稳健。这种额外的安全性使设计者有信心定期向闪存写入。

***2.4.1.2 对称与非对称的设备。***在代码+数据应用中存储的数据的类型和大小决定了存储数据所需的闪存块大小。小尺寸的数据，如参数和计时器，在较小的块中更合适，而较大的数据类型，如语音记录或来自互联网的数据包需要较大的块。一个不对称封锁的设备既包含参数存储的小块，也包含代码存储的大块。一个只需要存储参数的代码+数据设计，可能会发现非对称封锁的设备是满足代码和数据需求的有效方式。然而，如果数据类型

表2.4.为代码+数据系统选择一个闪存设备

特点理想的特征牢记在心

写入状态机

* 紧凑型指令集总线周期
* 有用的命令集扩展，如缓冲写入
* 长命令序列在传输数据序列时增加了总线周期

分块 - 较小的区块大小擦除速度更快

* + 非对称阻断可以为你的应用优化块的大小

电压 - 系统级写入电压

* + 12-V的程序操作，用于更快的生产线
  + 独立的读和写电压引脚允许切换写电源，以节省电力和保护数据
* 较小的区块增加了模具成本
* 软件媒体管理减少了对小块的需求
* 了解你的应用对Flash电源模式（待机、激活、关机）的使用比例是评估Flash功耗的关键。

写作暂停

* + 灵活的暂停命令：擦除/编程暂停读取，擦除暂停编程，擦除暂停编程暂停读取
  + 确定性的悬挂延迟
  + 低悬挂延时
  + 太频繁地暂停擦除或编程会降低数据吞吐量

分区 - 代码和数据分区的大小

适用于您的应用

锁定 - 能够改变系统内的锁定

* + - 每个区块的独立锁定控制
    - 快速、实时的锁定变化
    - 对代码和数据的不同程度的保护
    - 用于额外保护的硬件控制

软件 - 有能力针对单一（软件

边读边写）或来自同一代码库的双分区设备

* + - 磨损平整
    - 电力损失恢复
    - 有能力处理你当前和未来的数据类型
    - 支持和持续发展
* 固定的分区大小可能不适合你的应用程序的数据或代码大小。
* 你的应用程序的性能可能不需要双分区
* 只能在系统外定义的锁定
* 长的锁定延迟可能意味着你无法使用锁定功能
* 限制你的设计潜力的特点
* 媒体管理器的结构对你的应用来说是不理想的（磁盘式仿真，等等）。
* 限制你的Flash来源的许可或技术限制

需要更多的存储空间，数据可能需要存储在更大的块中，使对称封锁的Flash设备成为更有力的选择。蜂窝电话提供了一个很好的例子：基本的纯语音电话只需要存储无线电调谐或呼叫计时器的小参数，很适合非对称Flash，而记录语音或互联网数据的高端电话则需要使用非对称和对称Flash设备中的大块。请注意，软件

数据管理者倾向于将大数据和小数据混合在同一个块中，降低了数据存储领域中块大小的重要性。

***2.4.1.3 电压。***需要非标准电压来编程或擦除闪存的设备使这种操作成为例外，只有在关键的时候才会行使，比如纠正一个致命错误的代码更新。但是，"代码+数据 "是为了使对闪存的写入成为系统操作的常规、综合的一部分。因此，能够使用标准的系统工作电压进行编程和擦除的闪存器件是代码+数据系统设计的最佳选择。

***2.4.1.4 写入暂停和双分区。***因为代码+数据系统，就其性质而言，在同一个闪存设备中存储数据和执行代码，需要一些机制来交错读写操作或允许它们同时进行。在标准的单分区闪存设备中，编程和擦除暂停功能可以实现交错的读写操作。如果在编程或擦除操作期间需要执行代码，例如，为一个中断提供服务，那么可以暂停写操作，以便从闪存中读取。双分区或多分区的闪存设备更进一步，允许在不同的分区同时进行读写操作。对于代码+数据系统，使用单分区还是多分区的决定取决于系统对数据存储和代码执行读取的吞吐量要求。随着传入数据的数据率和代码执行性能要求的提高，在单分区闪存设备上使用暂停会降低系统性能。双分区或多分区设备可以缓解这一瓶颈。

***2.4.1.5 写保护和块锁定。***代码+数据系统经常写到他们的闪存设备上，这就需要写保护或块锁定方案。此外，系统应该保护闪存中的数据段，因为它们不能容忍损坏。写入保护方案的目的是通过延长改变闪存内容所需的命令序列来防止由系统噪声引起的意外写入，但是在代码+数据系统中，较长的命令序列可能会妨碍系统的性能，因为更多的总线周期必须用于命令本身。

区块锁定是另一种非常适合于代码+数据的方法。允许锁定单个区块的闪存设备是代码+数据的理想选择，因为只有受影响的区块在必要时需要解锁，其余的代码和数据则被安全锁定。有些方案需要将一组块一起解锁，或者在生产时只能在系统外进行锁定/解锁。另一个要看的项目是区块锁定方案的性能。如果一个锁的变化需要太长的时间，那么它要么会影响到代码存储的性能，要么就不会被使用。无论哪种情况，缓慢的锁定变化都会降低性能和安全性。

***2.4.1.6 擦除循环。***随着系统在单一系统闪存设备中加入代码和数据，块擦除操作不可避免地增加。一个只有代码的设计在其代码更新的生命周期中最多只能积累一百个擦除周期，而一个代码+数据的设计必须适应这些代码更新周期，此外还有不断的数据存储和更新。数据存储可以将设备的循环次数推到数千次或更多。然而，请注意，使用

适当设计的数据管理软件大大减少了擦除周期数，因为在需要擦除之前，多个数据在一个块中被存储和更新多次。

***2.4.1.7 软件。***前面讨论的所有创新都为Flash中的代码+数据存储提供了硬件构件，但还有一个构件，即软件媒体管理，需要在系统中实现代码+数据架构。之所以需要一个媒体管理器，是因为闪存块的大小一般在8到128 kB之间，相对于系统中使用的大多数数据参数的大小来说，这是一个很大的数字。由于单个字节的信息不能被擦除，所以每次需要写入新的信息时擦除一个闪存块是不现实的。

媒体管理器的主要功能是通过管理信息的存储和旧的、无效的数据所使用的空间的回收，使块大小对系统透明。

英特尔的Flash Data Integrator（FDI）软件就是这样一个媒体管理器的例子。FDI实现了代码+数据设计所需的两个关键元素。首先是一个实时中断处理程序，如擦除暂停部分所描述的。这段代码将对中断进行轮询，并相应地暂停对闪存的写入，使代码+数据应用于单分区的闪存设备。第二个是一个强大的媒体管理器，为系统应用代码提供应用编程接口（API），以在闪存中存储参数、数据流和其他数据类型。媒体管理器提供了一个标准的、更高层次的访问数据的方式，并处理存储、检索和保护闪存中的数据所需的所有细节。此外，媒体管理器还提供了一些功能，如磨损均衡和断电恢复，以确保实时系统中代码+数据系统的稳定运行。

选择正确的闪存管理软件是很重要的，因为它的功能和灵活性将决定系统开发资源的分配。有了全功能的闪存管理器，开发可以集中在闪存管理器基础上的额外系统功能上，而不是在闪存阵列中设置和查找比特的细节。虽然一个功能有限的数据管理器看起来更容易实现，但这些限制可能会阻碍系统的潜力。

在未来，如果应用需求变得更加标准化，许多媒体管理的元素可能会被纳入到Flash的硬设备中。就像写状态机抽象出编程和擦除Flash所需的写脉冲一样，一个更高级别的命令接口可以纳入媒体管理功能，这样参数就可以直接写到Flash中。闪存硬件可以处理空间分配、垃圾回收等问题。当寻找Flash硬件的未来时，看看驱动Flash的软件层。该软件的功能很可能被纳入下一代闪存芯片中。

# 2.4.2 将代码+数据的各个部分结合起来

代码+数据 "值得对系统的各个部分如何相互关联进行更详细的描述，这不仅仅是因为它是最新的，而且很可能是不熟悉的使用模式，而是因为它的实现锻炼了系统的许多方面，从CPU和Flash硬件到运行在CPU上的软件栈。

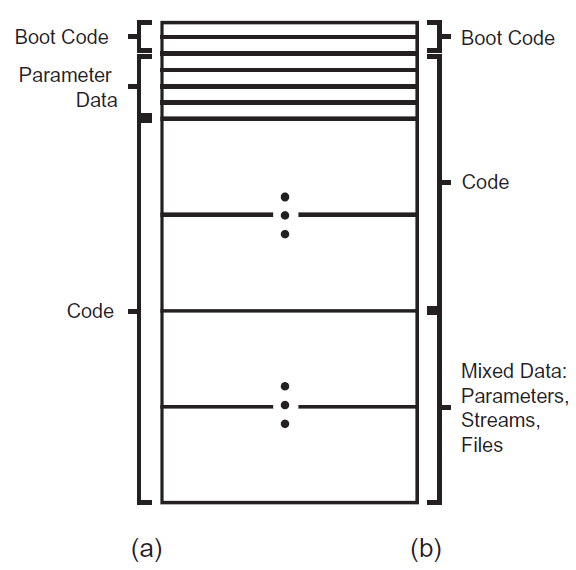


图2.18.显示启动代码、数据和代码之间闪存分配的内存图。

(*a*)小数据应用和(*b*)大数据应用。

***2.4.2.1 分配闪存空间。***一个代码+数据的设计需要在Flash中为三类基本信息分配空间：启动代码、代码和数据。启动代码需要放在存储器的顶部或底部，这取决于处理器的结构。因此，它的位置是预先确定的。在图 2.18 中，显示了一个不对称封锁的闪存设备是如何分配的，引导代码被显示在内存图的顶部，占据了前两个块。请记住，不对称封锁的闪存设备有顶部和底部的镜像配置，所以这张图可以垂直镜像来了解底部的启动配置。

其他代码和数据的放置更加灵活，能够根据应用需求进行调整。图2.18(*a*)显示了如何为一个只需要存储少量数据的代码+数据设计分配内存，类似于EEPROM。在这种情况下，非对称块结构的剩余小块可以用于数据，而所有较大的块可以分配给代码。

另一方面，图2.18(*b)*显示了如何为一个需要存储大量数据的代码+数据设计分配内存。这些数据不能放在小块中，所以需要移到大块中。但是，它应该被放在大块的顶部，与小块相邻，还是放在内存地图的底端？两种放置方式都可以，但图2.18(*b*)显示它们被放置在底部。这样做是为了使代码被放置在内存图的顶部，允许其大小向底部增长而没有问题。如果数据的大小需要扩大，那么就可以向上扩展。一个合适的媒体管理器将能够顺利处理这两种扩展。

***2.4.2.2 系统软件栈。***鉴于现代Flash组件，代码+数据的系统架构的关键补充是软件媒体管理。

|  |
| --- |
| 应用 |
| 初级的数据管理器 |
| 闪存驱动器代码 |
| 闪存硬件命令接口 |

|  |
| --- |
| 应用 |
| Flash媒体管理器 |
| 闪存驱动器代码 |
| 闪存硬件命令接口 |

* + - * 1. (b)

图2.19.传统和代码+数据系统软件栈的比较：（*a*）传统软件栈和（*b*）代码+数据软件栈。

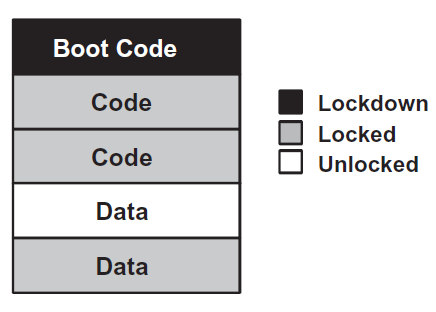


图2.20.内存分配显示适当的锁定方法。

一个典型的预编码+数据应用的软件栈可能如图2.19（*a）*所示。系统应用与低级别的驱动层对话，驱动层将实际的命令发送到Flash硬件的命令接口。这意味着系统应用必须结合媒体管理，以便格式化和检索存储的数据。由于这种闪存媒体管理通常不是系统级硬件或软件设计者的核心能力，一个简单的媒体管理器可能会限制闪存的数据存储。

实现代码+数据系统需要在低级驱动和系统应用之间增加一层软件，如图2.19（*b）*所示。这层软件，即媒体管理器，抽象了Flash特定的格式化和位操作，这样应用软件就可以简单地将数据作为参数或数据流来处理，并传递给媒体管理器进行存储。

图2.20显示了代码和数据在Flash中的映射，以及适当的锁定模式。从顶部开始，第一个区块包含了启动代码，并且已经被 "锁定 "了，所以要改变该区块的代码，需要对软件和硬件进行修改。接下来的两个区块包含数据，并已被锁定以防止更改。然而，在代码更新的情况下，可以使用一个软件命令来快速解锁这些区块。接下来的两个区块被指定为数据区块，其中一个被解锁以进行更新，而另一个则被锁定。

***2.4.2.3 系统运行。***到目前为止，我们已经讨论了 "代码+数据 "系统架构的所有元素，但我们需要把它们整合在一起，看看它是如何一步步运作的。相关的系统硬件一般由CPU、Flash和RAM组成。闪存是主要的内存设备，包含启动代码、数据和应用代码。

1. CPU从闪存中启动，并开始执行直接从闪存设备中获取的代码。启动代码使系统通过初始化，然后继续执行闪存中的应用代码。在这一点上，Flash媒体管理器可以被认为是被编译到了应用程序代码中。
2. 在某些时候，当应用程序需要将数据存储到Flash时，它将对媒体管理器API进行适当的调用。媒体管理器将格式化数据并开始将其写入Flash。
3. 暂时假设闪存设备是单分区的，一小部分控制写入过程的代码将被复制到RAM。
4. CPU将从RAM开始执行，并开始向Flash写入数据。
5. 如果在写的过程中收到中断，Flash将无法进行代码获取，所以RAM中的代码应该暂停Flash的写过程。
6. 然后，CPU可以回到从闪存中获取代码，为中断服务。
7. 当这一切完成后，然后执行返回到RAM。
8. 写入过程由RAM中的代码继续进行，直到完成。
9. 然后执行返回到Flash。

如果闪存设备是一个双分区设备，由于代码不需要复制到RAM中来控制写入过程，步骤3到9简化如下。

1. 如果闪存设备是双分区或多分区，控制写入过程的代码将从一个与包含数据的分区不同的分区开始执行。
2. 如果在写入过程中收到中断，CPU可以跳到Flash中的另一个代码段，只要该段与包含数据的分区不同。
3. 当中断的代码完成后，执行返回到媒体管理器，直到写入完成。代码的执行永远不会离开闪存，除非代码需要从正在写入的分区中执行。在这种情况下，那么写的过程将不得不暂停，处理方式类似于单分区设备。

# 2.4.3 代码+数据的好处

存储器子系统的集成是一个明显的设计趋势，通过将EEPROM功能结合到NOR型闪存设备中，可以直接减少元件数量、成本、功耗和电路板空间。这些器件具有多种特性，使它们成为结合代码执行和数据存储的明显选择，特别是在没有传统架构负担的嵌入式设计中。今天一些采用代码+数据架构的重要应用是无线电话和手持电脑。今天的闪存设备已经发展出支持集成代码+数据架构所需的功能。

从更广泛的角度来看，代码+数据不仅对减少组件和成本很重要，而且是为未来丰富的数据应用提供基础的架构步骤。依靠有限大小的EEPROM的传统系统架构将继续被锁定在小数据思维中，其创新和功能受限于其参数存储的大小。将代码+数据架构作为一种使能技术的系统将准备好迎接网络世界中不断增长的数据量。闪存设备的密度继续增加，围绕它们设计的系统可以与它们一起成长，无限制地适应不断扩大的代码和数据大小。

# 2.5 结论

闪存将实现哪些未来的应用？不久前，闪存制造商还在预言，硬盘驱动器行业即将崩溃，因为除了最高密度的产品外，所有的产品都将在2000年之前转换为闪存。世界并没有在千禧年结束时停止工作，而硬盘驱动器也没有灭绝。很明显，随着旋转媒体的密度升级，如果一个应用程序可以使用硬盘，它就会使用。每兆字节的价格实在是太惊人了。闪存将继续扩大其在功耗、可靠性、便携性和空间限制要求固态非易失性存储器的应用。

电话将把答录机整合到听筒中。手机、传呼机和数字助理将合并为一个单一的连接设备。化学胶片与电子胶片（Flash）共享货架。摄像机将变成数字摄像机，磁带将最终被固态存储卡取代。光盘最终将被高密度多级NOR或NAND闪存取代，就像唱片被CD取代一样。

在《星际迷航》中，人们将信用卡大小的设备插入系统中，收听信息和音乐，或者拥有支持语音识别的计算机，这种未来主义的设想比该剧的创作者预测的要早几十年成为现实。不管我们对Flash的未来应用作何假设，很明显，随着应用对内存技术的要求越来越高，Flash的发展方向是更大、更快、更便宜。