SSD 数据结构与算法综述

史高峰，李小勇

摘　要：SSD(Solid State Disk)是一种基于闪存的电可擦除可编程的新型存储器件。与普通硬盘相比，SSD 具有访问延迟小、低功耗等优点。但 SSD 的应用也需要解决写前擦除、损耗平衡等挑战。针对这些挑战，工业界和学术界研究提出了多种复杂的算法和数据结构。对这些研究成果进行了分析和综述，同时探讨了 SSD 在分层存储中的应用。

关键词：SSD；分配或回收策略；垃圾回收；二级存储；分层存储

中图分类号：TP393　　文献标志码：A

Algorithms and Data Structures for SSD

Shi Gaofeng, Li Xiaoyong

(School of Information Security Engineering, Shanghai Jiao Tong University, Shanghai200240, China)

Abstract: SSD(Solid State Disk) is based on flash memorywhich is a type of electrically erasable programmable read-only memory (eeprom). Compared to electromechanical HDDs, SSDs are typically less susceptible to physical shock, are silent, have lower access time and latency.But SSD suffers from two limitations. First, bits can only be cleared by erasing alarge block of memory. Second, each block can only sustain a limited number of erasures, afterwhich it can no longer reliably store data. Due to these limitations, sophisticated data structuresand algorithms are required to effectively use SSDs. These algorithms and data structuressupport efficient not-in-place updates of data, reduce the number of erasures, and level thewear of the blocks in the device. This article also introduces the technology of using SSDs in multi-tier systems.

Key words: SSD; Allocation/deallocation Strategies; Garbage Collection; Secondary Storage; Multi-tier Systems

0 引言

SSD 是一种基于闪存的电可擦除可编程的新型存储器件，本文主要介绍 SSD 闪存的数据结构和算法以及 SSD 闪存在分层存储中的应用。

闪存的读、写或擦除操作与易失的 RAM 和磁盘有很大的不同。闪存的每个单元只能擦写有限次（10，000 到 1，000，000 次），超过这个最大次数后，闪存就变得不可靠了。

闪存种类主要有两种，NOR 和 NAND，这两种有很大的区别。写操作都是把“1”置为“0”，而擦除操作需要把一整个擦除单元的所有位都置为“1”。一个固定设备的擦除大小是固定的，从几 KB 到上百 KB。NOR 闪存的每个位（bit）都能被处理机直接访问并擦除，但是 NOR 的擦除时间比较长，而 NAND 闪存，擦除时间比较短，但是处理机不能访问到每个位（bit），访问的最小单位是页（page），一个擦除单元由很多页构成，典型的页大小为 512 字节（byte）。一个擦除周期中的一个页只能被修改几次，因为修改几次后，页中的数据就不再可靠了。

因为这些特殊性，基于磁盘的一些存储技术就不太适合闪存了。随着闪存的广泛应用，在 20 世纪 90 年代后，基于闪存的存储技术不断发展。这些技术中，有一小部分是针对闪存的发明设计，而大部分发明是源自其他的存储技术。本文就是分析比较了闪存的各种数据结构和算法。

1 块映射技术

将闪存块当做磁盘块使用，会产生两个问题：

第一，一些块的修改次数远远大于其他块，对于磁盘块不会有任何问题，但是闪存中的这种块就会由于擦除太多而达到擦除上限，这种问题可以通过损耗均衡技术解决。

第二，不合理的映射会将一个很小的块映射到闪存的一个擦除单元，例如，文件系统将一个 4KB 的数据块映射到闪存中一个 128KB 的擦除单元中，那么修改这个数据块就需要将这 128KB 复制到内存，在内存中修改这 4KB 数据，擦除闪存中相应的 128KB 的擦出单元，然后，将这 4KB 数据写回闪存，如果中间掉电，可能会损失 128KB 数据，而磁盘中只会损失 4KB 数据。这个问题也可以通过损耗平衡技术解决。

1.1 块映射的思想

所有损耗平衡思想都是基于一个映射关系，即一个虚拟块号映射到闪存中的一个物理块号。闪存中，当一个虚拟块需要被重写，新数据不会写到之前映射到的那个物理块，而是写入一个新的物理块中，并修改逻辑块号和物理块号的

映射表。

闪存中的一个物理块一般是一个擦除单元中的固定大小的一部分，在 NAND 闪存中，一般是一个页，而在 NOR 闪存中，有可能还会是变长大小。

这种映射有几个目标：

首先，频繁修改的块会保存到不同的物理块中，便于损耗平衡。

再次，这种映射允许写擦除单元中的一个页，而不是立即擦除掉整个擦除单元再进行写。

第三，写操作保持原子性，如果写的过程中断电，可以回滚到之前的状态。

写操作的原子性主要通过以下技术实现，每个物理块有一个小的头部，这个头部可以在物理块中，也可以保存在其他擦除单元中。当写一个块时，首先找到一个可用的物理块，这种块和它的头部中的所有位都为“1”，然后头部中的“空闲/占用”位被置为“0”，表明这个块已经被占用，逻辑块号写入这个物理块的头部的相应位置，并且数据写入相应的物理块中，然后，头部中的“无效/有效”位被置“0”，表明这个块可以读，最后，之前旧的版本的物理块的头部中的“有效/过时”位被置“0”，表明这个块不再是虚拟块的最新版本。

如果在写操作中电源丢失，可能会出现两种状况，如果断电发生在新写的物理块被标记为有效之前，则新写的块被置为无效，并且“有效/过时”位被置“0”，标记新块可以被回收；如果断电发生在新写的物理块被标记为有效之后但是旧的物理块标记为过时之前，这两个物理块都是合法的，系统将随意选择一个标记为过时，如果最近更新的版本更重要，通过一个两位的版本号标示出哪个是最新的，具体是 1 比 0 新，2 比 1 新，3 比 2 新，0 比 3 新【Aleph One 2002】。

1.2 块映射的数据结构

系统如何通过逻辑块找到相应的物理块呢？一般情况下，通过两种数据结构，一种是正向映射，另一种是反向映射。正向映射是将逻辑块号和物理块号填入一张表中，系统可以通过逻辑块号查询到相应的物理块号，反向映射是将每个物理块的块头中保存逻辑块号，如果系统需要通过逻辑块找到相应的物理块，就需要遍历物理块直到找到相应的逻辑块为止。正向映射可能不是简单的队列实现而是一些更复杂的数据结构，以便于更高的查询效率。而反向映射必须是队列，队列中的物理位置上可以不是连续的。

反向映射保存在闪存中，当一个逻辑块写入物理块时，逻辑块号记录到相应的物理块的头部中，并且逻辑块号伴随物理块的擦除而擦除。反向映射的主要作用是在设备初始化时重构正向映射。

正向映射保存在闪存控制器的内存中，是易失的，正向映射的作用是支持快速查询逻辑块的相应物理块，当逻辑块和相应的物理块的映射关系更新时，正向映射也应该相应的更新，但是闪存不支持这种就地更新正向映射，其过程，如图 1 所示：

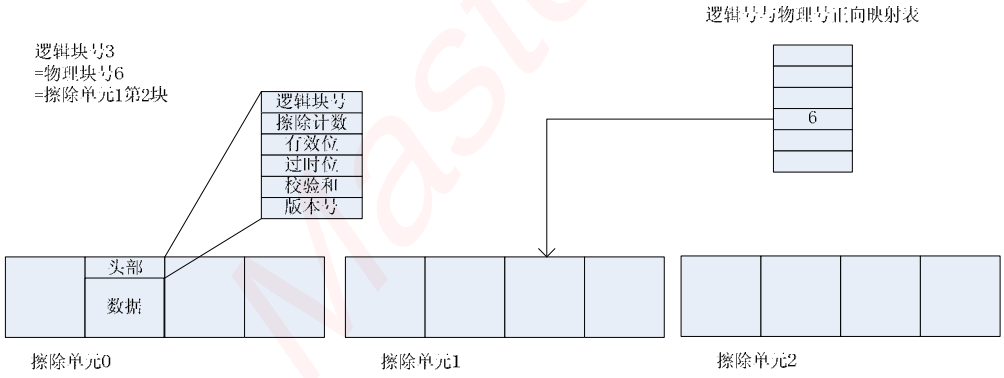

图1，闪存中的块映射。右上方是逻辑号与物理号的正向映射表，保存在闪存驱动器的内存中。每个物理块包含一个头部和相应的数据。头部包含逻辑块号、擦除计数、有效位、过时位、校验和和版本号等。所以块头部中的逻辑块号就构成了反向映射表，通过它们可以构成正向映射表。版本号可以区分同样的逻辑块号中的哪个数据块最新。

正向映射并不是必须的，系统可以通过线性查询反向映射找到逻辑块号对应的物理块，这样的查询速度慢，但是节省闪存控制器中的内存空间。如果把一个逻辑块仅仅映射到一个集合的物理块中，这样可以提高查询速度【Assar et al. 1995a; 1995b】。这种技术类似于缓存中的组相联映射，减少了内存占用，提高了查询速度，但是减少了映射的灵活度，可能会导致损耗平衡问题。

闪存转换层是一种将正向映射保存在闪存内存中，减少闪存中的映射更新。这个技术是 Ban 发明的【Ban 1995】，并且在后来被 PCMCIA 标准采用。

Ban 后来发明了基于 NAND 闪存的闪存转换层 NFTL【Ban 1999】，主要思想是，将一个逻辑块号映射成一个逻辑擦除单元和擦除单元中的块号，每个逻辑擦除单元对应一个物理擦除单元链，比如，查找逻辑擦除单元 7 的第 5 块，系统就顺着这个链，找到相应的物理块，将其返回，这个有效块的之前的擦除单元中的第五块都无效，之后的擦除单元中的块仍旧可用（所有位全是“1”）。如果修改这个第 5 块，则找出链中的下一个擦除单元的第 5 块，将数据更新到新块中，如果链中没有可用的第 5 块，则寻找新的物理擦除单元加入链中。当回收空间时，系统会将所有有效数据块写入链的最后一个擦除单元中，最后一个擦除单元作为新的链头，所有旧链中的其它擦除单元都被擦除，链的长度一般是 1 或更长。

逻辑擦除单元与物理擦除单元链的映射关系，保存在一个物理擦除单元头部，在闪存设备初始化时，在闪存控制器的内存中构造相应的正向映射，因为是擦除单元之间的映

射，而不是块之间的映射，所以这个映射占用的空间很少。

也可以将变长的逻辑块映射到闪存中，Wells at al.发明了这项技术【Wells at al.】，并且微软闪存文件系统【Torelli 1995】也采用了相似的技术，主要思想是，在一个擦除单元内，将变长的数据块从低地址存放，定长的头部信息从高地址存放。每个头指向相应的变长的块，微软闪存文件系统就是这样的系统。

Smith 和 Garvin 发明了一个类似的系统，但是粒度更粗，他们将每个擦除单元分为 3 部分：头部、分配映射表和多个固定大小的块。系统将一个逻辑块分配到多个连续的物理块中，这些连续分配的物理块称作 extent，并在分配映射表中有一个表项，这个表项中的信息包括 extent 的地址信息、长度和逻辑块号等。当 extent 中的块需要修改时，一个 extent 就被分裂成两个或 3 个 extent，将原来的映射表中的 extent 表项标记无效，并添加新的 extent 表项。

2 擦除单元回收

随着时间的推移，闪存设备上的过时块越来越多而空闲块越来越少，为了空间利用，过时块必须被回收，回收块的唯一方式就是擦除整个擦除单元。回收总是针对整个擦除单元来说的。

回收发生在系统空闲时或当空闲块数量低于一个预先设定的阈值时。

系统回收空间有几个阶段：

首先，一个或多个擦除单元被选为将要回收的单元。

然后，将这些擦除单元中的有效数据块复制到设备中别的空闲块中。

再次，修改迁移数据的逻辑块和物理块的映射关系。

最后，回收的擦除单元被擦除，并将这些回收的块加入到空闲块池中。

回收时需要将回收单元中的有效数据块复制出来，所以系统中一般都会预留一部分空闲擦除单元来满足擦除时的这种要求。

回收机制有两个主要策略：选择哪些单元回收？将这些单元中的有效数据块迁移到哪里？这两个策略又依赖于一个策略，就是在块更新时，如何分配物理块给新的数据块？这 3 个相关的策略以 3 种方式影响系统。它们影响回收进程的效率（以过时块占回收单元的比例为衡量），它们影响损耗平衡，它们影响映射关系的数据结构的相应更新。

损耗平衡的目标和回收效率往往是相悖的，例如一个擦除单元中仅仅包含静态数据（很长时间不被更新的数据），从回收效率的角度来说，不会选择回收这样的单元，因为这样的单元不能释放更多有效的多余空间，但是，这样的回收有利于整个设备的损耗平衡。假设我们知道一些数据属于静态数据，即一段时间内不会修改的数据，则我们将这些数据迁移到已经擦除次数较多的块上，这样我们就可以减少这些单元的劳损。

很显然，回收时机往往发生在设备中没有足够空闲空间的时候，这种策略称作及时回收，但是，有些系统的回收时机发生在系统空闲时，这种策略称作后台回收。后台自动回收需要整个系统的支持，有的系统可以识别空闲时期，有的系统则不能。闪存的特性也很重要，当擦除速度很慢时，就应该避免及时回收否则影响数据块更新的时间，当擦除速度很快时，这种影响就不是那么明显。同样，当擦除允许被中断并能从中断中恢复，那么后台回收就不太影响数据块的更新，而当擦除不允许中断时，后台回收就会很影响数据块的更新。但是，所有这些问题都与算法和数据结构有互相交错的影响。

2.1 损耗平衡为中心的回收和测量损耗的技术

这里描述的回收技术的主要设计是以减少损耗为中心的，这些技术在系统中用独立的有效回收机制和损耗平衡策略。系统中在大部分时间使用效率为中心的回收策略，但在一些时候转换为损耗平衡为中心的回收策略。有些时候不均衡的损耗引发转换，有些时候周期性的发生转换。很多技术用于测量损耗情况，接下来，介绍损耗测量技术。

Lofgren et al. 发明了一种由回收擦除单元触发的损耗平衡技术【Lofgren et al. 2000; 2003】。在这个技术中，每个擦除单元头部保存一个擦除计数。首先，系统中预留一个备用的擦除单元，当最劳损的擦除单元被回收时，计算最劳损的擦除单元与最不劳损的擦除单元的擦出计数的差，如果这个差超过一个阈值，比如 15,000，则负载平衡回收策略被采用，将最不劳损的单元中的有效块数据迁移到备用擦除单元中，擦除这个最不劳损单元，然后将最劳损单元中的有效数据块迁移的刚刚擦除的那个最不劳损单元中，再擦除那个最劳损单元，并将其置为备用单元。这个技术试图去识别最劳损单元和静态数据块，将静态数据块迁移到最劳损单元。而在下一个周期的损耗平衡中，最劳损块将保存最不劳损块中的数据。据推测，最不劳损单元中的数据相对静态，将这些数据块保存到最劳损单元中来平衡劳损单元的损耗，同时，将静态数据从最不劳损单元中迁移，提高了整个设备中的每个单元的损耗更加均匀。

很显然，这种技术依赖于擦除单元头部的擦除计数，如果一个擦除操作正在进行，新的擦除计数没有写入擦除单元头部时掉电，这个单元的擦除计数就会丢失，这种危险相当严重，尤其是当擦除速度比较慢时。

一种解决这个问题的方法是将擦除单元的计数保存到别的擦除单元中，比如单元 i 的计数保存到单元 j 中（$i \neq j$）。Marshall 和 Manning 就发明了这样一种技术【Marshall and Manning 1998】。系统中每个擦除单元保存头部保存一个擦除计数，在回收单元 i 之前，将 i 的计数复制到一个任意块 j 的一个特殊区域中（$i \neq j$），那么在回收中丢失单元 i 的计数，就直接可以从单元 j 中恢复。

Assar et al. 发明了一种更简单但是稍微低效的方法【Assar et al. 1996】，将一个擦除单元的边界为 8 的 8 位二进制数，保存在别的擦除单元中，比如单元 i 的擦除计数保存在单元 j 中（$i \neq j$），当单元 i 被擦除时，就将单元 j 中的一位二进制位擦除（“1”置为“0”），这样，单元 i 可以被擦除多次，擦除计数会相应地更新。但是，因为计数的边界是 8 位，当擦除次数多于 8 次后，这个擦除计数就不准确了。这种系统中，需要周期性的损耗平衡机制来会回滚所有的擦除计数。

Jou 和 Jeppesen 也发明了一种的擦除计数技术【Jou and

Jeppesen 1996】。他们系统中使用了称为“写前擦除”策略：将回收的擦除单元的有效内容复制到别的单元中，但是这个单元不会被立即擦除，而是将其标记为候选擦除单元，并加入到闪存内存中的候选擦除单元的优先级队列中，以损耗计数来排序，损耗计数最小的单元先被擦除为可用单元。这种技术在一定程度上平衡了损耗，因为延迟了一些损耗严重的单元被擦除，损耗的均衡与否取决于优先级队列中候选擦除单元的多少：如果候选擦除队列比较短，损耗严重的块就可能不会被延迟很久被擦除。

Han 发明了这个问题的另外一种解决方法【Han 2000】。依赖擦除延迟来预估损耗情况，一些闪存设备中，随着损耗的增加擦除延迟也相应增加，Han 的技术通过擦除所用时间来判断损耗情况，并将这些损耗情况排序，这避免了保存擦除计数。损耗情况排序可用于分配策略等，但是系统仅仅预估一段时间的擦除单元的损耗状况，对于那些一段时间内仅有少数单元被擦除的情况，就没有办法很好地使整个设备的损耗平衡。

另一种损耗平衡的方法依赖于随机性而不是预估块的损耗情况。Woodhouse 提出了一个简单的随机的损耗平衡技术【Woodhouse 2001】。每 100 次回收，系统会随机选择一个仅有有效数据的单元回收，这样就会将静态数据从所在的损耗小的单元迁移到损耗大的单元中，如果这个技术应用在一个倾向于回收效率而不是损耗平衡的系统中的话，极端的损耗不平衡仍旧存在，如果回收单元时仅仅根据其中的无效块的数量，那么一些仅有少量无效块的单元可能永远得不到回收。

大约在同一时间，Ban 发明了一个更加健壮的技术【Ban 2004】。他的这个技术依赖于一个备用单元。每一个确定的回收次数中，一个擦除单元被随机选中，将选中的擦除单元中的有效数据块复制到备用单元中，选中的擦除单元擦除后就标记为新的备用单元，这种损耗平衡事件的触发可以是确定的，比如每 1000 次擦除引发一次，或者随机的。随机的触发损耗平衡可能会仅仅几次擦除就产生一次。这个技术的目的是使闪存中的生命周期中的每个擦除单元有公平的互换，大量的互换可以减少一个擦除单元长时间地存储静态数据。另外，这个技术中的损耗平衡开销是可预见的并均匀的。

这个技术的思想在更早的软件中已经使用，M-Sytems 公司的 TrueFFS，使用了结合擦除次数和随机性的损耗平衡技术，M-Sytems 公司的描述中说，随机性的使用消除了保护每个擦除单元的确切擦除计数的必要。

2.2 结合有效回收和损耗平衡技术

Kawaguchi，Nishioka 和 Motoda 发明了一个这样的技术【Kawaguchi et al. 1995】。他们实现了一个闪存设备的块设备驱动程序，这个驱动程序被用于日志结构文件系统。这个文件系统操作更像一个块映射机制：它负责分配块并回收擦除单元。Kawaguchi et al. 描述了两种回收策略。第一个策略是选择下一个回收单元的原则是计算回收好处与回收代价的比率，一个单元的回收好处是这个单元中无效块的个数，回收代价是将这个单元中有效块复制到别的单元中的代价。一个单元距上次修改的时间也是其中的参数，即时间越长单元中的有效数据就越像静态数据，这个时间越长就越容易被回收。而这个技术避免了回收好处与回收代价比率增长时回收，因为这时擦除单元距上次修改的时间比较短。这个技术没有专门为损耗平衡设计，但是也达到了损耗平衡的结果，因为即使少量无效块的擦除单元，随着时间的推移，终将被回收。

Kawaguchi et al. 发现这个策略在某些情况下仍旧低效率，于是提出了另外一个策略，他们提出将数据分成两种单元，一种单元中存放“冷”数据，即在回收擦除单元中的不经常修改的数据，另一种单元存放“热”数据，通常不在回收过程中动态写入的数据。这种方式可以提高回收效率，因为动态的数据存放在一起，回收的时候可以将这个单元回收，而只迁移少量有效数据。存放静态数据的单元不需要回收，因为它上面没有无效数据块。显然，这样的静态数据单元很长时间不会得到回收，这会导致损耗平衡问题，除非采用别的损耗平衡策略。

Wu 和 Zwaenepoel 描述了一个更缜密的策略【Wu and Zwaenepoel】。系统将擦除单元分为固定大小的区域，假设小号的区域存储“热”逻辑块，大号的区域存储“冷”逻辑块，每个区域都有一个活跃的擦除单元存放最近更新的数据块，当一个虚拟块被更新，新的数据就写入当前区域的活跃单元的一个相应的物理块中，当一个区域中的那个活跃单元被写满，系统选择这个区域中有效块最少的单元回收，假如擦除单元分为前部和后部，在回收时，这个单元中的有效数据被复制到一个空单元的前部，这样，不经常更新的块就被保存到擦除单元的前部，即“冷”块，而新更新的块就保存到擦除单元的后部，也就是相对“热”块，Wu 和 Zwaenepoel 就是这样区分冷热块的，其冷热区如图 2 所示：

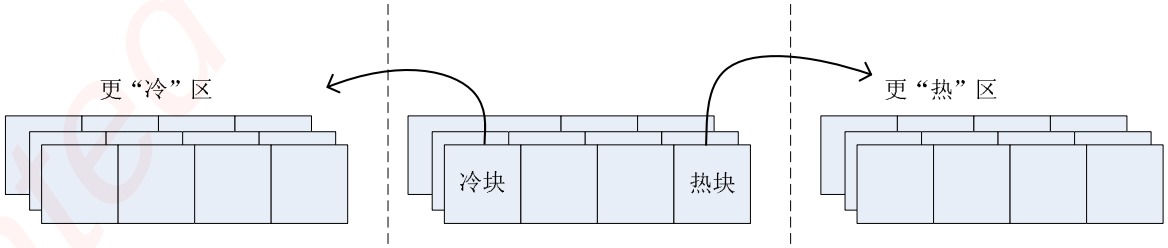

图2，闪存中的冷热区示图。其中，一个单元中的“　”块存放在一端，而“热”块存放在另一端。当这个区中的修改频率高于设备的平均频率时，回收一个单元时，　“　”块迁移到“冷”区，将“热”块迁移到“热”区。保持设备中每个区的访问频率均衡，达到损耗平衡。

对于每个区域，系统试图去获得均衡的回收频率，然而，热区域应该比冷区域有更少的块，因为热数据无效的速度更快。在每次回收时，系统会将当前区域的回收频率与区域的平均频率作比较。如果，当前区域的回收频率比平均频率高，就将区域中的一些块迁移到临近的区域中，擦除单元中的前部，即冷块迁移到冷区域中，擦除单元的后部，即热块被迁移到热区域中。

Wu 和 Zwaenepoel 实现了简单形式的损耗平衡，当最劳损的单元比最不劳损的单元擦除计数高 100 时，交换这两个单元的数据。当最不劳损的擦除单元保存的是静态数据时，这是很有效的，这种交换会使劳损的单元休息下来。

Wells 发明了一种依赖于有效回收和损耗平衡的回收策略【Well 1994】。系统选择下一个回收单元是基于一个分数，单元 j 的分数的定义如下：

$$分数(j) = 0.8 \times 无效块(j) + 0.2 \times [\max_i\{擦除数(i)\} - 擦除数(j)]$$

其中，无效块(j)是指擦除单元 j 中的无效数据块的总数，擦除数(j)是指擦除单元 j 的擦除次数，最大分数的块将是下一个回收的块，无效块(j)和擦除数(j)可以从每个擦除单元中获得，0.8 和 0.2 是发明中的数据，可以根据具体情况调整。总的原则是，回收有效性权重，而损耗平衡权轻。在足够多的单元回收后，系统会检查是否需要更激进的损耗平衡策略。如果最劳损和最不劳损的擦除次数相差超过 500 或更多，系统将用另外一个分数公式来计算回收策略。

$$分数(j) = 0.2 \times 无效块(j) + 0.8 \times [\max_i\{擦除数(i)\} - 擦除数(j)]$$

这种策略，先是考虑到回收效率，如果导致损耗不均衡，则进入损耗平衡阶段，在这个阶段回收效率不是那么重要。再者，没有必要过早进入损耗平衡阶段，因为那样反而会增加系统整体的损耗。

Chiang，Lee 和 Chang 提出了一个称作 CAT 的块集群技术【Chiang et al. 1999】，Chiang 和 Chang 后来改进这个技术为 DAC【Chiang and Chang 1999】。这些思想的中心概念是“温度”，一个块的温度表明这个块是否将要被更新。系统基于两个简单规则来为每个块维护温度：一个是当一个块被更新时，温度升高，另一个是，随着时间的增长块的温度降低。CAT 将块分为三类：只读、冷和热。这种分类没必要必须与温度一致，因为在 CAT 中，块仅仅在回收的时候重新分类，每个擦除单元存储一类块，当擦除单元要回收时，擦除单元上的有效块被重新分类，CAT 选择一下公式计算分数最高的回收单元。

$$CAT\,分数(j) = \frac{无效块(j) \times 年龄(j)}{有效块(j) \times 擦除次数(j)}$$

其中，有效块(j)是擦除单元 j 中的有效数据块的个数，年龄(j)是从上次更新为止的时间，从这个分数公式可以看出，系统会优先选择无效块多而有效数据块少的单元、不是最近更新的单元和擦除次数少的单元。这一定程度上结合了有效性和损耗平衡。CAT 策略还包括额外的损耗平衡机制：当一个擦除单元将要达到它的擦除上限时，会将其与擦除次数最少的擦除单元交换数据。

DAC 策略则更加复杂。首先，块被分为 3 个以上的分类，更重要的是，块在每次更新时重新分类，所以冷块升温后将被重新分配到一个更热的擦除单元中，即使这个冷块所在的单元没有回收时。

TrueFFS 选择回收单元是基于无效数据个数、擦除次数和静态数据的识别【Dan and Williams 1997】。TrueFFS 还试图将相关的块放在一起，这样一个擦除单元中的块就可能同时无效，做法是将连续编号的逻辑块存放在一个擦除单元中，这种做法基于的假设是，上层软件将相关的块是有连续的逻辑编号。

Kim 和 Lee 提出一个自适应的结合损耗平衡和回收有效性的技术【Kim and Lee 2002】。他们的方法是设计一个文件系统，而不是一个块映射机制，但是因为它也适应于块映射机制。这个方法称作 CICL，选择一个擦除单元（事实上是一个称为段的一组擦除单元）回收，回收依据为一下公式的最小分数。

$$CICL\,分数(j) = (1-\lambda)\left(\frac{有效块(j)}{有效块(j)+无效块(j)}\right) + \lambda\left(\frac{擦除次数(j)}{1+\max_i\{擦除数(i)\}}\right)$$

在这个表达式中，λ 不是一个常量，而是依赖于更加最劳损和最不劳损擦除单元的差值。

$$0 < \lambda\left(\max_i\{擦除数(i)\} - \min_i\{擦除数(i)\}\right) < 1$$

当 λ 小的时候，单元回收主要考虑擦除效率，而 λ 大的时候则以损耗平衡为主。当 λ 增加时，损耗平衡问题越来越严重，这时候这个公式就会将重点转向损耗平衡，而损耗均衡时则以擦除效率为主。

3 SSD 在分层存储中的应用

SSD 闪存具有较高的 IO 性能并且耗电量小等优点，但是 SSD 盘的单位价格仍旧太高，纯的 SSD 盘性价比比较低，而利用 SSD 的高性能和 SATA 盘的低价格组合的分层存储已经成为业内的更好的选择，很多大型数据中心已经开始研究如何使用 SSD 与 SATA 结合来提供更高的性价比，比如 IBM 数据中心等，SSD 做为分层存储有两种方案，一种是将 SSD 与磁盘并行，另一种是将 SSD 作为磁盘的缓存，如图 3 所示：

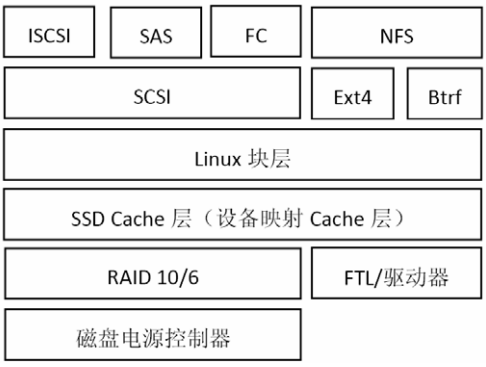

图 3　SSD 分层存储结构图

3.1FlashCache

FlashCache 是一个为 innoDB 设计的块 cache 应用，也可以用于一般的应用。FlashCache 是基于 linux 设备映射（Device Mapper）框架，将 SSD 和 SATA 盘虚拟成一个独立的块设备的软件，利用 SSD 的高性能和 SATA 盘的高容量来提升整体的性价比。

FlashCache 是将 cache 部分（SSD）分成逻辑的集合，然后将数据块通过 hash 运算到相应的集合中，其中，块大小、集合大小以及 cache 大小是在创建时配置的参数，默认的集合大小为 512 块，这样，集合个数就时 cache 大小除以集合大小。其中，dbn（disk block number）即磁盘块号，也就是磁盘的逻辑块号，FlashCache 使用以下的 hash 公式，将相应的磁盘块映射到相应的集合中。

$$\text{目标集合} = \text{hash}(\text{dbn}) = \left(\frac{\text{dbn}}{\text{块号}}\right)\%\text{集合个数}$$

所以，每个块一定会映射到相应的集合上，根据块号就能从相应的目标集合中就能线性探相应的块。设备映射层会将 IO 组合成固定块大小，然后将块存放在 cache 上，FlashCache 缓存所有的块级别的 IO，在每个集合中替换算法选择是 FIFO 或者 LRU，可以通过 sysctl 进行算法切换。

对于每个逻辑块来说，都有一个逻辑块头，逻辑块头中包括逻辑块号（dbn）、闪存状态（Dirty/Valid/Invalid）、用于 LRU 算法的静态指针和校验和信息。校验和信息主要保证块信息的正确性，也可以不设置校验和。

所有的逻辑头，即逻辑块的元数据，可以拼接成物理块，存放在 SSD 中。

逻辑头所占用的空间开销，例如 300GB 的 SSD cache 用 16KB 固定块大小，大约 2 千万个 cache 块，元数据占用 480MB，如果是 300GB SSD 用 4KB 的块大小，元数据就占用 1.8GB 空间，如图 4 所示：

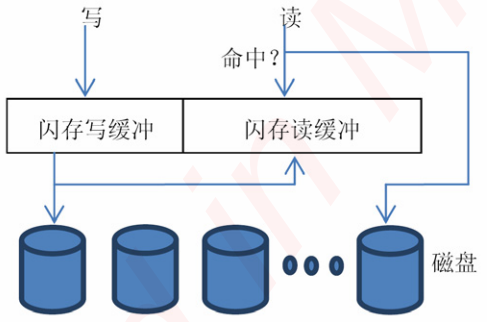

图 4　SSD 写回架构

读过程，首先，通过 hash 计算找出 dbn 所在的 SSD 中的目标集合，然后从相应的目标集合中线性找到 dbn，如果命中则返回数据，否则从磁盘中读数据返回，然后将数据块替换到相应的集合中。

写过程，FlashCache 的写过程使用写回策略，即所有的写操作都将数据写入 flash 中，并将写入的数据标记成脏块。（如果写入的数据已经是脏块，则不需要重新标记。）每个集合中的脏块百分比有个阈值，当脏块的比例超过这个阈值时，就被设置成将要写回磁盘，对于脏块的选择，使用一个类似时钟算法选择空闲的脏块写回。

读写过程中，尽量少写回元数据，避免不必要的损耗，比如读数据时，就不需要写回。

替换策略选择 FIFO 先进先出算法或 LRU 最近最久未使用算法，具体 LRU 算法的实现是使用静态双向链表，链表表头表示最近被访问的块，而链表表尾则表示最久未访问块，当一个块被访问时，被访问块就被更新为新表头。

4 结束语

SSD 已经广泛应用于计算机、手机、摄像机以及工作站等，SSD 为这些设备提供了高速可靠的存储，随着 SSD 技术的不断提高、大规模的生产，SSD 的价格也越来越低，性价比逐年攀升。

这篇文章的目的就是总结已发明的 SSD 闪存技术，希望能对新的 SSD 闪存技术的研究和开发提供更好的参考。

参考文献：

[1] Assar, M., Nemazie, S., and Estakhri, P. 1995a. Flash memory mass storage architecture. US patent 5,388,083. Filed March 26, 1993; Issued February 7, 1995; Assigned to Cirrus Logic.

[2] Assar, M., Nemazie, S., and Estakhri, P. 1995b. Flash memory mass storage architecture incorporation wear leveling technique. US patent 5,479,638. Filed March 26, 1993; IssuedDecember 26, 1995; Assigned to Cirrus Logic.

[3] Assar, M., Nemazie, S., and Estakhri, P. 1996. Flash memory mass storage architecture incorporation wear leveling technique without using CAM cells. US patent 5,485,595. Filed October 4, 1993; Issued January 16, 1996; Assigned to Cirrus Logic.

[4] Ban, A. 1995. Flash file system. US patent 5,404,485. Filed March 8, 1993; Issued April 4, 1995; Assigned to M-Systems.

[5] Ban, A. 1999. Flash file system optimized for page-mode flash technologies. US patent 5,937,425. Filed October 16, 1997; Issued August 10, 1999; Assigned toM-Systems.

[6] Ban, A. 2004. Wear leveling of static areas in flash memory. US patent 6,732,221. Filed June 1, 2001; Issued May 4, 2004; Assigned to M-Systems.

[7] Barrett, P. L., Quinn, S. D., and Lipe, R. A. 1995. System for updating data stored on a flash-erasable, programmable, read-only memory (FEPROM) based upon predetermined bit value of indicating pointers. US patent 5,392,427. Filed May 18, 1993; Issued February 21, 1995; Assigned to Microsoft.

[8] Chang, L.-P., Kuo, T.-W., and Lo, S.-W. 2004. Real-time garbage collection for flash-memory storage systems of real-time embedded systems. ACM Transactions on Embedded Computing Systems 3, 4, 837–863.

嵌入式资源免费下载

总线协议：

1. 基于 PCIe 驱动程序的数据传输卡 DMA 传输
2. 基于 PCIe 总线协议的设备驱动开发
3. CANopen 协议介绍
4. 基于 PXI 总线 RS422 数据通信卡 WDM 驱动程序设计
5. FPGA 实现 PCIe 总线 DMA 设计
6. PCI Express 协议实现与验证
7. VPX 总线技术及其实现
8. 基于 Xilinx FPGA 的 PCIE 接口实现
9. 基于 PCI 总线的 GPS 授时卡设计
10. 基于 CPCI 标准的 6U 信号处理平台的设计
11. USB30 电路保护
12. USB30 协议分析与框架设计
13. USB 30 中的 CRC 校验原理及实现
14. 基于 CPLD 的 UART 设计
15. IPMI 在 VPX 系统中的应用与设计
16. 基于 CPCI 总线的 PMC 载板设计
17. 基于 VPX 总线的工件台运动控制系统研究与开发
18. PCI Express 流控机制的研究与实现
19. UART16C554 的设计
20. 基于 VPX 的高性能计算机设计
21. 基于 CAN 总线技术的嵌入式网关设计
22. Visual C 串行通讯控件使用方法与技巧的研究
23. IEEE1588 精密时钟同步关键技术研究
24. GPS 信号发生器射频模块的一种实现方案
25. 基于 CPCI 接口的视频采集卡的设计
26. 基于 VPX 的 3U 信号处理平台的设计
27. 基于 PCI Express 总线 1394b 网络传输系统 WDM 驱动设计
28. AT89C52 单片机与 ARINC429 航空总线接口设计
29. 基于 CPCI 总线多 DSP 系统的高速主机接口设计
30. 总线协议中的 CRC 及其在 SATA 通信技术中的应用
31. 基于 FPGA 的 SATA 硬盘加解密控制器设计
32. Modbus 协议在串口通讯中的研究及应用
33. 高可用的磁盘阵列 Cache 的设计和实现
34. RAID 阵列中高速 Cache 管理的优化

35. 一种新的基于 RAID 的 CACHE 技术研究与实现
36. 基于 PCIE-104 总线的高速数据接口设计
37. 基于 VPX 标准的 RapidIO 交换和 Flash 存储模块设计
38. 北斗卫星系统在海洋工程中的应用
39. 北斗卫星系统在远洋船舶上应用的研究
40. 基于 CPCI 总线的红外实时信号处理系统
41. 硬件实现 RAID 与软件实现 RAID 的比较
42. 基于 PCI Express 总线系统的热插拔设计
43. 基于 RAID5 的磁盘阵列 Cache 的研究与实现
44. 基于 PCI 总线的 MPEG2 码流播放卡驱动程序开发
45. 基于磁盘异或引擎的 RAID5 小写性能优化
46. 基于 IEEE1588 的时钟同步技术研究
47. 基于 Davinci 平台的 SD 卡读写优化
48. 基于 PCI 总线的图像处理及传输系统的设计
49. 串口和以太网通信技术在油液在线监测系统中的应用
50. USB30 数据传输协议分析及实现
51. IEEE 1588 协议在工业以太网中的实现
52. 基于 USB30 的设备自定义请求实现方法
53. IEEE1588 协议在网络测控系统中的应用
54. USB30 物理层中弹性缓冲的设计与实现
55. USB30 的高速信息传输瓶颈研究
56. 基于 IPv6 的 UDP 通信的实现
57. 一种基于 IPv6 的流媒体传送方案研究与实现
58. 基于 IPv4-IPv6 双栈的 MODBUS-TCP 协议实现
59. RS485CAN 网关设计与实现
60. MVB 周期信息的实时调度
61. RS485 和 PROFINET 网关设计
62. 基于 IPv6 的 Socket 通信的实现
63. MVB 网络重复器的设计
64. 一种新型 MVB 通信板的探究
65. 具有 MVB 接口的输入输出设备的分析
66. 基于 STM32 的 GSM 模块综合应用
67. 基于 ARM7 的 MVB CAN 网关设计
68. 机车车辆的 MVB CAN 总线网关设计
69. 智能变电站冗余网络中 IEEE1588 协议的应用
70. CAN 总线的浅析 CANopen 协议
71. 基于 CANopen 协议实现多电机系统实时控制
72. 以太网时钟同步协议的研究
73. 基于 CANopen 的列车通信网络实现研究
74. 基于 SJA1000 的 CAN 总线智能控制系统设计
75. 基于 CANopen 的运动控制单元的设计
76. 基于 STM32F107VC 的 IEEE 1588 精密时钟同步分析与实现

77. 分布式控制系统精确时钟同步技术
78. 基于 IEEE 1588 的时钟同步技术在分布式系统中应用
79. 基于 SJA1000 的 CAN 总线通讯模块的实现
80. 嵌入式设备的精确时钟同步技术的研究与实现
81. 基于 SJA1000 的 CAN 网桥设计
82. 基于 CAN 总线分布式温室监控系统的设计与实现
83. 基于 DSP 的 CANopen 通讯协议的实现
84. 基于 PCI9656 控制芯片的高速网卡 DMA 设计
85. 基于以太网及串口的数据采集模块设计
86. MVB1 类设备控制器的 FPGA 设计
87. MVB 接口彩色液晶显示诊断单元的显示应用软件设计
88. IPv6 新型套接字的网络编程剖析
89. 基于规则的 IPv4 源程序到 IPv6 源程序的移植方法
90. MVB 网络接口单元的 SOC 解决方案
91. 基于 IPSec 协议的 IPv6 安全研究
92. 具有 VME 总线的车载安全计算机 MVB 通信板卡
93. SD 卡的传输协议和读写程序
94. 基于 SCTP 的 TLS 应用
95. 基于 IPv6 的静态路由实验设计
96. 基于 MVB 的地铁列车司机显示系统研究
97. 基于参数优化批处理的 TLS 协议

VxWorks：

1. 基于 VxWorks 的多任务程序设计
2. 基于 VxWorks 的数据采集存储装置设计
3. Flash 文件系统分析及其在 VxWorks 中的实现
4. VxWorks 多任务编程中的异常研究
5. VxWorks 应用技巧两例
6. 一种基于 VxWorks 的飞行仿真实时管理系统
7. 在 VxWorks 系统中使用 TrueType 字库
8. 基于 FreeType 的 VxWorks 中文显示方案
9. 基于 Tilcon 的 VxWorks 简单动画开发
10. 基于 Tilcon 的某武器显控系统界面设计
11. 基于 Tilcon 的综合导航信息处理装置界面设计
12. VxWorks 的内存配置和管理
13. 基于 VxWorks 系统的 PCI 配置与应用
14. 基于 MPC8270 的 VxWorks BSP 的移植

15. Bootrom 功能改进经验谈
16. 基于 VxWorks 嵌入式系统的中文平台研究与实现
17. VxBus 的 A429 接口驱动
18. 基于 VxBus 和 MPC8569E 千兆网驱动开发和实现
19. 一种基于 vxBus 的 PPC 与 FPGA 高速互联的驱动设计方法
20. 基于 VxBus 的设备驱动开发
21. 基于 VxBus 的驱动程序架构分析
22. 基于 VxBus 的高速数据采集卡驱动程序开发
23. Vxworks 下的冗余 CAN 通讯模块设计
24. WindML 工业平台下开发 S1d13506 驱动及显示功能的实现
25. WindML 中 Mesa 的应用
26. VxWorks 下图形用户界面开发中双缓冲技术应用
27. VxWorks 上的一种 GUI 系统的设计与实现
28. VxWorks 环境下 socket 的实现
29. VxWorks 的 WindML 图形界面程序的框架分析
30. VxWorks 实时操作系统及其在 PC104 下以太网编程的应用
31. 实时操作系统任务调度策略的研究与设计
32. 军事指挥系统中 VxWorks 下汉字显示技术
33. 基于 VxWorks 实时控制系统中文交互界面开发平台
34. 基于 VxWorks 操作系统的 WindML 图形操控界面实现方法
35. 基于 GPU FPGA 芯片原型的 VxWorks 下驱动软件开发
36. VxWorks 下的多串口卡设计
37. VxWorks 内存管理机制的研究
38. T9 输入法在 Tilcon 下的实现
39. 基于 VxWorks 的 WindML 图形界面开发方法
40. 基于 Tilcon 的 IO 控制板可视化测试软件的设计和实现
41. 基于 VxWorks 的通信服务器实时多任务软件设计
42. 基于 VXWORKS 的 RS485MVB 网关的设计与实现
43. 实时操作系统 VxWorks 在微机保护中的应用
44. 基于 VxWorks 的多任务程序设计及通信管理
45. 基于 Tilcon 的 VxWorks 图形界面开发技术
46. 嵌入式图形系统 Tilcon 及应用研究
47. 基于 VxWorks 的数据采集与重演软件的图形界面的设计与实现
48. 基于嵌入式的 Tilcon 用户图形界面设计与开发
49. 基于 Tilcon 的交互式多页面的设计
50. 基于 Tilcon 的嵌入式系统人机界面开发技术
51. 基于 Tilcon 的指控系统多任务人机交互软件设计
52. 基于 Tilcon 航海标绘台界面设计
53. 基于 Tornado 和 Tilcon 的嵌入式 GIS 图形编辑软件的开发
54. VxWorks 环境下内存文件系统的应用
55. VxWorks 下的多重定时器设计
56. Freescale 的 MPC8641D 的 VxWorks BSP

57. VxWorks 实验五[时间片轮转调度]
58. 解决 VmWare 下下载大型工程.out 出现 WTX Error 0x100de 的问题
59. 基于 VxWorks 系统的 MiniGUI 图形界面开发
60. VxWorks BSP 开发中的 PCI 配置方法
61. VxWorks 在 S3C2410 上的 BSP 设计
62. VxWorks 操作系统中 PCI 总线驱动程序的设计与实现
63. VxWorks 概述
64. 基于 AT91RM9200 的 VxWorks END 网络驱动开发
65. 基于 EBD9200 的 VxWorks BSP 设计和实现
66. 基于 VxWorks 的 BSP 技术分析
67. ARM LPC2210 的 VxWorks BSP 源码
68. 基于 LPC2210 的 VxWorks BSP 移植
69. 基于 VxWorks 平台的 SCTP 协议软件设计实现
70. VxWorks 快速启动的实现方法[上电到应用程序 1 秒]

Linux：

1. Linux 程序设计第三版及源代码
2. NAND FLASH 文件系统的设计与实现
3. 多通道串行通信设备的 Linux 驱动程序实现
4. Zsh 开发指南-数组
5. 常用 GDB 命令中文速览
6. 嵌入式 C 进阶之道
7. Linux 串口编程实例
8. 基于 Yocto Project 的嵌入式应用设计
9. Android 应用的反编译
10. 基于 Android 行为的加密应用系统研究
11. 嵌入式 Linux 系统移植步步通
12. 嵌入式 CC++语言精华文章集锦
13. 基于 Linux 的高性能服务器端的设计与研究
14. S3C6410 移植 Android 内核
15. Android 开发指南中文版
16. 图解 Linux 操作系统架构设计与实现原理（第二版）
17. 如何在 Ubuntu 和 Linux Mint 下轻松升级 Linux 内核
18. Android 简单 mp3 播放器源码
19. 嵌入式 Linux 系统实时性的研究
20. Android 嵌入式系统架构及内核浅析
21. 基于嵌入式 Linux 操作系统内核实时性的改进方法研究
22. Linux TCP IP 协议详解

23. Linux 桌面环境下内存去重技术的研究与实现
24. 掌握 Android 7.0 新增特性 Quick Settings
25. Android 应用逆向分析方法研究
26. Android 操作系统的课程教学
27. Android 智能手机操作系统的研究
28. Android 英文朗读功能的实现
29. 基于 Yocto 订制嵌入式 Linux 发行版
30. 基于嵌入式 Linux 的网络设备驱动设计与实现
31. 如何高效学习嵌入式
32. 基于 Android 平台的 GPS 定位系统的设计与实现
33. LINUX ARM 下的 USB 驱动开发
34. Linux 下基于 I2C 协议的 RTC 驱动开发
35. 嵌入式下 Linux 系统设备驱动程序的开发
36. 基于嵌入式 Linux 的 SD 卡驱动程序的设计与实现
37. Linux 系统中进程调度策略
38. 嵌入式 Linux 实时性方法
39. 基于实时 Linux 计算机联锁系统实时性分析与改进
40. 基于嵌入式 Linux 下的 USB30 驱动程序开发方法研究
41. Android 手机应用开发之音乐资源播放器
42. Linux 下以太网的 IPv6 隧道技术的实现
43. Research and design of mobile learning platform based on Android
44. 基于 linux 和 Qt 的串口通信调试器调的设计及应用
45. 在 Linux 平台上基于 QT 的动态图像采集系统的设计
46. 基于 Android 平台的医护查房系统的研究与设计
47. 基于 Android 平台的软件自动化监控工具的设计开发
48. 基于 Android 的视频软硬解码及渲染的对比研究与实现
49. 基于 Android 移动设备的加速度传感器技术研究
50. 基于 Android 系统振动测试仪研究
51. 基于缓存竞争优化的 Linux 进程调度策略
52. Linux 基于 W83697 和 W83977 的 UART 串口驱动开发文档
53. 基于 AT91RM9200 的嵌入式 Linux 系统的移植与实现
54. 路由信息协议在 Linux 平台上的实现
55. Linux 下 IPv6 高级路由器的实现

Windows CE:

1. Windows CE.NET 下 YAFFS 文件系统 NAND Flash 驱动程序设计
2. Windows CE 的 CAN 总线驱动程序设计
3. 基于 Windows CE.NET 的 ADC 驱动程序实现与应用的研究

4. 基于 Windows CE.NET 平台的串行通信实现
5. 基于 Windows CE.NET 下的 GPRS 模块的研究与开发
6. win2k 下 NTFS 分区用 ntldr 加载进 dos 源代码
7. Windows 下的 USB 设备驱动程序开发
8. WinCE 的大容量程控数据传输解决方案设计
9. WinCE6.0 安装开发详解
10. DOS 下仿 Windows 的自带计算器程序 C 源码
11. G726 局域网语音通话程序和源代码
12. WinCE 主板加载第三方驱动程序的方法
13. WinCE 下的注册表编辑程序和源代码
14. WinCE 串口通信源代码
15. WINCE 的 SD 卡程序[可实现读写的源码]
16. 基于 WinCE 的 BootLoader 研究
17. Windows CE 环境下无线网卡的自动安装
18. 基于 Windows CE 的可视电话的研究与实现
19. 基于 WinCE 的嵌入式图像采集系统设计
20. 基于 ARM 与 WinCE 的掌纹鉴别系统
21. DCOM 协议在网络冗余环境下的应用
22. Windows XP Embedded 在变电站通信管理机中的应用
23. XPE 在多功能显控台上的开发与应用
24. 基于 Windows XP Embedded 的 LKJ2000 仿真系统设计与实现
25. 虚拟仪器的 Windows XP Embedded 操作系统开发
26. 基于 EVC 的嵌入式导航电子地图设计
27. 基于 XPEmbedded 的警务区 SMS 指挥平台的设计与实现
28. 基于 XPE 的数字残币兑换工具开发
29. Windows CENET 下 ADC 驱动开发设计
30. Windows CE 下 USB 设备流驱动开发与设计
31. Windows 驱动程序设计
32. 基于 Windows CE 的 GPS 应用
33. 基于 Windows CE 下大像素图像分块显示算法的研究
34. 基于 Windows CE 的数控软件开发与实现

PowerPC：

1. Freescale MPC8536 开发板原理图
2. 基于 MPC8548E 的固件设计
3. 基于 MPC8548E 的嵌入式数据处理系统设计
4. 基于 PowerPC 嵌入式网络通信平台的实现

5. PowerPC 在车辆显控系统中的应用
6. 基于 PowerPC 的单板计算机的设计
7. 用 PowerPC860 实现 FPGA 配置
8. 基于 MPC8247 嵌入式电力交换系统的设计与实现
9. 基于设备树的 MPC8247 嵌入式 Linux 系统开发
10. 基于 MPC8313E 嵌入式系统 UBoot 的移植
11. 基于 PowerPC 处理器 SMP 系统的 UBoot 移植
12. 基于 PowerPC 双核处理器嵌入式系统 UBoot 移植
13. 基于 PowerPC 的雷达通用处理机设计
14. PowerPC 平台引导加载程序的移植
15. 基于 PowerPC 嵌入式内核的多串口通信扩展设计
16. 基于 PowerPC 的多网口系统抗干扰设计
17. 基于 MPC860T 与 VxWorks 的图形界面设计
18. 基于 MPC8260 处理器的 PPMC 系统
19. 基于 PowerPC 的控制器研究与设计
20. 基于 PowerPC 的模拟量输入接口扩展
21. 基于 PowerPC 的车载通信系统设计
22. 基于 PowerPC 的嵌入式系统中通用 IO 口的扩展方法
23. 基于 PowerPC440GP 型微控制器的嵌入式系统设计与研究
24. 基于双 PowerPC 7447A 处理器的嵌入式系统硬件设计
25. 基于 PowerPC603e 通用处理模块的设计与实现
26. 嵌入式微机 MPC555 驻留片内监控器的开发与实现
27. 基于 PowerPC 和 DSP 的电能质量在线监测装置的研制
28. 基于 PowerPC 架构多核处理器嵌入式系统硬件设计
29. 基于 PowerPC 的多屏系统设计
30. 基于 PowerPC 的嵌入式 SMP 系统设计
31. 基于 MPC850 的多功能通信管理器
32. 基于 MPC8640D 处理系统的技术研究
33. 基于双核 MPC8641D 处理器的计算机模块设计
34. 基于 MPC8641D 处理器的对称多处理技术研究

ARM:

1. 基于 DiskOnChip 2000 的驱动程序设计及应用
2. 基于 ARM 体系的 PC-104 总线设计
3. 基于 ARM 的嵌入式系统中断处理机制研究
4. 设计 ARM 的中断处理
5. 基于 ARM 的数据采集系统并行总线的驱动设计

6. S3C2410 下的 TFT LCD 驱动源码
7. STM32 SD 卡移植 FATFS 文件系统源码
8. STM32 ADC 多通道源码
9. ARM Linux 在 EP7312 上的移植
10. ARM 经典 300 问
11. 基于 S5PV210 的频谱监测设备嵌入式系统设计与实现
12. Uboot 中 start.S 源码的指令级的详尽解析
13. 基于 ARM9 的嵌入式 Zigbee 网关设计与实现
14. 基于 S3C6410 处理器的嵌入式 Linux 系统移植
15. CortexA8 平台的μC-OSⅡ及 LwIP 协议栈的移植与实现
16. 基于 ARM 的嵌入式 Linux 无线网卡设备驱动设计
17. ARM S3C2440 Linux ADC 驱动
18. ARM S3C2440 Linux 触摸屏驱动
19. Linux 和 Cortex-A8 的视频处理及数字微波传输系统设计
20. Nand Flash 启动模式下的 Uboot 移植
21. 基于 ARM 处理器的 UART 设计
22. ARM CortexM3 处理器故障的分析与处理
23. ARM 微处理器启动和调试浅析
24. 基于 ARM 系统下映像文件的执行与中断运行机制的实现
25. 中断调用方式的 ARM 二次开发接口设计
26. ARM11 嵌入式系统 Linux 下 LCD 的驱动设计
27. Uboot 在 S3C2440 上的移植
28. 基于 ARM11 的嵌入式无线视频终端的设计
29. 基于 S3C6410 的 Uboot 分析与移植
30. 基于 ARM 嵌入式系统的高保真无损音乐播放器设计
31. UBoot 在 Mini6410 上的移植
32. 基于 ARM11 的嵌入式 Linux NAND FLASH 模拟 U 盘挂载分析与实现
33. 基于 ARM11 的电源完整性分析
34. 基于 ARM S3C6410 的 uboot 分析与移植
35. 基于 S5PC100 移动视频监控终端的设计与实现
36. UBoot 在 AT91RM9200 上的移植简析
37. 基于工控级 AT91RM9200 开发板的 UBoot 移植分析

Hardware：

1. DSP 电源的典型设计
2. 高频脉冲电源设计
3. 电源的综合保护设计
4. 任意波形电源的设计

5. 高速 PCB 信号完整性分析及应用
6. DM642 高速图像采集系统的电磁干扰设计
7. 使用 COMExpress Nano 工控板实现 IP 调度设备
8. 基于 COM Express 架构的数据记录仪的设计与实现
9. 基于 COM Express 的信号系统逻辑运算单元设计
10. 基于 COM Express 的回波预处理模块设计
11. 基于 X86 平台的简单多任务内核的分析与实现
12. 基于 UEFI Shell 的 PreOS Application 的开发与研究
13. 基于 UEFI 固件的恶意代码防范技术研究
14. MIPS 架构计算机平台的支持固件研究
15. 基于 UEFI 固件的攻击验证技术研究
16. 基于 UEFI 的 Application 和 Driver 的分析与开发
17. 基于 UEFI 的可信 BIOS 研究与实现
18. 基于 UEFI 的国产计算机平台 BIOS 研究
19. 基于 UEFI 的安全模块设计分析
20. 基于 FPGA NiosⅡ的等精度频率计设计
21. 基于 FPGA 的 SOPC 设计
22. 基于 SOPC 基本信号产生器的设计与实现
23. 基于龙芯平台的 PMON 研究与开发
24. 基于 X86 平台的嵌入式 BIOS 可配置设计
25. 基于龙芯 2F 架构的 PMON 分析与优化
26. CPU 与 GPU 之间接口电路的设计与实现
27. 基于龙芯 1A 平台的 PMON 源码编译和启动分析
28. 基于 PC104 工控机的嵌入式直流监控装置的设计
29. GPGPU 技术研究与发展
30. GPU 实现的高速 FIR 数字滤波算法
31. 一种基于 CPUGPU 异构计算的混合编程模型
32. 面向 OpenCL 模型的 GPU 性能优化
33. 基于 GPU 的 FDTD 算法
34. 基于 GPU 的瑕疵检测
35. 基于 GPU 通用计算的分析与研究
36. 面向 OpenCL 架构的 GPGPU 量化性能模型
37. 基于 OpenCL 的图像积分图算法优化研究
38. 基于 OpenCL 的均值平移算法在多个众核平台的性能优化研究
39. 基于 OpenCL 的异构系统并行编程
40. 嵌入式系统中热备份双机切换技术研究
41. EFI-Tiano 环境下的 AES 算法应用模型
42. EFI 及其安全性研究
43. 基于 UEFI Shell 的 PreOS Application 的开发与研究

Programming:

1. 计算机软件基础数据结构 - 算法
2. 高级数据结构对算法的优化
3. 零基础学算法
4. Linux 环境下基于 TCP 的 Socket 编程浅析
5. Linux 环境下基于 UDP 的 socket 编程浅析
6. 基于 Socket 的网络编程技术及其实现
7. 数据结构考题 - 第 1 章 绪论
8. 数据结构考题 - 第 2 章 线性表
9. 数据结构考题 - 第 2 章 线性表 - 答案
10. 基于小波变换与偏微分方程的图像分解及边缘检测
11. 基于图像能量的布匹瑕疵检测方法
12. 基于 OpenCL 的拉普拉斯图像增强算法优化研究
13. 异构平台上基于 OpenCL 的 FFT 实现与优化
14. 数据结构考题 - 第 4 章 串
15. 数据结构考题 - 第 4 章 串答案
16. 用 IPv6 编程接口实现有连接通信的方法

FPGA / CPLD:

1. 一种基于并行处理器的快速车道线检测系统及 FPGA 实现
2. 基于 FPGA 和 DSP 的 DBF 实现
3. 高速浮点运算单元的 FPGA 实现
4. DLMS 算法的脉动阵结构设计及 FPGA 实现
5. 一种基于 FPGA 的 3DES 加密算法实现
6. 可编程 FIR 滤波器的 FPGA 实现
7. 基于 FPGA 的 AES 加密算法的高速实现
8. 基于 FPGA 的精确时钟同步方法
9. 应用分布式算法在 FPGA 平台实现 FIR 低通滤波器
10. 流水线技术在用 FPGA 实现高速 DSP 运算中的应用
11. 基于 FPGA 的 CAN 总线通信节点设计
12. 基于 FPGA 的高速时钟数据恢复电路的实现
13. 基于 FPGA 的高阶高速 FIR 滤波器设计与实现
14. 基于 FPGA 高效实现 FIR 滤波器的研究
15. FPGA 的 VHDL 设计策略
16. 用 FPGA 实现串口通信的设计
17. GPIB 接口的 FPGA 实现