# NAND存储器中的BCH和LDPC纠错码

如今，NAND闪存在许多方面已经成为我们生活的一部分。有了NAND，存储世界成了全新的世界。事实上，如果没有使用NAND存储器作为存储介质，就不可能拥有智能手机。 继USB密钥和数码相机之后，固态硬盘 (SSD) 现在是闪存的新颠覆性应用。 消费级超轻超薄笔记本电脑需要 NAND 存储，但它在云端和企业服务器中NAND的使用确实是一种范式转变。

由于NAND设备无法制造出没有缺陷的设备，因此使用纠错码(ECCs)一直是一种常见做法。虽然 BCH（Bose-Chaudhuri-Hocquenghem）是消费者应用程序的事实标准，但 LDPC（低密度奇偶校验）代码是企业界的典型选择。在查看平面（2D）超尺度（例如 15 nm）NAND 时尤其如此。一般来说，LDPC提供了更高的纠错能力，但是当带宽要求非常严格时，BCH仍然是一个很好的解决方案。

如前几章所述，3D NAND正在成为市场上的现实。在噪声模型方面，2D和3D有一些共同点：它们都非常复杂，并且在 NAND 的生命周期中会发生变化！

我们确实希望3D NAND将新的失效模型带入局；所有致力于非易失性存储器ECC的科学家都必须尽最大努力尽可能接近香农极限。

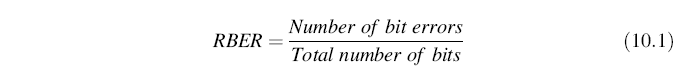
为了做好准备，本章我们将介绍BCH和LDPC码。简单介绍后，我们将看到将这些代码与“真实”的NAND通信通道耦合时的实现问题；同时还将讨论实际的解决方法。

## 简介

在生命周期中，多个来源可能会破坏存储在NAND单元中的数据。最流行的数据恢复方法，有时与其他技术（例如信号处理）结合使用，就是采用纠错码。

ECC为消息添加了冗余项，这样在接收方就可以检测到错误并恢复“最有可能”传输的消息。一组“编码”数据，即添加了冗余项的数据，通常称为码字。

换言之，ECC 可以降低 NAND的原始误码率 (RBER)。RBER定义在式(10.1)中。



一个能够恢复t个错误的ECC，码字错误率，有时称为帧错误率(FER)，如公式(10.2)所示计算。

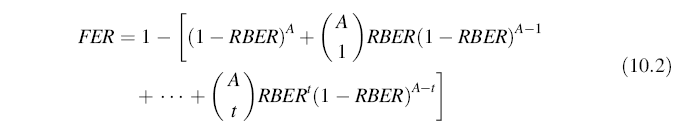
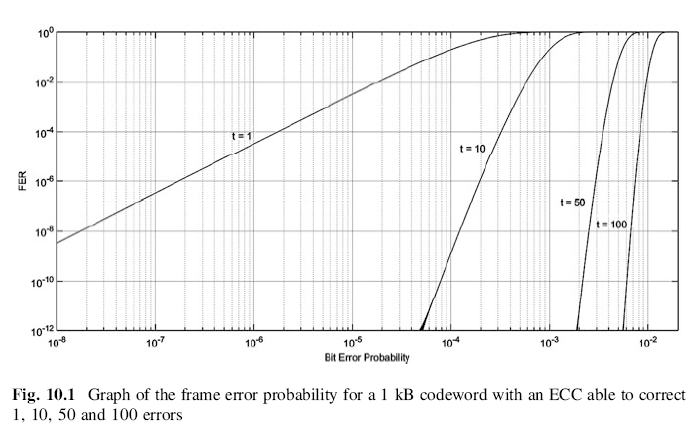
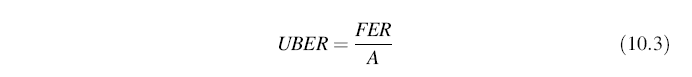
其中A是码字大小。

图 10.1 显示了1 kB码字的FER，其ECC能够纠正1、10、50或100个错误。



用于衡量ECC影响的另一个量是不可纠正的误码率 (UBER)。这在方程(10.3)中定义。

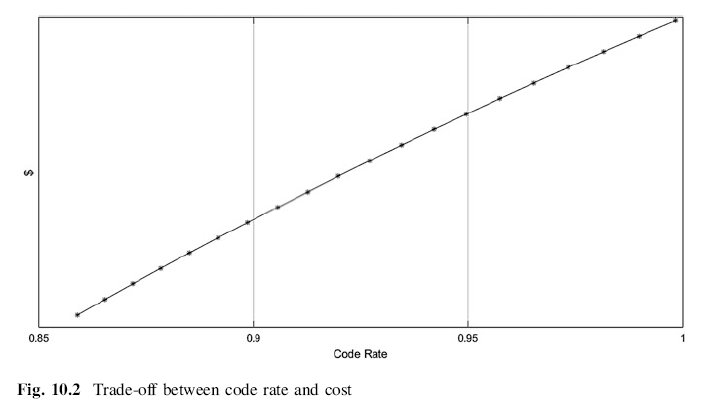


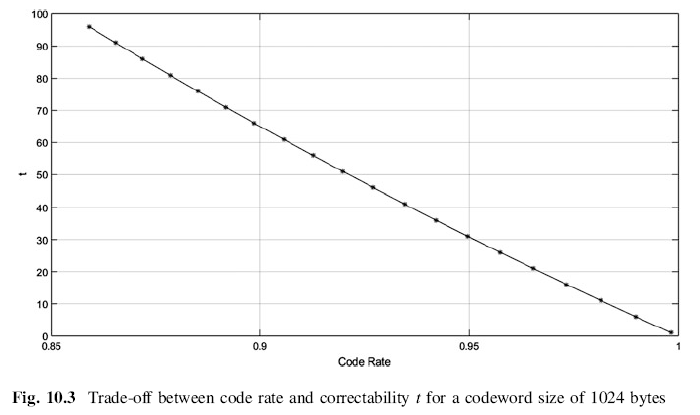
用于决定应用哪个ECC的基本量是码率。码率定义为受保护比特数与传输比特总数（码字大小）之间的比率。如果码率高，则ECC校验位很少，即纠错能力低。另一方面，我们不需要太多额外的空间来存储它们。如果码率低，我们有更多的校验位来保护数据，纠错能力就高。在这种情况下，我们需要更多的额外空间来存储校验位，而在某些情况下这是不可能的。即使这是可能的，也需要花钱。

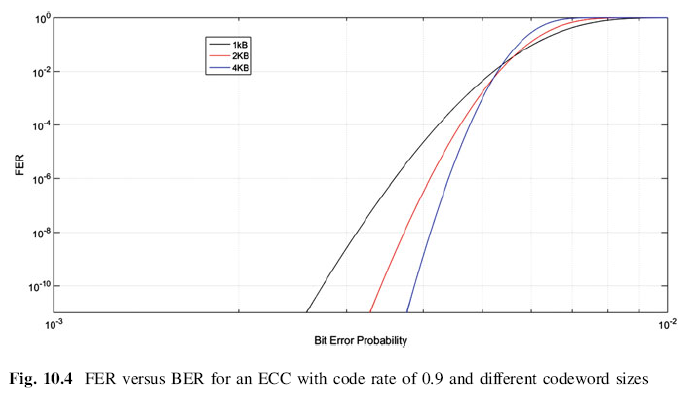
码率和成本（$）之间的权衡如图10.2所示。ECC可纠正性（即每个码字的可纠正位数）是码率的函数，如图10.3所示。较低的码率在芯片面积方面效率较低，但它可以恢复更多的错误。

纠错能力也受码字大小的影响（图10.4）。在相同的码率下，码字越长，纠错能力越高。另一方面，码字越长，ECC硬件越复杂；恢复损坏数据的延迟时间较长是另一个缺点。

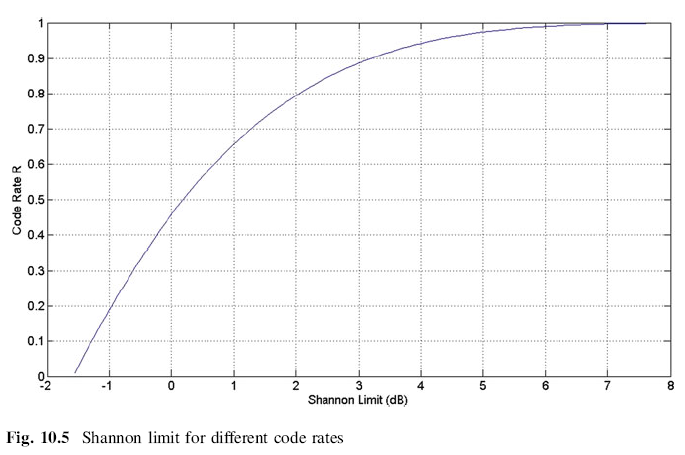
在通信理论中，通常采用信噪比（SNR）来代替RBER。信噪比是一种将所需信号的水平与背景噪声水平进行比较的量度。它被定义为信号功率与噪声功率的比值，通常以分贝表示。比率大于1（即大于0 dB）表示信号多于噪声。







纠错码属于信息论，其“父亲”是香农；他演示了一个称为香农极限[1]的基本定理。该定理根据可实现的信噪比 (SNR) 在码率R的编码系统中建立了一个无差错通信的极限。香农极限的功率如下：如果我们能保证信噪比不超过这个限制，那么我们肯定存在一个能够实现无差错通信的编码系统（速率为R）。不幸的是，没有任何建设性的方式来构建这样一个编码系统。这就是为什么有大量研究活动来寻找尽可能接近香农极限的代码。假设AWGN（加性高斯白噪声）通道和 BPSK（二进制相移键控）调制，可以计算极限（图10.5)。



可实现的SNR可以转换为可实现的BER。香农极限用于评估不同的编码系统：最好的编码是最接近极限的编码系统[2, 3]。

ECC可以分为硬判决码和软判决码。这种区别不是基于代码本身的结构，而是基于代码处理信息的方式。二进制硬判决码以数字方式处理所有数据，即“0”或“1”；换言之，使用一个固定的参考电平将模拟信息转换为数字格式。相反，软决策代码使用可靠性信息来做出决策：例如，读取“0”有90 % 的可靠性而“1”有10 %的可靠性读取。在接下来的部分中，我们将看到软信息如何应用于 NAND 闪存，以及硬代码和软代码之间的比较[3]。

基本上，代码C是通过以单一方式将空间A中长度为k的qk个信息与空间B中长度为n的qk个字相关联而获得的码字集合。如果给定两个码字，它们的和是一个码字，则代码被定义为线性的。当代码是线性的时，编码和解码可以用矩阵运算来描述。

我们将代码C的生成矩阵定义为G。由此可见，所有的码字都可以作为G的行的组合得到。因此，对数据信息m进行编码相当于根据方程式(10.4)将信息m与生成矩阵G相乘。



如果G=(Ik, P)，则G被称为标准形式或系统形式，其中Ik是单位矩阵k×k，P是矩阵k×(n - k)。如果G是标准形式，则码字的前k个符号称为信息符号。

从系统形式的矩阵G可以直接推导出校验矩阵 H=(-PT, In-k)，其中PT是P的转置，它是一个(n - k)×k矩阵，并且In-k是(n - k)×(n - k)的单位矩阵 [4, 5]。

系统码的优点是可以在码字中清楚地识别数据消息，因此可以在解码之前对其进行读取。 对于非系统形式的代码，消息在编码序列中不再可识别，需要具有逆编码功能来识别数据序列。

如果C是具有校验矩阵H的线性码，那么x•HT称为x的校验子。因此，所有码字都有一个等于0的校验子。

校验子是解码的关键。一旦接收到消息r（即从内存中读取），有必要通过计算来确认它是否已损坏：



有两种可能：

* s=0 ⟹ 信息r被认为是正确的；
* s≠0 ⟹ 收到的信息有错误。

在后一种情况下，解码过程开始。

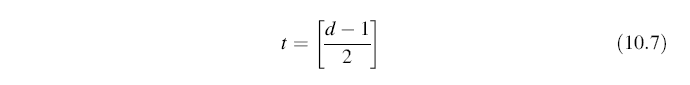
为了了解代码能够纠正和检测多少错误，我们需要一个指标。在编码理论中，它被称为码的最小距离或汉明距离d，它对应于任意两个码字之间不同符号的最小数量。

如果一个码能够将最多包含v个错误的所有消息识别为已损坏，则该代码具有检测能力 v。

检测能力与式(10.6)中描述的最小距离有关。



如果一个代码能够纠正每个组合最多t个错误，则该码具有纠错能力t。纠错能力是用最小距离d用方程(10.7)计算的：



其中方括号代表向下取整函数。

码可以根据应用程序进行操作或组合。增加码最小距离的可能操作是扩展：通过再添加一个校验符号，将代码C[n，k]扩展为代码C'[n+1，k]。一般来说，对于二进制代码，附加校验位是信息的总校验。这被计算为信息所有位的和模2（XOR）。

当代码的“自然”长度不符合应用程序约束条件（例如NAND闪存页）时，可以通过缩短操作对其进行更改：将C[n，k]缩短为代码C'[n− j, k− j] 通过擦除校验矩阵的j列。请注意，删除的列是与用户数据相对应的列。通过此操作，可降低码率。

一个类似的操作，但结果非常不同的是打孔操作。打孔是在编码后去除一些校验位的过程。这与使用更高速率的纠错码进行编码具有相同的效果。好处是可以使用相同的解码器，不管有多少比特被打孔；因此，穿孔大大增加了系统的灵活性，而不会显着增加其复杂性[6]。

## BCH码

BCH 码属于循环代数码中最重要的一类。它们是在1959年由Hocquenghem和1960年由Bose和Ray-Chauduri 独立研究发现的[7, 8]。

对于BCH码，在构建过程中可以确保最小距离。代码本身的定义是基于距离概念和伽罗华域 [9, 10]。

设β为伽罗华域GF(qm)中的一个元素。设b为一个非负整数，一个设计距离为d的BCH码由一个最小次数为d-1的多项式g(x)生成，这个多项式有d-1个β的连续次幂的根：βb，βb+1，...，βb+d-2.设Ψi是βb+i的最小多项式（0≤i＜d-1），g(x)可以如下计算：

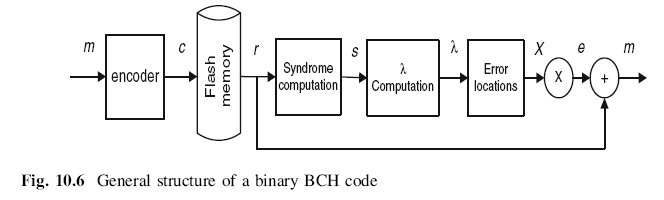


保护的数据大小是k=n-deg(g(x))。

可以证明设计的 d 至少为 2t + 1，因此代码能够纠正 t 个错误。

如果我们假设b=1，并且β是GF(qm)的原始元素，那么码变成长度为qm - 1的狭义原始 BCH码，能够纠正t个错误。现在我们将考虑狭义原始BCH码。

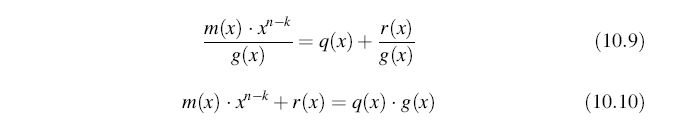
一般来说，BCH码的解码至少比编码复杂10倍。在本章中，我们只处理二进制BCH码，其结构如图10.6所示。



### BCH编码

让我们假设一个BCH码[n, k]具有生成多项式g(x)和要编码的信息m(x)，它被写成k-1次多项式。

首先，信息m(x)乘以xn-k，然后除以g(x)，从而根据等式(10.9)和(10.10)获得商q(x)和余数r(x)。



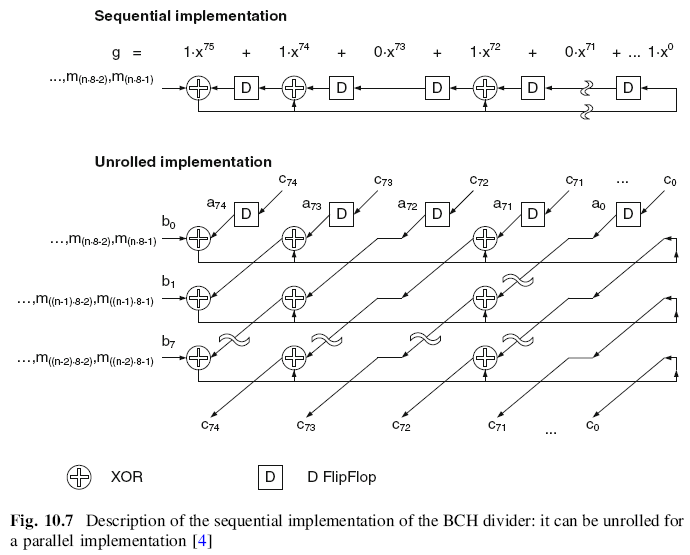
结果，信息m(x)与xn-k相乘产生一个n-1次多项式，其中前n-k个系数现在为空，然后将被校验位占用。

因此，编码字c(x)计算为：



方程式(10.11)的实际实现如图10.7所示。请注意，由于我们正在考虑二进制BCH代码，因此求和实际上是XOR，而乘积是AND。

BCH编码的“自然”结构是顺序的；这在高速实现中不是很好，因为它缓慢地按字节、字或双字进行。图10.7b显示了展开的实现，假设一次处理1个字节[4]。在图中可以看出，每个寄存器的内容不再取决于单个输入，而是取决于整个字节。



### BCH解码

解码操作遵循三个基本步骤，如图10.6所示：

* 计算校验子；
* 计算错误地址多项式的系数(通常用Berlekamp-Massey算法[4, 5])
* 计算错误多项式的根(通常采用钱搜索算法[4，11])

在编码信息的传输（读取）过程中，可能会出现一些错误。错误可以用一个多项式表示，该多项式在每个错误的位置都有系数“1”：

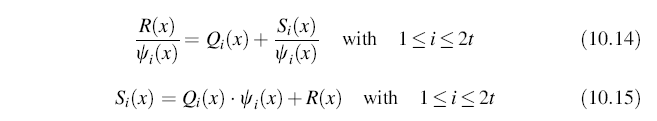


如果ECC可以纠正t个错误，则方程（10.12）中最多允许有t个非零系数。

因此，传输（读取）向量 R(x) 为：



第一个解码步骤包括计算读取信息的2t校正子：



根据方程式(10.14) 和(10.15)，接收到的向量除以形成生成多项式的每个最小多项式Ψi，从而得到商Qi(x)和余数Si(x)，称为校正子。

此时，必须将2t个校验子转换为为元素β，β2，β3，...，β2t，其Ψi是最小多项式。 根据式(10.16)，这个转换是对在β，β2，β3，...，β2t中接收到的信息的转换，因为根据最小多项式的定义，Ψi (βi) = 0(对于 1 ≤ i ≤ 2t) 。



因此，第i个校正子可以计算为接收到的信息与最小多项式Ψi之间除法的余数，然后在 βi 中进行转换，或者作为接收到的消息在 βi 中转换。

在没有任何错误的情况下，接收到的多项式是一个码字：因此，等式（10.14）除法的余数为空，所有校验子都为空。 验证校验子是否都为空是为了了解读取的消息是否是码字（或是否发生了某些错误）的充分必要条件。

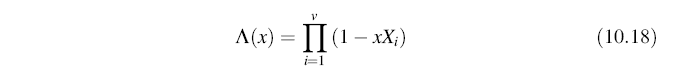
对于二进制代码，我们使用以下特性：



所以我们可以只计算t个校验子。

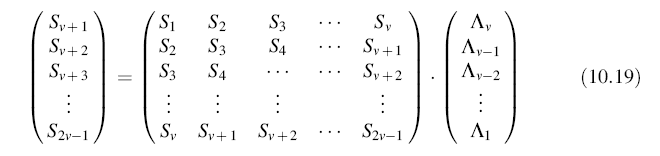
由于校验子被计算为伽罗华域中两个多项式之间除法的余数，因此很容易理解其实现类似于编码器的实现。

错误定位多项式 Λ(x) 被定义为根是错误位置的倒数的多项式。



错误位置多项式的次数给出了发生的错误数。由于Λ(x)的度数最多为t，在超过t个错误的情况下，Λ(x)可能错误地指示t个或更少的错误。

错误位置多项式的系数通过以下等式与校验子相关联。



一般来说，计算错误位置多项式系数的方法是Berlekamp-Massey算法[4, 12]。

Berlekamp算法的哲学在于以迭代方式通过连续逼近来求解(10.19)中的方程组。

经过2t次迭代，Λ(x)是错误位置多项式；在二进制情况下，可以在t次迭代中执行 Berlekamp算法。在下文中，我们描述了无反转二进制Berlekamp Massey算法的流程图（图 10.8）[13]。

首先，我们将校验子多项式定义为

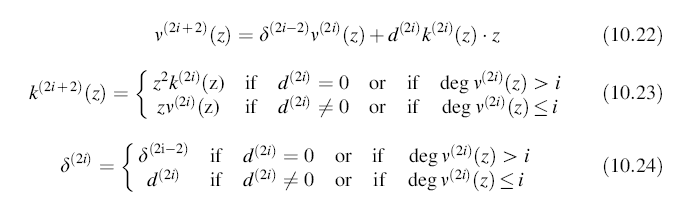


初始条件如下：

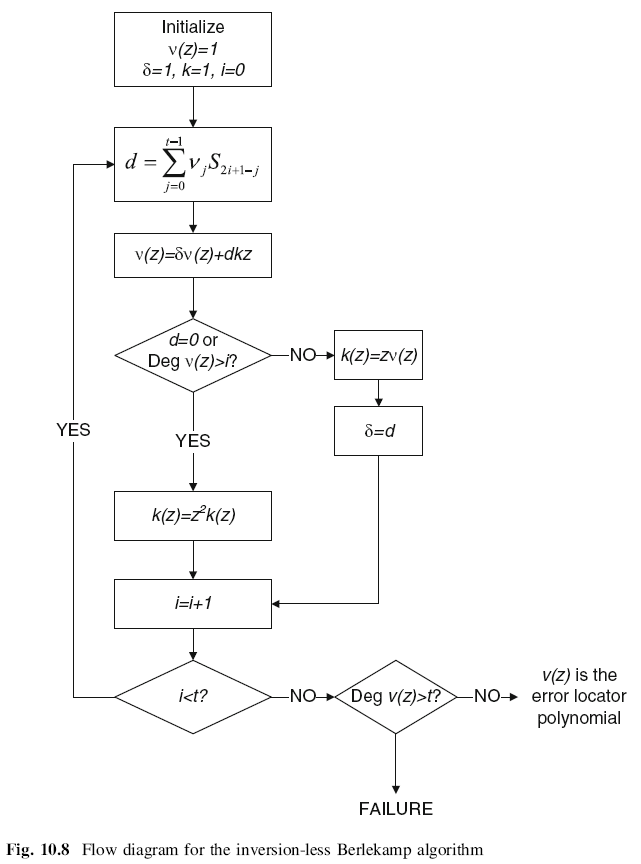


我们将 d(2i) 定义为乘积(1 + S(z))v(2i)(z)中z2i+1的系数。

* 如果S2i+1不存在则算法结束；
* 否则



v(2t)(z)的根与∧(2t)(z)的根重合。

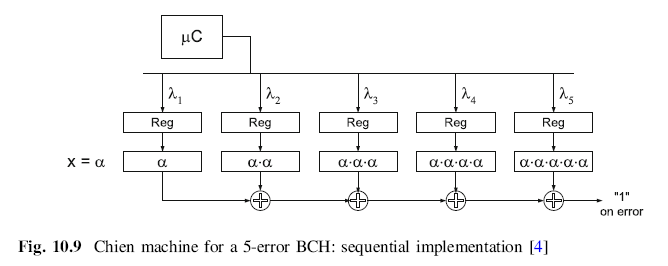


即使算法非常复杂，通常也不需要并行实现，因为在大多数情况下，内存缓冲区的大小和执行延迟都是可以接受的。

解码过程的最后一步包括根据式（10.25）搜索错误位置多项式的根。如果根不重合并且它们属于伽罗华域，那么计算它们的倒数就足以得到误差位置。如果它们重合，或者它们不属于正确的域，则表示接收到的信息与码字的距离大于t。这种情况下，出现了无法纠正的错误图形，解码过程失败。



为了确定多项式的根，Chien机器用域中的所有元素α0、α1、α2、α3、…αN来巧妙地计算∧（x）。对于域的每个元素i，使得多项式为空，相应的位置（2m–1− i）是一个错误位置。Chien机器的可能实现如图10.9所示。



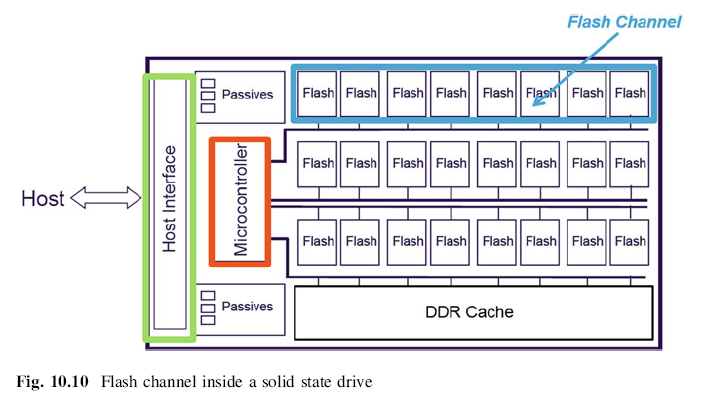
### 多通道BCH

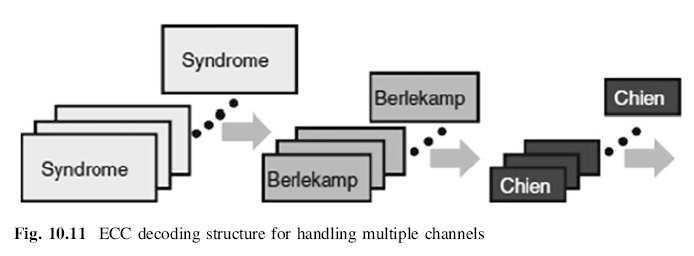
当BCH用于基于NAND的系统如固态驱动器时，必须找到面积和带宽之间的平衡。事实上，SSD并行运行多个NAND设备，以实现其目标性能带宽和IOPS。通常，NAND被分成称为“闪存通道”的组：通道并行工作，读/写/擦除操作可以在同一通道（图10.10）。在此多通道方案中，多重编码考虑到这一点，尤其是超规模的几何图形和多级存储（第3章），始终需要更正（因为RBER较高）。

为了满足带宽要求，最简单的解决方案是每个通道一个编码器和一个解码器。然而，这种方法非常消耗面积，特别是因为解码器。

就编码而言，将来自主机（CPU或操作系统）的数据无延迟地调度到各个通道是非常重要的。从消耗较少的面积开始，有三种可能的方法：

* 所有Flash通道共享一个编码器
* 编码器池
* 每个通道一个编码器





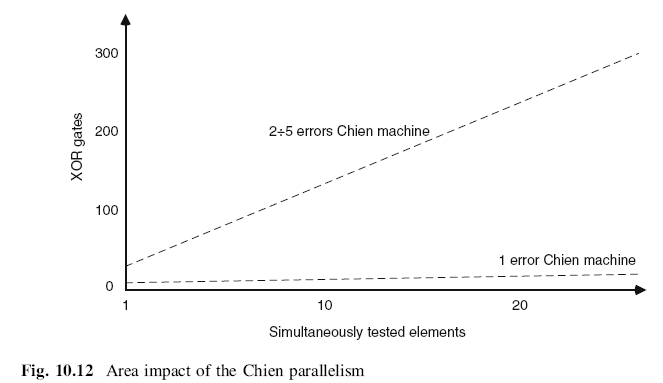
如前所述，正确的硬件选择来自芯片面积和延迟之间的权衡。

现在让我们进入解码阶段。整体结构如图10.11所示。在这种情况下，执行校验子计算、Berlekamp-Massey算法和Chien计算的硬件机器的数量可能不同。

可以用与编码器相同的方式处理综合征计算，因为所有读取的消息都需要这种计算。Berlekamp–Massey算法的执行速度非常快，因为它只需要t次迭代。

如上一节所述，Chien机器一次一个地搜索根。对消息的所有位执行这样的操作会导致非常耗时。解决方案当然是并行架构。与必须以等于输入数据并行度的并行度运行的校验码和校验子计算机不同，Chien 机除了复杂性、面积和功耗之外没有特定限制。在这个并行实现中，在每个计算周期同时评估更多的错误位置。

Chien算法的执行时间通常被系统视为额外的延迟时间。如果出现一个或多个错误的可能性变得相当大，则此延迟会显着影响系统性能。Chien并行度的缺点是对芯片面积的影响，如图 10.12 所示。



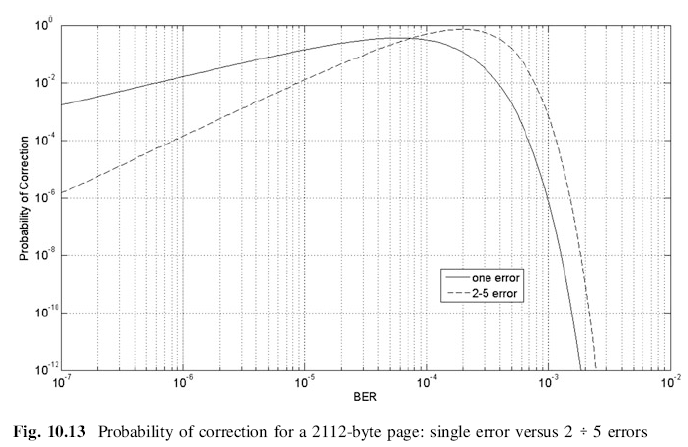
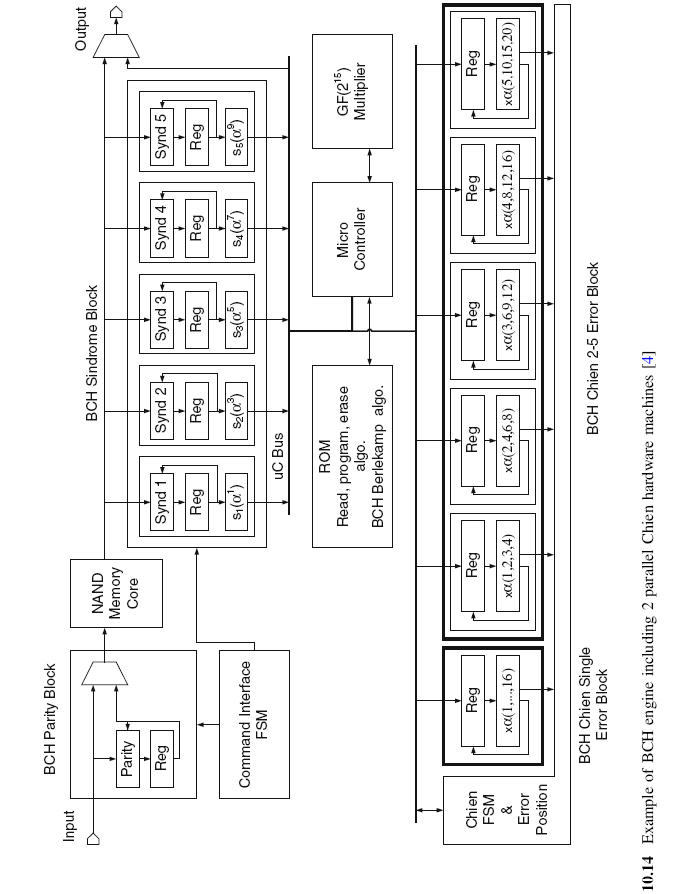


图 10.13 显示，对于2112字节的页面，仅纠正一个错误的概率和纠正 2÷5个错误的概率。假设BER为10-6，我们有单个错误的概率等于1.7×10-2，2÷5错误的概率分别等于1.5 ×10-4。 单个错误的概率肯定更大，并且由于Berlekamp算法准确地指示要纠正的错误数量，因此利用此信息可能很有用。

因此，生成的系统由几台具有不同并行度的Chien机器组成，一台用于纠正单个错误，第二台用于纠正2÷5错误（图 10.14）。

这个解决方案可以乘以任意数量的机器，特别是如果我们正在处理的BCH的纠错能力 t 很高。在这种情况下，给定估计的原始误码率，我们可以计算更可能发生的错误频率t'，并让多个Chien机器以高并行度搜索t'根。相反，定位t\* > t'误差根的硬件机器的数量可以更少，并行度也更小[3, 4]。

当然，上面提到的数字只是一个例子；它们可能会根据NAND技术节点和存储在同一物理单元（例如MLC或TLC）中的位数而发生显着变化。



### 多码率BCH

正如本章介绍中所讨论的，NAND处理在其生命周期内变化的噪声源是典型的。当 NAND是崭新的（即很少的编程/擦除周期，第2章）并且没有记忆时，RBER可能非常低；在使用寿命结束时情况完全不同，即当设备已被读取/擦除/写入多次。因此，希望ECC能够在使用寿命期间改变其校正能力。

有些代码很容易更改码率，而在其他情况下则不是那么简单：BCH 是其中之一，因为它的结构。在本节中，我们提出了一种以最小面积开销构建多码率BCH的方法。

编码器是主要问题。如上所述，校验位被计算为用户数据和生成多项式之间除法的余数，其中后者被计算为t个元素的最小多项式之间的乘积。如果我们想调整一个能够纠正t错误的BCH码来纠正t'错误，其中t' < t，最简单的方法是使用第二个编码器来计算用户数据和生成多项式之间除法的余数，其中后一个计算为t'元素的最小多项式之间的乘法。这种方法的面积开销很大，因为编码面积增加了一倍。

一种更智能且占用面积更少的方法是从t码的校验位中导出t'码的校验位。实际上，对于生成多项式，等式 (10.26) 成立。



校验位r(x)被计算为用户数据c(x)和生成多项式g(t, x)之间除法的余数。根据余数的定义，我们可以得到

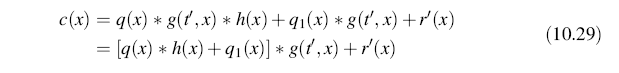


其中q(x)是除法的商，同时deg(r(x)) < deg(g(t, x))

r(x)除以g(t', x)可得



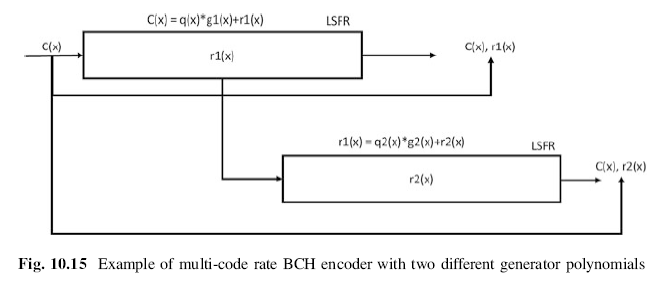
其中 q1(x)是除法的商且deg(r'(x))<deg(g(t', x))。通过替换式(10.27)中的(10.26)和(10.28) 我们得到



很明显，r'(x)是c(x)和g(t', x)之间除法的余数。多码率BCH编码器的电路如图10.15所示。

此实现的开销是可编程LSFR，它将第一次除法的余数除以g(x)因子。当然，我们可以拥有超过2个编码器和多个可编程LSFR。得益于LSFR的可编程性，当NAND比较新时，我们可以选择纠错能力小的BCH码，用户数据用后续的两次除法进行编码，而不再需要除法。

解码要容易得多。校验子由g(t, x)的所有因子计算为不同的除法。如果我们想通过使用 g(t′, x) 的因子来计算校验子，其中g(t′, x)是g(t, x)的因子，那么就可以禁用计算最后 t - t'个校验子的电路。



Berlekamp-Massey算法不受多码率的影响：它在生命开始时以较少的迭代完成，系数为 t'而不是t。

Chien算法完全不受影响。同样，它会在找到t'根而不是t后停止。但是，为了跟上SSD的带宽，多码率环境中可能会实施多钱机器方法（第 10.2.3 节）。

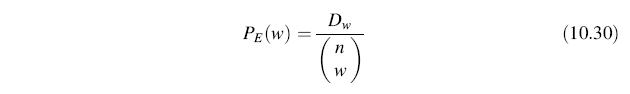
### BCH检测属性

BCH 码不是完美码，因此一个错误数大于t的码字很难在另一个码字的校正范围内移动。 BCH码分离的很好，只有一部分远大于t的错误部分才会重叠它们的纠错区域[3]。因此，只有当接收到的信息位于与原始码字不同的校正范围内时，才会进行错误校正。

给定一个能够纠正t个错误的二进制线性码C，错误纠正概率PME定义为理想有界距离解码器执行错误纠正的概率。加权概率PE(w)是当w个错误发生时执行错误校正的概率。

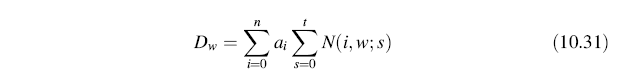
请注意，概率PME取决于码C和传输信道。

定理10.2.1加权概率PE(W)计算为：

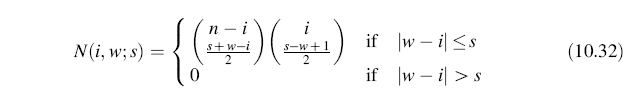


其中 Dw 是可解码字的数量，w在[t + 1, n]范围内。

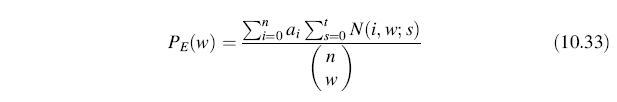
可解码字的数量可以计算为



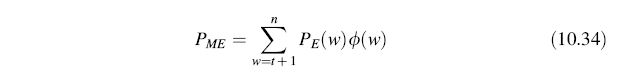
其中 N(i, w; s)是权重为w且与权重为i的字的距离为s的字的数量。这是由式(10.32)计算的。



通过替换式(10.30)中的式(10.30)我们得到：

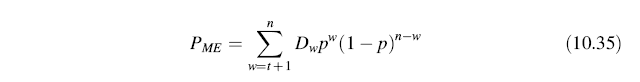


PME是基于PE(w)计算的，如式(10.34)中所述。



其中φ(w)是一个字权重为w的概率。

对一个二进制对称信道BSC

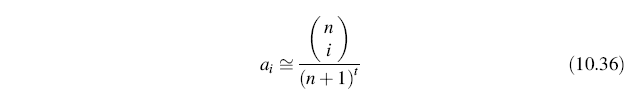


其中，p是bit错误概率。

Dw可以根据式(10.31)计算。不幸的是，对于BCH码，权重ai是未知的，必须进行估计。

有许多不同的定理可以帮助估计BCH码的这些权重。

定理 10.2.2 彼得森估计 长度为n且纠错能力为t的原始BCH码的权重 ai可以近似为

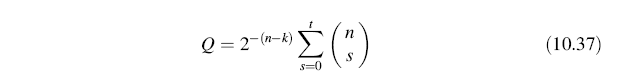


为了有上界，不同的校正项被添加到式(10.36)。

图10.16和10.17显示了基于Peterson估计的BCH[16383,15851,77]的PE和PME。PE和 PME 都表现出单调的行为。

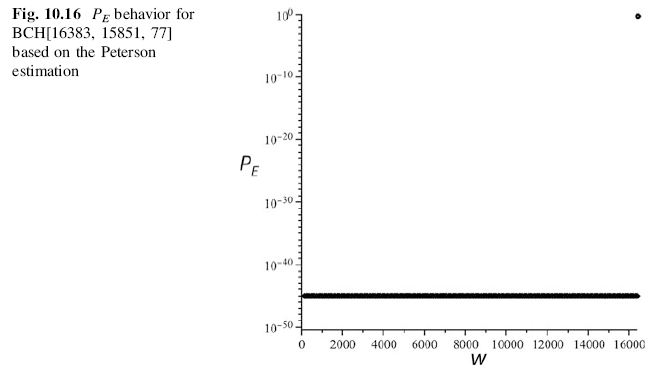
因此，真正的PE和PME行为应该越来越单调，中间有一个长的下限[11]。

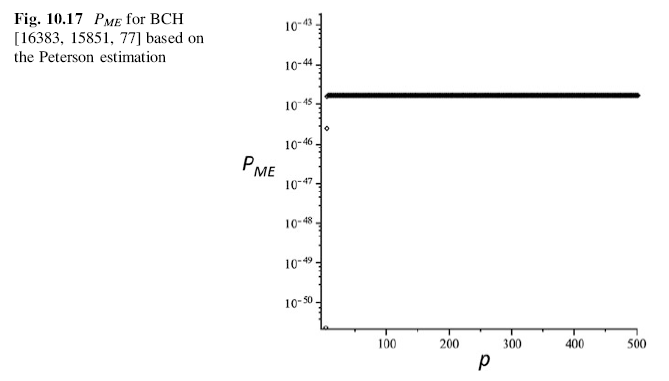
当码长和码率都很高时，这个下限可以近似为



综上所述，BCH码对于长码字具有很好的检测性能；此功能非常适合基于NAND的系统，例如 SSD。事实上，当发生灾难性错误或超出码的纠错能力时，在绝大多数情况下，BCH 会发出解码失败的信号，而不会尝试进行错误纠正。

当然，当 BCH与另一个代码连接时，这种行为变得非常关键。





## 低密度校验码(LDPC)

自从1990年代后期重新被发现以来，LDPC码因其出色的纠错能力而受到了极大的关注，并在许多现实生活中的数据通信和存储应用中得到了广泛的应用。1960年代，Gallager博士发明了LDPC码[15]，其中利用了两个创新思想：迭代解码和约束随机码构造。

LDPC码被称为“容量逼近码”；换句话说，它们是能够达到非常接近香农极限的帧错误率的一类代码。主要原因是强大的软解码，如图10.18和图10.19所示。图10.18显示了2个 BCH码和2个硬解码LDPC码的香农限制。在这种情况下，LDPC没有显示出任何明显的优势，主要是因为两个原因：使用硬而不是软，以及采用的解码算法（即位翻转）[5]。由于软解码，LDPC在图10.19中是明显的赢家。公平地说，事实是软解码推进了香农极限。仔细查看该图会发现，软LDPC非常接近硬香农极限，但仍远离软香农极限。

LDPC是用非常稀疏的校验矩阵H定义的块线性码。每个矩阵都可以转换成其对应的 Tanner图，其中校验的数量等于称为“校验节点”的矩阵行数；还有许多变量节点等于矩阵列的数量。如果矩阵H中对应位置有“1”，则校验节点连接到变量节点。

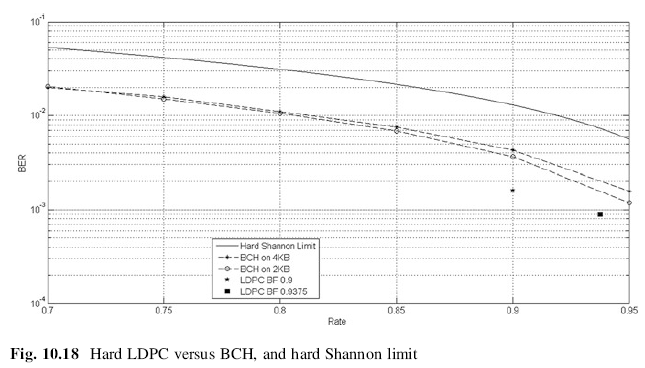


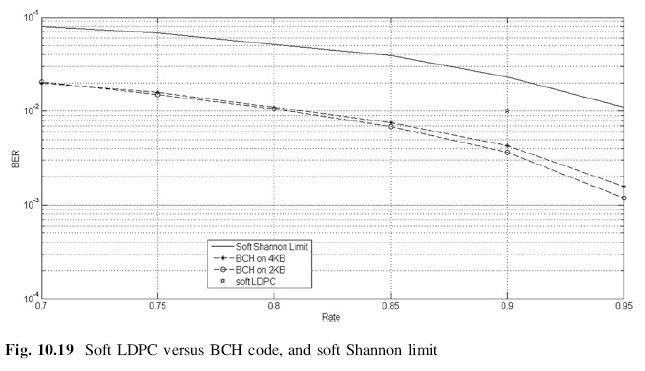
图10.20显示了式(10.38)描述的矩阵的Tanner图。

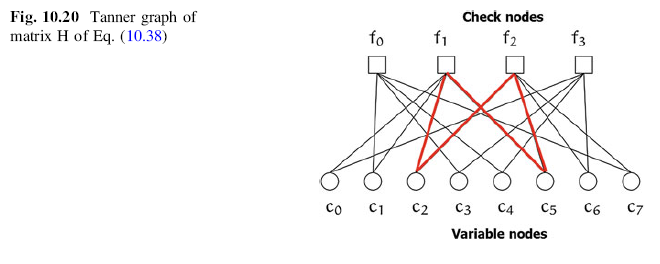
Tanner Graph可以有循环；换句话说，我们可以从一个变量节点开始，然后通过不同的路径回到它。最小环的大小称为LDPC矩阵的周长。在图10.18中，矩阵的周长为4，循环以粗体红色路径显示；等式（10.38）中对应的1用红色圆圈突出显示，它们是矩形的顶点。

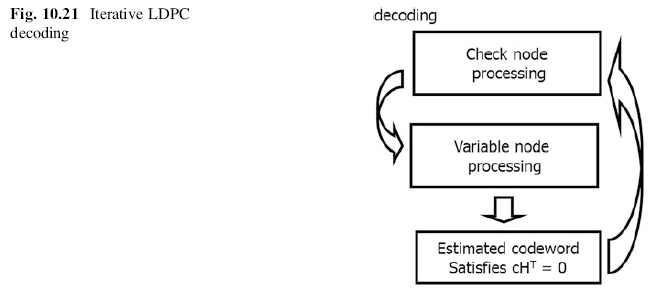
循环在LDPC解码中非常危险，因为在那里解码器可能会被“困住”，无法找到解决方案。

虽然从概念上讲，编码器是传输数据和生成矩阵G之间的乘积，但LDPC码可以通过迭代置信传播 (BP) 算法（也称为Sum-Product或SPA）进行有效解码。BP解码匹配底层代码二分图：解码信息在每个变量节点和校验节点上计算，并通过相邻节点迭代交换（图 10.21）。 在每次迭代结束时，都会产生一个估计的码字；通过将这个临时码字与H相乘，我们可以检查它是否正确。如果是这种情况，则解码停止，否则开始新的迭代。众所周知，如果底层代码二分图不包含太多的短循环，BP解码算法就可以很好地工作。因此，通常要求图是无环的，这相对容易实现。构建高阶无环图绝对不是一件容易的事。

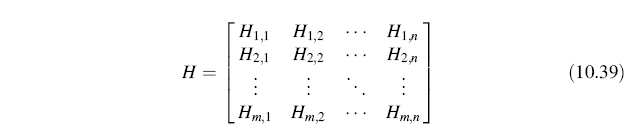








有很多不同的LDPC系列。如果每个变量节点的次数为j且每个校验节点的次数为k，则称LDPC码为 (j, k)正则码。同样也有非正则的代码。为了对闪存有用，LDPC码不仅必须以高码率实现非常低的解码错误率，而且还必须适用于高速VLSI实现，以最小的芯片和功耗成本。已经很好地证明了准循环(QC)LDPC码是此类面向实现的LDPC码的一族。 QC-LDPC码的校验矩阵由循环数组组成。循环是一个方阵，其中每一行是其上一行的循环移位，第一行是最后一行的循环移位。QC-LDPC码的校验矩阵H可以写为



其中每个子矩阵Hi,j是一个二进制循环。闪存等数据存储系统需要非常高的编码率（例如8/9或更高）。已经证明，性能最好的LDPC是非正则的[16-18]。然而，对于高码率，通常使用常规QC-LDPC码，因为它们更容易在硬件中实现。在这种情况下，所有行的1数量相同，所有列的1数量都相同，所有子矩阵Hi,j的列权重都为1或2。由于LDPC码属于错误下限，代码校验矩阵列权重通常为4，甚至更高，以确保足够低的错误下限（例如，错误下限仅发生在10−12的解码失败率以下）[5]。常规和QC-LDPC码校验矩阵的循环结构可以用来大大提高其编码器和解码器的实现效率，如下文所述。

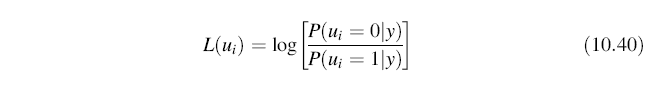
### LDPC码和NAND闪存存储器

Planar TLC NAND最近推动采用LDPC代码，主要是因为NAND原始BER非常高。将 LDPC码调整到NAND特性的复杂性肯定很高。好消息是行业已经付出了努力（在研发方面），今天可以利用LDPC来进一步促进3D进化（缩小）。

NAND环境中的读取操作本质上属于硬类型。读出放大器将单元阈值电压转换为数字值，“0”或“1”（第 3 章）。这就是它不容易提取软信息的原因。

在图10.22中，两个VTH 分布代表两种可能的单元状态：“0”和“1”（假设SLC NAND）。当分布重叠时，会出现错误。硬判决解码器将所有正值读取为0，将负值读取为1，因此图中的重叠区域代表NAND原始BER。但是，A和B是非常不同的错误，因为A有点正，而B远离0。这就表示B比A更可能是错误。通过利用A和B的确切值 ，解码器可以有一个更好的起点。这就是所谓的软信息，它是通过对数似然比（LLR）来衡量的。

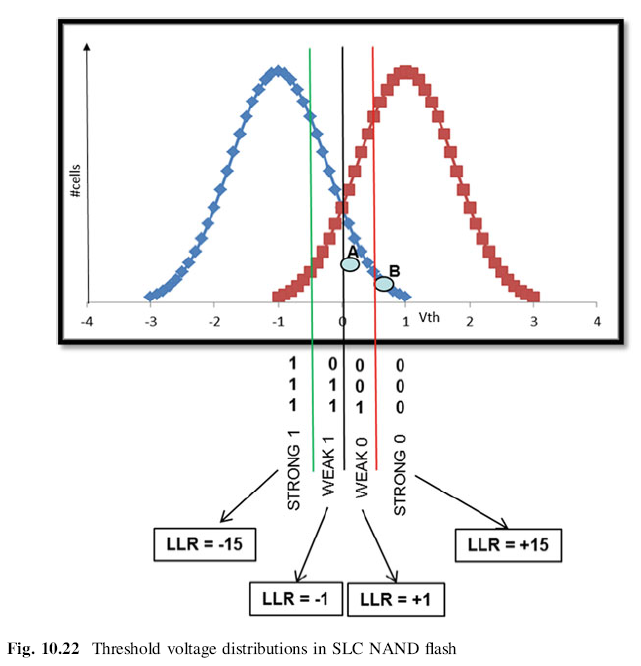
特定值x的LLR是给定读取值y，位x为0的概率与给定读取值y，位x为1的概率之间的对数比。给定这个定义，LLR 可以写成：

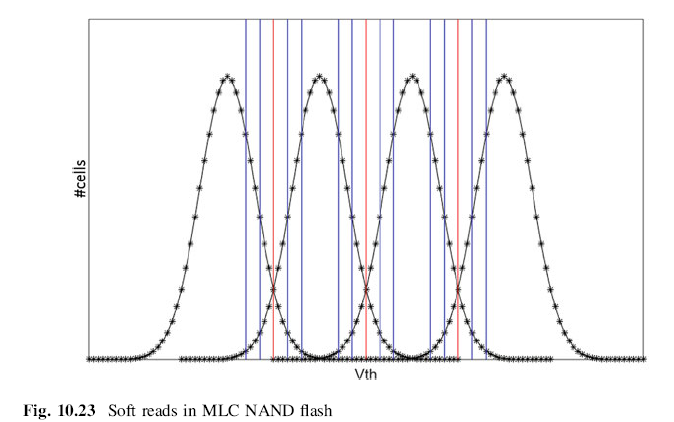


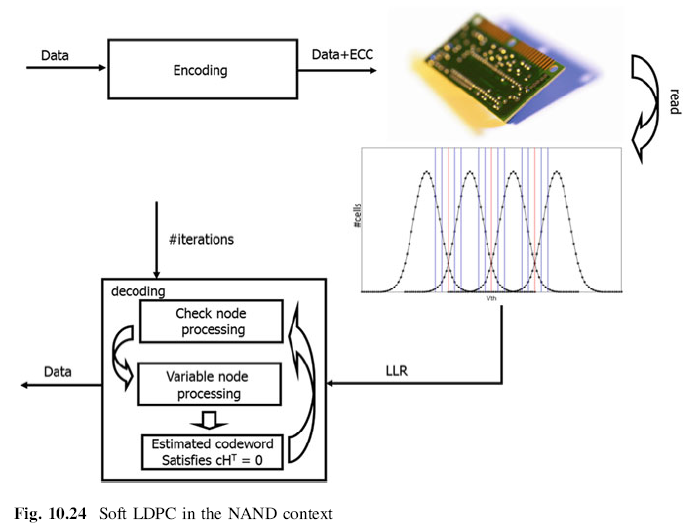
对于NAND，不可能知道阈值电压VTH的确切值。作为近似，通过调整参考电压，每个重叠区域被分成多个切片。图10.23显示了一个MLC NAND，其中每个重叠区域被分成4个切片，因此每个位（LSB和MSB）用3个软位读取。软比特数越高，信息越准确。这种技术是有代价的，因为每个位必须被读取 3 次（在这个例子中）。基本上，软信息要求读取过采样。

为了最大化软信息的回报，有必要仔细了解如何调整每个读取参考电压以及调整多少次，因为每次额外读取都会增加延迟。

LDPC和NAND Flash之间的交互如图 10.24 所示。

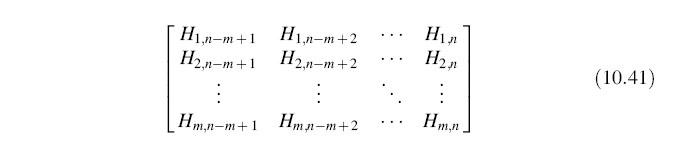




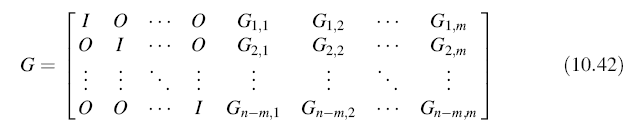


### LDPC码编码

在LDPC编码器设计的上下文中，最直接的方法是将信息位与从稀疏校验矩阵导出的密集生成矩阵相乘。由于非常高的实现复杂性[19]，生成矩阵的密度以及大的代码长度使得生成矩阵-向量乘法的并行实现变得不切实际。因此，部分并行的编码器实现是必须的。然而，对于随机构造的一般非QC LDPC码，它们的密集生成矩阵可能不具有任何可用于开发有效的部分并行编码器架构的结构规律。对于QC-LDPC码，部分并行编码器设计变得更加实惠。 假设QC-LDPC码校验矩阵是一个m×n的循环数组，每个循环是p×p。在最简单的情况下，矩阵的满秩为m·p。我们假设代码校验矩阵可以按列排列，因此以下子数组的满秩为m· p：



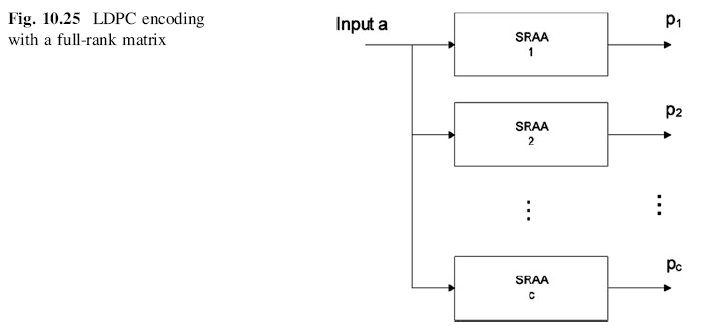
我们还考虑一种系统编码，即每个码字中的前(n–m)·p位是信息位，校验矩阵的前(n–m)·p列对应于(n–m)·p个信息位。因此，相应的生成矩阵具有以下形式：

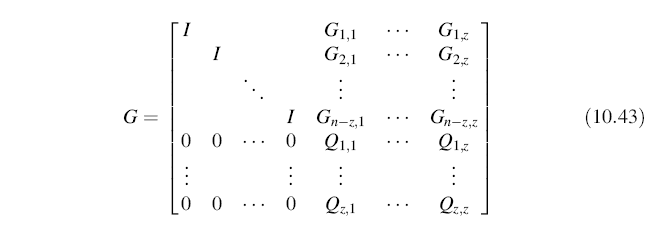


其中I和O表示p×p单位矩阵和p×p零矩阵。作为G生成矩阵，它必须满足H· GT=0，这清楚地表明每个Gi,j也应该是一个p×p循环。

QC-LDPC编码的生成矩阵向量乘法可以通过利用生成矩阵的固有循环结构以部分并行的方式执行图10.25）。

如果矩阵H不是满秩的，则代码是半系统的。换句话说，矩阵G显示在式(10.43)中。



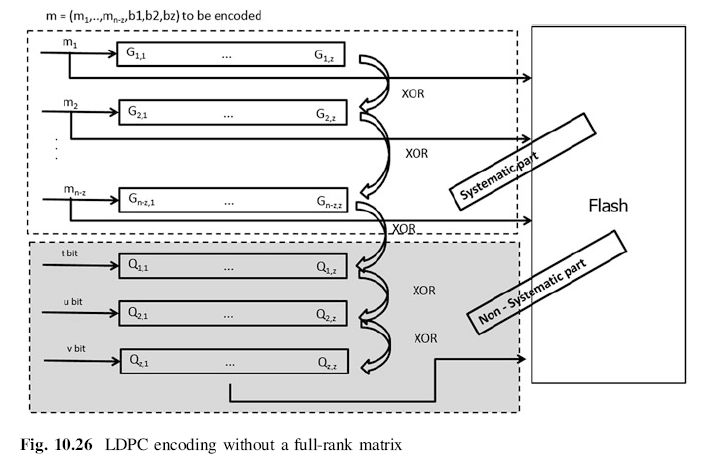


其中由Q表示的部分既不是系统的也不是规则的(在大小上)。

硬件结构如图10.26所示。系统部分相当于满秩H矩阵之一。灰色部分是非系统的并且不正则，因为Qs循环的大小不固定。除此之外，由于其不正则性，使其并行并不容易。

在读取过程中，一旦解码停止，需要将非系统部分乘以 Q-1，以恢复原始数据[20]。

正如所讨论的，半系统的实现比系统的实现要复杂得多。当H不是满秩时，一种可能的解决方法是修复校验部分。校验矩阵H上的校验部分由特定的循环组成。这些循环可以是全零循环，因此矩阵H将不再是规则的。关于QC-LDPC码编码器设计的更详细讨论，读者可以参考[19,21,22]。

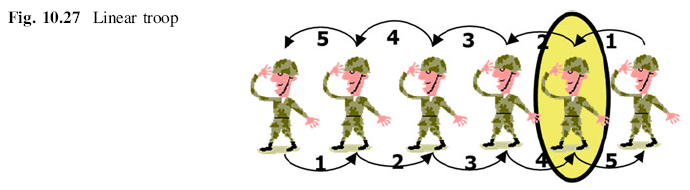


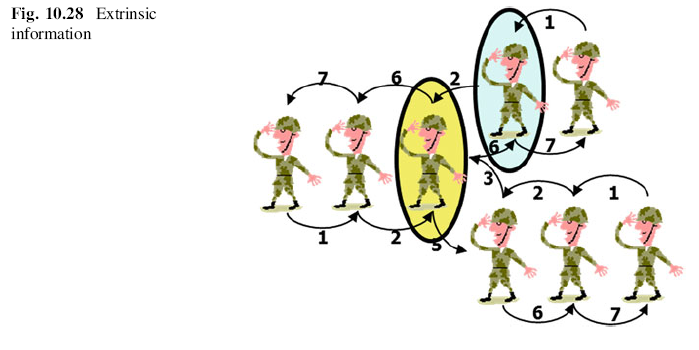
### LDPC码解码

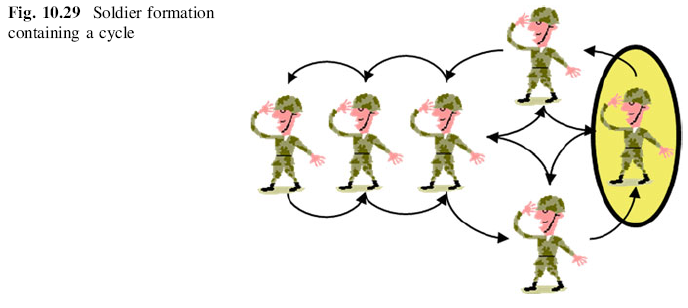
要理解LDPC解码，关键概念之一是外在信息。这里通过一个例子来解释[5]。

我们有一个6名士兵的部队，每个士兵都想知道部队中的士兵总数。在图10.27中，我们有一个线性部队。在这种情况下，每个士兵取后面邻居提供的数字，他加1，然后将结果传送给他前面的邻居。边缘的士兵从没有邻居的一侧收到0。对于每个士兵，接收和发送的数字之和等于士兵总数。

第二个部队（图10.28）稍微复杂一些，它需要不同的规则来传递信息。每个士兵从他的邻居那里取出所有的数字，他加1，然后他减去他想要发送消息的邻居传递的数字。例如，黄色士兵将2+3+2+1-2=6发送给绿色士兵。边缘的士兵从没有邻居的一侧收到0。一名士兵从他的任何一个邻居那里得到的人数加上士兵传递给该邻居的人数之和等于士兵总数。这就引入了外在信息的概念。这个想法是士兵不会将任何信息传递给邻近的士兵。换句话说，只传递外部信息。







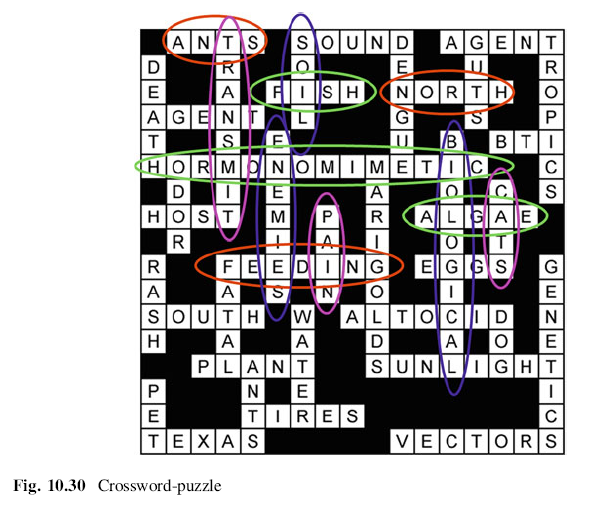
最后一个部队（图10.29）包含一个循环。这种情况是无法解决的：无论人们设计什么计数规则，循环都代表一种正反馈，无论是顺时针还是逆时针方向，因此循环内传递的信息将无限增加。这表明如果图表包含一个或多个循环，则不能声称在图上传递的信息是最优的。 然而，虽然大多数实际代码都包含循环，但众所周知，假设代码设计正确，信息传递解码性能非常好。

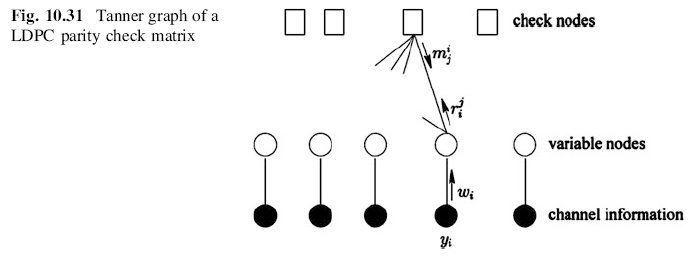
LDPC码背后的关键创新在于校验矩阵的低密度特性，这有助于迭代解码。信息传递解码是指在级联编码方案中以分布式方式对接收到的码字进行解码的低复杂度解码器的集合。通过使用纵横字谜类比，我们可以更好地理解这个句子（图10.30）。

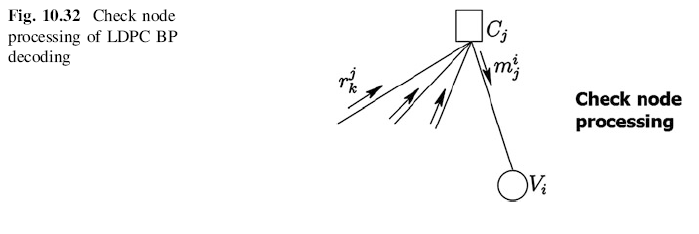
解决填字游戏的过程如下：

* 从我们知道的所有水平词开始 → 红色圆圈；
* 继续使用我们知道的所有垂直单词 → 蓝色圆圈；
* 重新开始，看看我们是否能够完成更多的水平单词，前提是添加了上一步的垂直单词 → 绿色圆圈；
* 重新开始，看看我们是否能够完成更多的垂直单词 → 洋红色圆圈；
* 继续循环，直到填字游戏完成（或找到一个码字），并在我们无法解决它（我们陷入错误下限）或我们太累（我们达到了最大迭代次数）时停止。

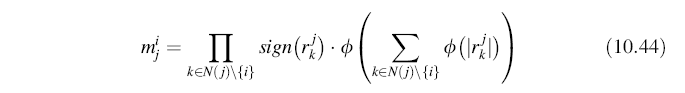
Belief Propagation算法是LDPC的最佳迭代解码方法。为了理解它，考虑校验矩阵的Tanner Graph可能很有用（图10.31）。





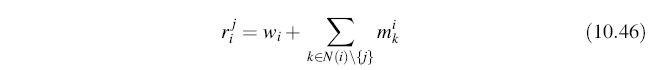


在校验节点处理阶段（图 10.32），每个校验节点必须计算它发送到所连接的变量节点的值 m。值根据式(10.44)计算。



记住士兵的例子，请注意只考虑外部信息：实际上，值 mi 是使用连接到特定校验节点的变量节点发送的所有值计算的，变量节点 i 除外。

相同的想法适用于变量节点处理（图10.33），其中值 rj 是通过使用连接到变量节点的校验节点发送的所有值来计算的，除了校验节点 j。使用公式 (10.46)

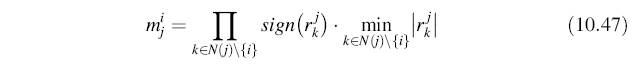


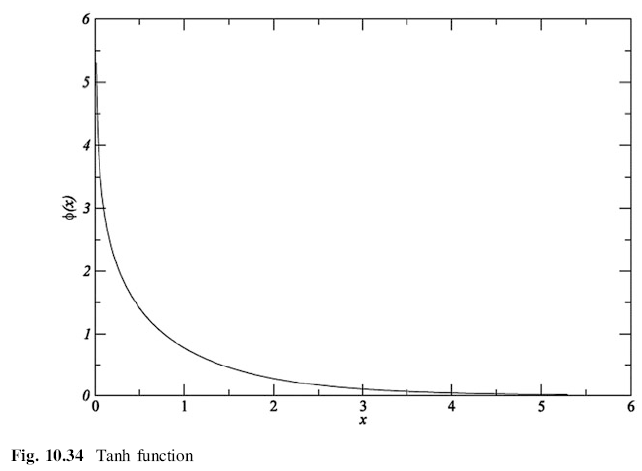
其中w是LLRs的输入。

值r代表估计的码字。在每次迭代结束时，这个字乘以H的转置来检查它是否是一个真正的代码字。如果结果为空，则r为码字，解码完成，否则开始新的迭代。

用于校验节点处理的公式是一个非常复杂的公式，它涉及到函数tanh，如图 10.34 所示。

BP可以用所谓的min-sum解码算法来近似：只要付出小的解码性能下降就可以大大降低计算复杂度。BP和min-sum的主要区别在于校验节点：式（10.44）适用于BP，而对min-sum的校验节点处理由式(10.47)描述。

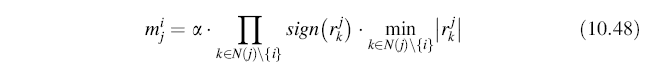




因此，通常作为LUT实现的函数Φ(x)（即tanh）在min-sum解码算法中被消除。Min-sum 可以进一步优化，如下所述。

图10.35a显示了通过sum-product(SPA)计算的值与通过min-sum计算的值之间的比较。 平分线上的点意味着min-sum是sum-product的一个很好的近似值，但事实并非如此；甚至平均值也有不同的斜率。通过引入衰减因子α，近似值可以更好，如图10.35b所示。

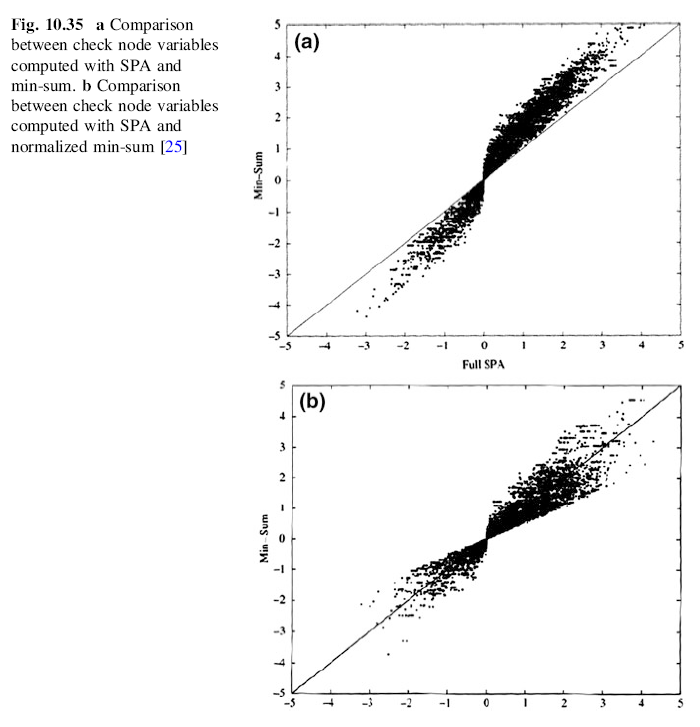
换句话说，式(10.47)可以计算为

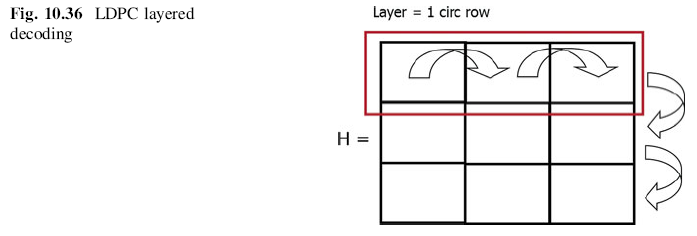


衰减因子可能会在每次迭代中发生变化，因此必须对其进行适当研究。

不管具体的解码算法如何，硬件实现可以通过拆分循环处理（变量和校验节点）并行化，如图 10.36 所示。这种解决方案被称为“分层解码”。

再次以填字游戏类比为例，在最小和的情况下，我们首先处理所有水平词（检查节点），然后才切换到垂直词（变量节点）。在分层的情况下，一旦我们有足够的信息水平词（一个循环行的检查节点），我们立即切换到垂直词（变量节点）。这样，在第二层（第二循环行）的校验节点上的计算具有更清晰的输入（因为它不使用初始变量节点值，而是使用第一层已经计算的值）。确实，分层解码所需的迭代次数比标准min-sum要少得多。



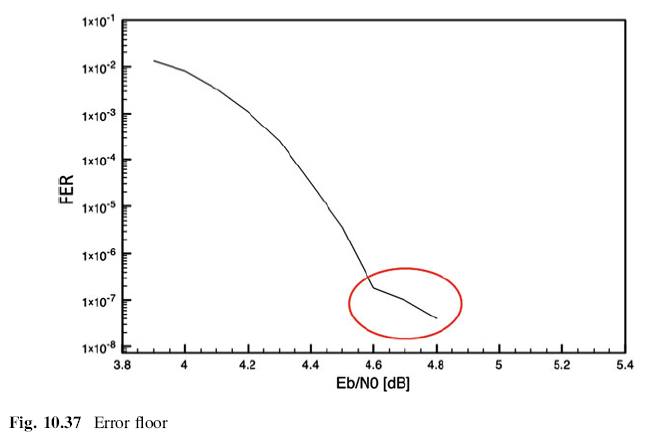


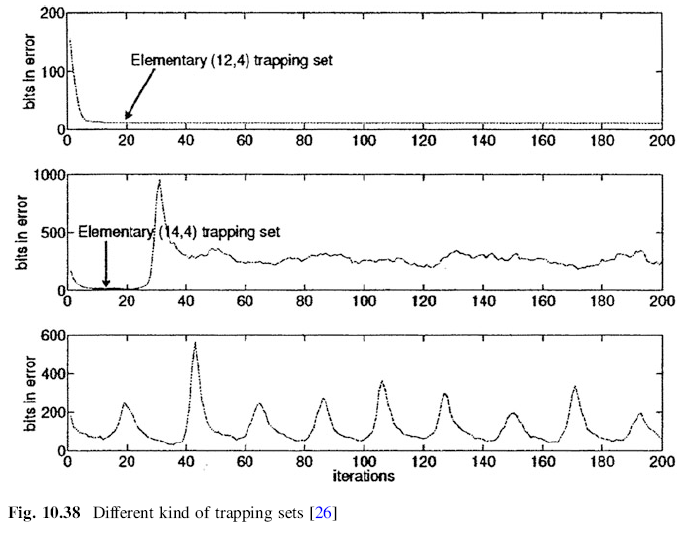
### 应用于NAND闪存存储器的QC-LDPC

对于企业级SSD，目标UBER为10−16（式 10.3）。不幸的是，没有仿真就不可能评估LDPC的性能，因为没有像BCH那样的封闭公式。

除此之外，LDPC解码算法由于其迭代性质，有一个很大的缺点，即错误下限[23,24,27]。

图10.37显示了错误下限是如何表现出来的：它基本上是低BER下斜率的变化。使用 BCH，可以准确预测在哪个BER下，得到的UBER将是10−16；对于LDPC，我们不知道错误下限将出现在哪个BER及其斜率。唯一可以确定的是它会出现。





为什么会出现错误下限仍然是个谜。如今，数学家认为这是由于陷阱集。一旦解码器被困在一个捕获集中，随着解码的进行，一些错误比特对应的变量节点的值会变得越来越大；换句话说，在某些时候，解码器几乎不可能恢复其决定。在没有找到码字的情况下，解码将达到允许的最大迭代次数。

因为有3种不同类型的捕获集（图10.38），解码器的输出可能是：

* 包含少量恒定错误的码字；
* 包含随机数错误的码字；
* 包含周期性错误数量的代码字。

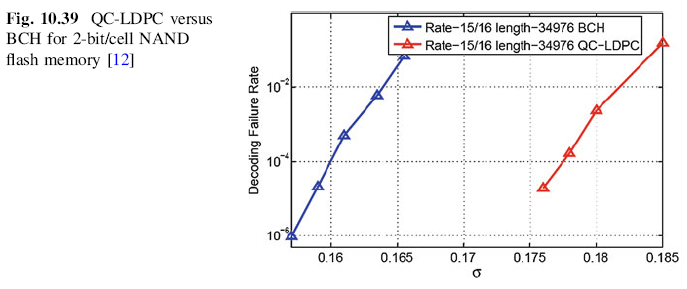
最后一个是非常危险的，因为一个有6个错误的码字解码后可能有200个错误！

回到仿真话题，软件仿真不是达到10-16的UBER的可行解决方案；硬件协同仿真是必须的。单个FPGA每天可以运行几亿个码字，这只有在目标FER在10-6范围内时才可接受。

另一方面，由于错误下限，不可能用简单的直线来近似低于10-6的FER的图形。归根结底，企业应用程序要求仿真不少于1013个代码字。一个FPGA需要100,000天的仿真时间！这就是为什么FPGA网络是解决这个问题的唯一实用方法[24]。

值得强调的是，运行“正确”的仿真很重要。事实上，每个参数变化都需要不同的仿真。 例如，不可能从硬错误下限中提取软错误下限。出于同样的原因，最小min-sum码错误下限不能用于推断归一化min-sum的下限。

图10.39显示了LDPC和BCH在AWGN通道上的比较。NAND VTH 分布被建模为两个对称高斯分布，其平均值分别为VTH = -1和VTH = +1。在该模型中，NAND原始BER由分布的方差σ表示。

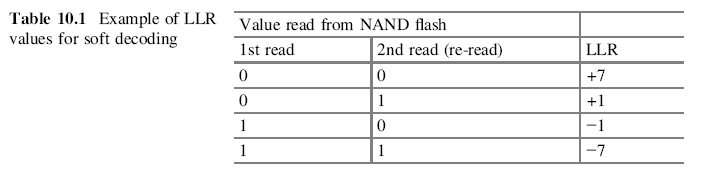


为了了解特定LDPC码的实际性能，基于从芯片中提取的数据进行仿真是至关重要的。

从NAND闪存读取的数据始终为0或1，如第1节所述。10.3.2. 因此，起点始终是硬解码；如果失败，软解码接管，我们需要：

* 重新读取以获得每个位的可靠性信息；
* 将每个位映射到一个LLR值；
* 运行软仿真。

10.3.2节中描述了重读策略：基本上，读取参考电压会发生偏移，执行一个或多个额外的读取操作以了解位在电压分布中的位置。



每个Re-Read操作返回一个0和1的序列，可以耦合到前一个Read的序列，如表10.1所示。

LLR符号表示第一次读的位更可能是0还是1；幅度表示与第一次读取相关的置信水平。 让我们看几个例子：“+1”表示我们读到了0，但我们没有那么自信，而“+7”表示我们读到了0，我们很确定这个位是正确的。

一旦传输信息的每个比特都映射到一个LLR值，这个值就是软解码仿真的输入，用于构建如图10.19所示的曲线。

综上所述，尽管存在与错误下限和软信息相关的挑战，但LDPC可以成功地用于提升 ECC性能，它绝对是3D NAND闪存最有前途的解决方案，尤其是在TLC和QLC存储方面。