# 用于现代 NVM 的高级代数和基于图形的 ECC 方案

在本章中，我们将讨论高级纠错码技术。特别是，我们专注于两种互补策略，非对称代数码和非二进制低密度校验（LDPC）码。这两种技术都受到传统编码理论的启发；然而，在这两种情况下，我们都偏离了经典方法，并开发了专门设计用于利用描述非易失性存储器的固有通道特性的新概念。

我们特别关注现代闪存设备，包括多级和3D闪存技术。Flash是一种非常流行的技术。 它受到的关注导致了许多工艺创新。因此，当前闪存的实现，例如3D闪存，包含大量紧密封装的晶体管。闪存单元遭受各种物理问题，包括干扰/串扰（由于封装设计参数，3D闪存在某些尺寸上比其他尺寸更强）、读写干扰、电荷泄漏等；我们必须寻找新的方法。我们选择两个不同的、相反的攻击点。首先是对经典代数码的改进，它提供已知的、高效的编码和解码算法，适用于具有温和容错要求的廉价、高效的设备。第二个是改进尖端的非二进制 LDPC码，它具有所有已知编码方案中最好的纠错能力，但代价是更复杂的编码和解码电路。此外，新的代数码特别适用于硬读信道，而LDPC码最有利于软读信道。因此，这两种编码方法分别针对了闪存质量/成本折衷曲线的两端。

在代数码的情况下，我们讨论了一组依赖于传统对称码（如BCH码）作为构建块的码结构。最终的结果是专门针对非对称信道定制的一系列代码，例如作为三级单元 (TLC) 闪存数据存储通道，可应用在2D和3D闪存中。我们介绍了这些代码的变体，它可以处理一种非常具体的闪存错误，以及适用于动态的代码阈值方案，这对非易失性存储器十分有效。对于我们技术的一个子集，我们量化了从实际闪存设备测量的数据集提供的改进。

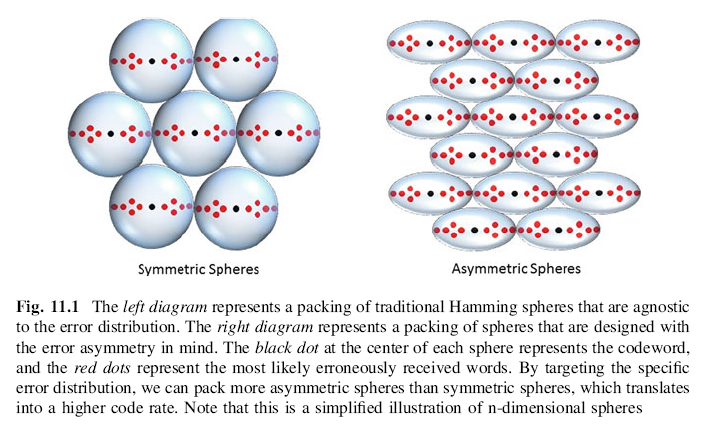
在LDPC码的情况下，我们提出了设计和优化技术，这些技术可以使非二进制LDPC码具有更低的误码率。错误下限是一种随着输入SNR的增加而降低迭代解码的LDPC码的输出错误率改善的效应；这种效应发生在高SNR上，限制了LDPC码在闪存等高可靠性应用中的适用性。为了解决这个问题，我们确定了某些子图对象，称为吸收集，它们出现在LDPC码的Tanner图结构中并导致错误下限。我们针对非二进制LDPC情况对这些对象进行了表征，并提出了一种去除最小吸收集的算法。在这里，生成的代码结构也是为非对称信道量身定制的。该技术的威力针对一系列非二进制LDPC码进行了说明，包括实用的准循环(QC-LDPC) 码。

## 非对称代数ECC

现实生活中的内存通道最有趣的特征之一是它们的不对称性；也就是说，并非此类通道中的所有错误都以相同的概率发生。例如，由多级闪存设备引起的通道，其在一个擦除态和非擦除态之间产生错误的可能性要比在两个非擦除态之间大得多。

传统编码理论在很大程度上不关心这些不对称性。二进制对称（BSC）和二进制擦除（BEC）信道是最常研究的离散信道，而加性高斯白噪声（AWGN）信道是最常用的连续信道。这些通道都没有模拟超出特定通道参数的不对称性。因此，为了将传统编码理论的工具应用到现实生活中，基于最坏情况错误选择对称信道。这种保守的方法允许一个安全的边际。

另一方面，这种方法对于非对称信道也是浪费的，因为大量的代码强度随后被用于纠正罕见的错误。这种不需要的强度会导致低于必要的编码率，从而浪费能量或存储容量。 反之，如果码率保持不变，将代码的力量投入到纠正频繁错误中会更有效，从而改善系统的整体错误概率。图11.1说明了这种不对称编码的概念。



在本节的其余部分，我们将讨论非对称纠错码。我们将前面讨论中的直觉形式化。如前所述，我们特别关注闪存中数据存储的情况。事实上，从生产闪存设备收集的数据集是可用的。 由于代数代码的编码器和解码器易于指定和实现，我们可以直接在实际数据上测试我们提出的代码（而不是使用合成数据进行仿真）。

### 分级位纠错码

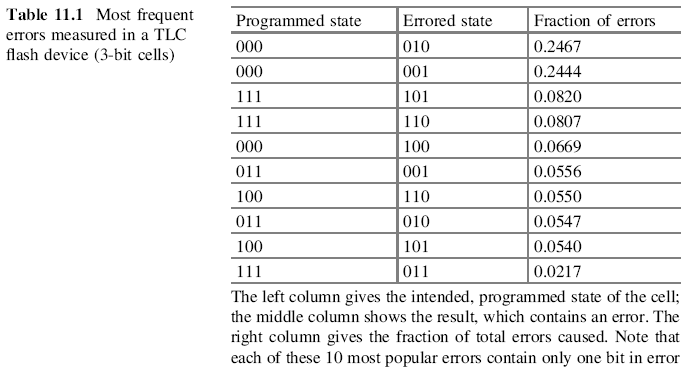
我们首先考虑TLC（三级单元）闪存通道，直到最近，它还是最先进和最密集的闪存技术。尽管给这些设备起了名字，但每个单元都有八个可能的电荷水平，因此代表了三位信息。 闪存设备的组织将这三个位中的每一个放在不同的页面上；页面本身被收集为块，这些块被进一步组织成平面[1]。

这种组织允许我们以两种自然的方式对TLC闪存通道进行建模。首先，分别查看每个单元，我们可以将单元视为8元通道，因为每个单元有八种可能的状态。其次，我们可以单独查看每个位，因为这些位被放置在不同的页面上。在这种情况下，可以将单元建模为三个独立的二进制通道。

在8进制信道的情况下，我们可以应用非二进制码。我们可以使用8进制通道的统计数据来估计一个单元块中预期的错误数量。使用此信息和目标错误率，我们可以选择适当的代码，例如，来自8元BCH代码系列的代码。类似地，如果我们把单元看成三个独立的二进制通道，我们可以根据每个通道的错误概率选择三个二进制码，比如三个二进制BCH码。

然而，事实证明，这两种方法都不合适。[n,k,t]8 BCH码（长度为n和维度k的t纠错8进制码，例如，包含8k个码字）可纠正任何t个8进制错误。例如，状态2和6之间的错误（2→6 错误）可以像1→2错误一样被纠正。但是，我们的通道产生的1→2错误要多得多。 特别是，大多数错误仅存在于每个8元状态的三位二进制表示中的一位。

为了说明这个想法，我们展示了在超过5000个编程/擦除(P/E)操作周期的TLC闪存芯片上发生的最常见错误。比较最常见错误的编程状态和错误状态确实证实了大多数错误仅发生在三位三元组的单个位中（表11.1）。



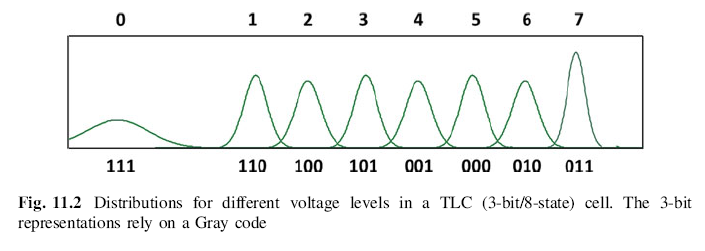


图11.2解释了这样做的原因：3位二进制级别的代表都基于格雷码，因此从一个连续状态到下一个连续状态只改变一个位。我们得出结论，纠正许多2→6错误的能力是代码效率低下的原因。

在二进制情况下，也有类似的问题。从三个二进制通道实际上都在一个单元中运行的事实可以预测，通道不是独立的。在这种情况下，假设独立性低估了不止一个位出错的错误数量。也就是说，诸如e=(1,1,0)和e'=(1,0,1)之类的错误（这里三元组中的每个非零值表示三个位之一中的错误）的代表性不足。事实上，在我们的TLC设备中，我们测量了2位错误的错误比例为0.0314，3 位错误的比例为0.0069。每个页都是独立的。然而，与8进制通道相比，单独的二进制代码方法是更准确的模型。

我们如何设计专门处理此类错误模式的代码？首先，如上表所示，我们可以对通道进行解析，以发现有多少错误通常是单比特错误，有多少是多比特错误。我们之后寻求引入一种码，以精确地纠正这些比率的错误。我们将在下面详细介绍这个概念。

定义1 令 t,v > 0。那么，在 (GF(2)m)n 上的向量e = (e1, e2, …, en) 如果满足以下两个性质，则称为[t;v]-位错误向量：

1. wt(e) = |{i : ei ≠0}| ≤ t，并且

2.对于所有i，wt(ei) ≤ v。

定义 2令 0 < v1 < v2 ≤ m且t1, t2 > 0。向量e = (e1, e2, ..., en) 在 (GF(2)m)n 上是a [t1, t2; v1, v2]-分级误码向量，如果它满足以下性质：

1. wt(e) = |{i : ei ≠ 0}| ≤ t1 + t2，

2.对于所有i，wt(ei) ≤ v2，并且

3. |{i : wt(ei) > v1}| ≤t2。

在前面的定义中，wt() 指的是向量的汉明权重（该向量中非零分量的数量）。基本思想是引入误差向量通常定义的更精细版本。我们不是简单地计算非零分量的数量，而是根据错误的位数对它们进行分类。第一种定义特别适用于所有错误只涉及少量比特的情况。第二个定义更灵活：它使我们能够将错误描述为在几位中涉及的一些错误，在更多位中包含（通常较小）数量的错误。接下来，我们定义能够纠正此类错误模式的代码：

定义 3 令v, t > 0。那么，如果码C能够纠正每个[t;v]-级位错误向量，则码C是[t;v]-位纠错码。

定义 4 令0 < v1 < v2 ≤ m 且 t1, t2 > 0。如果码C能够纠正每个[t1, t2; v1, v2]-分级误码向量，则其为[t1, t2; v1, v2]-分级比特纠错码。

要了解这些定义如何工作（以及它们如何应用于我们的非对称TLC闪存通道），请考虑以下示例。我们存储长度为n的向量，其中每个元素是一个3位向量。为了这个例子，我们取n = 7。假设我们存储向量



一段时间后，我们将存储的数据回读为



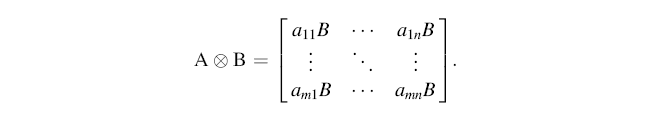
我们可以得出结论，错误向量是



我们可以将此错误向量分类为 [3,1; 1,3]-分级误码向量。总共有3+1=4个错误单元格（4 个不全为0的三元组）。在这四个中，三个只有一位错误，而其余的有三位错误。基于此，我们可以取v1 = 1,v2 = 3,t1 = 3, t2 = 1。

观察这种分类与BCH码使用的更粗略的错误定义有何不同。在8进制BCH码的情况下，我们只记录有 4 个错误，不区分单位错误和多位错误。

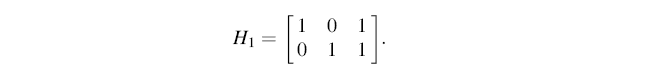
接下来，我们的目标是引入分级位纠错码结构。正如我们将看到的，称为张量积的线性代数运算是这些结构中的关键要素。张量积是对矩阵的运算，定义如下。假设A是 Rm×n中的矩阵，B是Rp×q中的矩阵。然后，张量积 A ⊗ B定义为



换句话说，A ⊗ B 是一个mp×nq块矩阵，其中A的每个元素（标量）乘以矩阵B。这个运算在数学和物理学中有许多重要的性质。在编码理论中，Wolf首先使用它来生成[t;v]位纠错码[2]：

构造 1 令CA为具有由HA = H2 ⊗ H1 给出的校验矩阵的码，其中H1是二进制[m,k1,v]2 码C1的校验矩阵，H2是有校验矩阵为[n,k2,t]d 的码C2，其中d=2m−k1。 那么，CA 是一个[t;v]- 位纠错码。

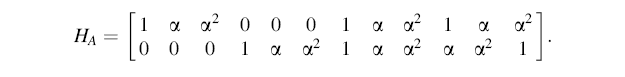
我们可以提供这样一个代码构造的简单示例。对于C1，我们使用汉明码 [3,1,1]2，它具有校验矩阵



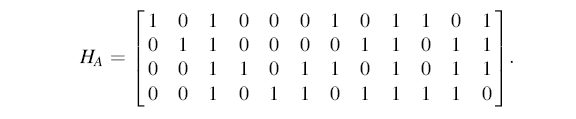
换句话说，我们将使用带有码字{000,111}的二进制3位重复代码作为我们的两个组成代码之一。接下来，我们可以为C2选择不同的代码。请注意，在我们的例子中，此码必须超过 GF(4)，因为根据我们对TLC闪存单元的要求，我们的最终输出必须超过GF(8)。由于我们需要GF(4)上的码，我们让α成为这个有限域上的原始元素。然后，我们可以把 C2 当作 [4,2,1]4 码，这样也纠正了一个错误：



那么，不难看出得到的矩阵是



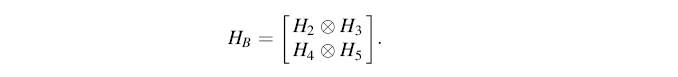
当然，也可以取这个 GF(4) 矩阵的二值图像：



由于H1和H2是具有所需属性的单纠错码的校验矩阵，因此我们期望CA（具有校验矩阵 HA）是[1;1]位纠错码。情况确实如此：我们观察到HA的列都是不同的，因此，可以纠正具有单个位错误的错误向量。 此外，如果我们将CA中的12位长码字分成4组，每组3位，我们将重新获得对码的GF(8)解释。

最近，一种更精细的分级比特纠错码的结构被引入[3]。这种构造也依赖于张量积运算； 但是，结构要复杂一些：

构造 2 令CB为具有由下式给出的校验矩阵的码



在这里，我们有C1和 [m,k,v2]2 二进制码和校验矩阵H1。令r = m-k。我们认为H1使得 H1的前r3行是对于某些r3 < r的[m,m − r3,v1]2代码的校验矩阵。该代码将被称为C3（具有校验矩阵H3）。我们让H5是H1的子矩阵，包括H1的底部 r5 = r-r3行。最后，我们令H2为2r3 元 [n,k2,t1 + t2]d 码C2 (d = 2r3) 的校验矩阵，H4 为2r5 元[n, k4,t2]f 码C4 (f=2r5)的校验矩阵。

那么，CB是长度为n的[t1,t2;v1,v2]分级比特纠错码。

让我们看看解码对于这种类型的码是如何工作的。对于码CB，我们引入了解码器DB，它将向量y=c+e作为输入，其中c是CB中的码字，e a[t1,t2; v1,v2]2 m-位错误向量。这里的输出是误差向量e的估计e'（请注意，我们使用稍微不正常的协议，其中输出是误差估计而不是传输码字的估计。码字估计可以计算为c'=y-e')。然后，解码器DB以下列方式操作。其他每个Di 都是相应码Ci的解码器。

1.形成来自解码器D2(H2 ·(H1'·y1T,..., H1' ·ynT)T)的向量(s10 ,…,sn 0 )。

2.设置误差e\*为(D1′(s10 ),…,D1′(sn0 ))。

3.设置码字y′为y+e\*。

4. 根据 D2(H2 ·(H1' ·y1'T,...,H1' ·yn'T)T) 计算 (s1',…,sn')。

5. 将 (s1″,…,sn″) 设为 D3(H3 ·(H1″·y1′T,...,H1″·yn′T)T)。

6. 将 I 设为 {i : (si′,si″) ≠ (0,0)}。

7. 如果i在I中，则y″满足yi″= yi，如果i不在I中，则yi″= yi′。

8. 将(s11 ,…,sn 1 )设为D3(H3 ·(H1″·y1″T, …, H1″·yn″T)T)。

9. e=(e1,…,en) 如果i不在I中，则ei = ei\*，否则 ei =D1(si0,si1)。

这里的基本思想是首先用更少的错误位纠正错误。当然，有些错误的错误位太多，所以会被错误纠正（但最多只能到权重v1+v2）。接下来，我们检测哪些错误是误纠正的错误，并同时纠正它们。

我们还注意到，解码过程中的所有重要操作都是解码函数D1、D2、D3 的使用。此外，这些操作中的每一个最多执行两次。因此，整体解码复杂度是最差（就复杂度而言）构成代码的复杂度乘以一个小的常数因子。所以，例如，如果我们使用BCH码作为我们的组成码，我们的整体解码算法的复杂度大约是最大组成 BCH 码的两倍。

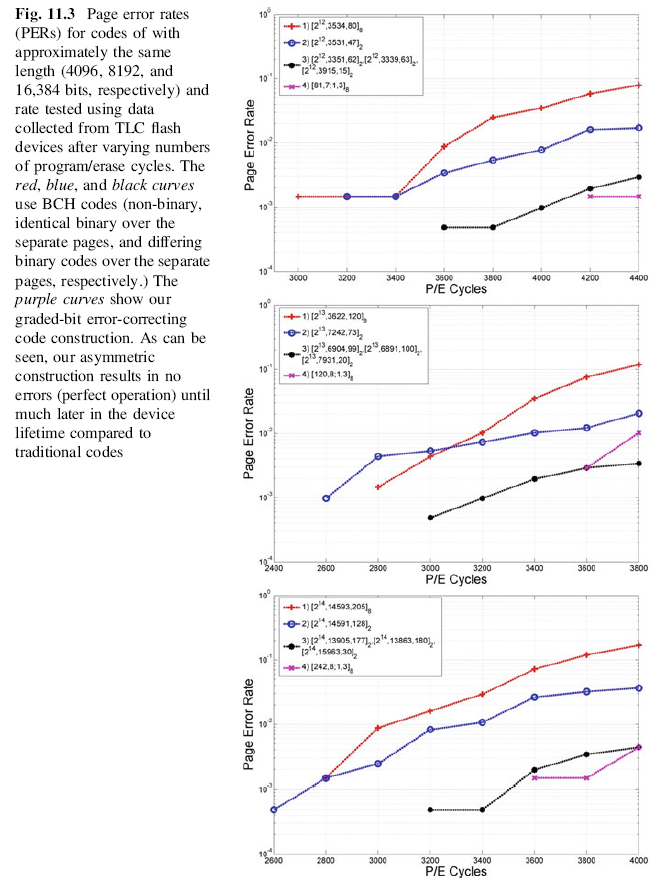
如前所述，我们可以在从TLC闪存设备收集的实际数据上测试建议的分级位纠错码。通过以下方式收集数据：将随机数据模式写入设备，填充每个块。此过程重复5000个编程/擦除 (P/E)周期；每100个周期，数据被回读查看错误[1]。

与具有相同码率和长度的其他BCH代码进行了比较。在图11.3的三个图中，代码的长度分别为4096、8192和16,384。例如，下图中的紫色曲线表示带有参数的分级比特纠错码 [t1,t2; v1,v2] = [242,8; 1,3]。红色曲线代表8元BCH码。蓝色曲线代表（相同的）二进制BCH代码，用于分别保护单元中3位的每一位。

黑色曲线代表3个二进制BCH码（具有不同的参数），分别选择用于优化每个位的错误率。对于我们的分级比特纠错码结构，我们还选择了BCH码作为我们的底层组成码，以便通过堆叠 BCH 码的校验矩阵的张量积来产生最终的校验矩阵HB。

可以看出，使用分级位纠错码，直到设备寿命后期（超过3000个P/E周期），测量数据才出现错误。在此之后，这些代码的性能与单独的二进制BCH代码一样或更好。同时，8元对称方案的性能最差。

我们必须指出，这不是使用为处理特定错误模式而定制的非对称代码所能完成的限制。 我们还观察到了其他几种类型的错误[4]。在TLC闪存的情况下，对错误模式的仔细研究表明，单元可以大致分为可靠和不可靠单元，其中不可靠单元更容易出错。在我们检查的数据集的情况下，我们注意到，在测试的5000个P/E周期中，一组特定的大约65,000个单元（约占单元总数的0.05%）导致超过50个错误。换句话说，大约10-4个单元占了超过10%的错误。



这些不可靠的单元行为是什么？我们观察到，当它们被编程到更高的电压电平时，这些单元会产生这些错误。TLC单元有8个可能的电压电平；当不可靠的单元被编程到4-7级（但不是0-3级）时，经常发生错误。因此，希望引入一种码，该码具有与先前讨论的分级比特纠错码相同的特征，同时还避免将不可靠的单元编程到危险的高电平。

幸运的是，事实证明这是可能的。我们将自己限制在TLC闪存的特定情况下，这意味着我们希望在GF(8)中创建一个码，或者等效地，创建一个长度为3n的二进制代码。当然，也可以为更一般的情况创建类似的结构。

在我们继续之前，让我们介绍一种辅助代码结构，称为“stuck-at”纠错码。首先，定义运算 ◦: GF(2)m × GF(3)m 到 GF(2)m 使得b = a ◦ s，其中bi = si 如果si < 2否则bi = ai。 Pj 是定义为Pj中所有向量s = (s1, ..., sm) 的集合，使得 |{i : si < 2}| ≤ j。然后，stuck-at纠错码定义如下：

定义 5 对于正整数m,k,t,j，一个[m,k,t,j]2 二进制码C是GF(2)上长度为m、维度为k的线性码，具有编码和解码映射EC和DC

1. 对于Pj中的所有s和任何信息h，EC(h,s) ◦ s = EC(h,s)，并且

2. 对于GF(2)m中的任何错误向量e，且wt(e) ≤ t，则DC(EC(h,s) + e) = h。

定义5背后的想法是，即使特定的单元子集被卡住（子集的大小最多为 j），卡住的纠错码C仍然可以从错误中恢复。正如我们将看到的，我们可以使用这些类型的码作为我们结构的构建块；我们调整stuck-at错误的行为来限制我们的目标单元的电平。

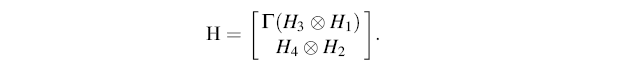
接下来，我们介绍一下我们的目标代码：

定义 6 令 n,k,t1,t2,j为正整数，其中j,t1,t2 < n。那么，一个[3n,k,t1,t2,j] 动态比特纠错码C是长度为3n、维度k的二进制线性码，能够纠正任何[t1,t2]比特错误向量。还有一个额外的限制：如果我们在C中写一个码字为c = (c1,c2, ..., cn)，其中每个ci是GF(8)中的一个元素，那么，给定一个大小最多为j的集合I，对于I中的所有i和C中的所有码字c，ci ≤ 3。

定义6与我们之前讨论的定义4相匹配，但添加了仅将特定单元子集编程为低级别的要求。因此，我们引入了基于构造2的构造并添加了相应的约束。让我们也使用从GF(4)中的元素到长度为2的二进制向量的简单映射（同样，α是GF(4)中的原始元素）：



构造 3 令H1 = (α α2 α3) 且H2 = (1 1 1)，其中H1 是GF(4)1×3 中的矩阵，H2是 GF(2)1×3 中的矩阵。令H3为[n,k3,t1+t2]4 码C3的校验矩阵。此外，令H4为[n,k4,t2,j]2 纠错码（如定义5中介绍的）的校验矩阵。然后，长度为3n的[3n,2k3+k4,t1,t2,j]2 动态比特纠错码的校验矩阵由下式给出

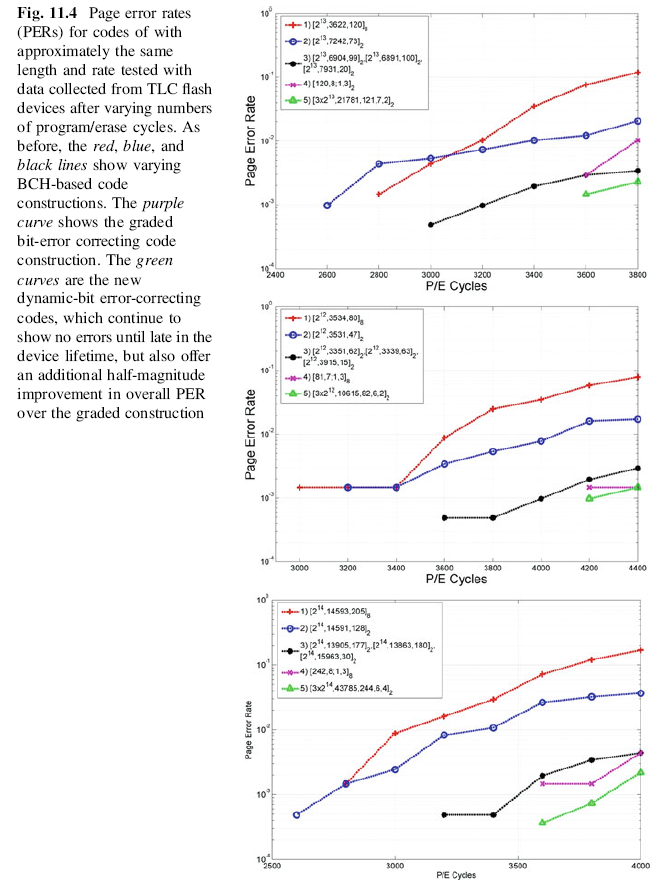


这里的基本思想是通过强制使用固定纠错码结构来稍微修改我们之前的分级位纠错结构（结构2）。而不是使用它来专门纠正固定错误，我们几乎反着做。该构造允许将消息映射到几个可能的码字之一，以处理卡住行为。我们通过选择我们处于较低级别的不可靠码字来利用这一点。

对于我们的动态比特纠错码的情况，我们也可以进行仿真以显示此类码的优势。我们与分级比特纠错码（缺乏避免不可靠单元中的高电平的特定约束）和其他前面提到的码（包括各种BCH代码）进行比较。

在图11.4中，分别显示了长度为4096,8192和16,384的码的页面错误率 (PER)。和以前一样，比较的代码的长度和速率大致相等。相同的代码结构如前所示；绿色曲线显示了新的动态错误码结构。在顶部的两个图中，有2个不可靠的单元被强制降低级别，而在底部的图中使用了4个不可靠的单元。紫色曲线显示了分级位纠错码。其他曲线与前几节中用于比较的代码类型相同。

注意到动态位纠错码具有最佳的整体PER，而在设备的生命周期中很长一段时间内仍不会出现任何错误。这些码的额外不对称性（与分级位纠错码相比）使我们的PER性能额外提高了半个数量级。因此，我们拥有两全其美的优势：具有非常好的PER的码，直到闪存设备的使用寿命后期才允许出现错误。我们成功地利用了不对称性来生成码，这些码比传统的对称码有了显着的改进。



### 动态阈值

Flash通道的另一个特别有趣的特性是它是随时间变化的。数据写入和数据访问操作之间的时间间隔越长，读取错误的概率就越高。此属性是由于作用于闪存晶体管的物理效应导致的。 例如，随着时间的推移，被困在闪存单元浮栅上的电子会泄漏出来，从这些栅中逸出。 因此，由这种效应引起的误差本质上是不对称的。

除了这些不对称性之外，传统编码技术也没有考虑或利用信道的时变特性。回想一下，闪存器件的工作方式如下：测量浮栅上的电荷量并将其与一组阈值进行比较。该比较的结果决定了从设备中读出的离散值。使用的阈值传统上是固定的和永久的，忽略了通道随时间的退化。尽管这些固定阈值可能适用于特定操作周期的通道，但它们通常被证明对于不同的保留周期效率低下。

该问题的一种解决方案是引入动态阈值，该阈值可以随时间而改变。尽管有很多方法可以完成这项任务，但有一个特别简单的方法。我们以这样一种方式设置阈值，即读取时单元块中的值分布与写入时相同[5, 6]。换句话说，我们使用这种值的分布作为辅助信息。

让我们形式化这个想法。假设我们有一个由n个单元组成的块x = (x1,x2,…, xn)，每个单元可以取任意q值 (0,1,…, q−1)。在TLC闪存中，正如我们之前讨论的那样，q=8。现在，经过一段时间，写入的值已变为实际值v = (v1,v2,…, vn)。我们有阈值t= (t1,…, tq−1)，我们用它来以下列方式读取v：输出y = t(v)由下式给出



我们将 t0 设为负无穷大，将 tq 设为正无穷大。

现在，让我们用k = (k0,…,kq−1) 表示x中值的分布。也就是说，ka = |{i | xi = a, 1 ≤ i ≤ n}|。 因此，例如，x =(1,0,0,3,1,1,1,2) 有k(x) = (2,4,1,1)，因为x有2个值0，4个值为1，以此类推。

然后，我们可以通过以下方式定义动态阈值

定义 7 一个阈值向量t是一个动态阈值，如果



例如，假设上面的向量x是写的，实际电荷值由下式给出



然后，如果我们使用固定阈值t1 = (0.5,1.5,2.5)，我们将读取输出



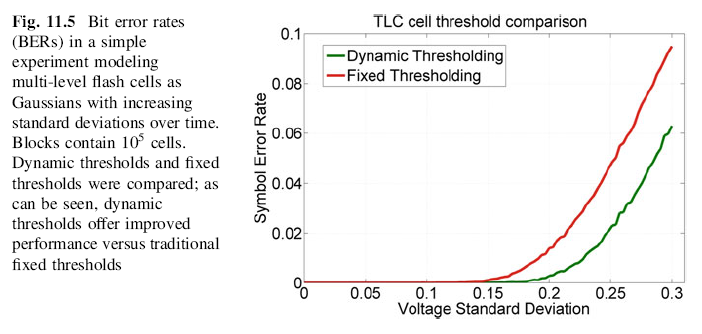
在第三和第四位置有错误。但是t1不是动态阈值：y1的分布是k(y1)=(1,5,2,0)，不等于k(x)=(2,4,1,1)。让我们改为选择一个动态阈值td = (0.7,2,2.25)。然后，我们正确得到



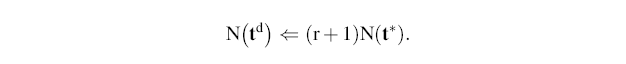
当然，动态阈值并不能保证读取序列没有错误。然而，它们降低了错误率，因为要产生错误，两个分量（具有不同的初始值）必须相对于彼此交换它们的值。例如，如果我们有xi < xj，我们必须有vi > vj 来产生错误。与简单地要求xi < txi 相比，此事件发生的概率较低，这对于固定阈值情况下的错误是足够的。

图11.5说明了这种说法，我们通过将闪存单元建模为标准偏差随时间增加的高斯模型，模拟了通道随时间推移的退化。然后，我们通过写入随机值并使用动态阈值与固定阈值回读错误来模拟105个单元的块。随着对闪存通道建模的高斯标准偏差的增加，动态阈值方案产生的错误概率增长要慢得多。

这种类型的模拟为动态阈值优于固定阈值的建议提供了实验支持。但是，我们也添加了理论比较。假设N(x,y) 是向量x和y之间的汉明距离。如果y是通过读取x生成的，即y = t(v(x)) 是使用阈值t从v读取的值（本身由写入的值x形成），那么，我们写N(x,y)为N(t)。我们也说t\*是最佳阈值，因为对于某些固定的x,y，t\* = mint N(x,y)。



我们还假设最大可能的误差幅度由r给出，对于{0,…, q−1}中的一些r。对于flash，这是一个合理的假设：我们预计大多数错误的幅度很小，最多可能为1。这样，我们可以说任何动态阈值td 都非常接近最佳阈值t\*：



换句话说，任何动态阈值至多是最优阈值的常数因子（取决于最大误差幅度）。当然，这个最优阈值需要x本身的认知来计算。该认知在读时不可用：对x的可靠估计是读操作的目标。换句话说，动态阈值提供了一个非常接近无法获得的最优值的实用解决方案。

到目前为止，我们还没有讨论如何生成一组阈值。当然，这是一个重要的实际问题。有两种可能的方法（以及这两种方法的多种组合）[6]。第一种是使用块中值的分布作为辅助信息。

然后将此辅助信息存储在其他地方。例如，我们可以将这些值存储在非常健壮、高度可靠的单元格中，由强大的代码保护，然后可以使用固定阈值以低错误风险读取。

另一种方法是将数据存储在恒定权重码字中。这些码字具有固定的值分布。由于分布是固定的，它可以从生产中硬编码到系统中，完全绕过了传递辅助信息的需要。当然，这里的权衡是恒定权重代码消除了某些码字的使用，从而产生了可能的更小的总体速率。

无论采用哪种方法，我们都必须使用纠错码进一步保护我们的系统。动态阈值本身不足以将系统的错误率降低到目标率。这给我们留下了选择什么码的问题。当然，我们可以使用现有的现成码，例如BCH码。然而，这些方案忽略了动态阈值产生不对称误差这一事实，就像忽略3位TLC错误向量中的不对称一样（导致我们改进了基于张量积的构造）。向量不能出现，因为这会改变读取码字的分布，而动态阈值的定义是不可能的。但是，传统的代码不能利用这个想法。

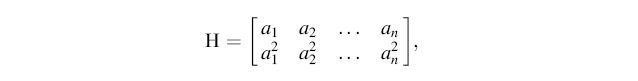
相反，我们可以提出专门针对动态阈值运行的专用非对称代码。

定义 8 让向量x存储在具有动态阈值的系统中。如果e最多有t个非零分量并且每个分量的幅度最多为v，则x在动态阈值下可能遇到的错误e称为[t,v]-DT错误。能够纠正任何[t,v]-DT 错误称为[t,v]-动态阈值纠错(DTEC)码。

请注意，并非所有[t,v]-错误向量都是[t,v]-DT错误向量。例如(1,0,0,0) 是长度为1的[1,1]-误差向量，但不是DT错误向量，因为使用动态阈值，误差需要至少两个位置非零才能排序。保持x和x + e之间的值分布。换句话说，DT错误向量比一般错误向量少。虽然传统的纠错码可以纠正DT错误，但它也纠正了DT错误不会出现的错误向量，这会通过牺牲未使用的纠错强度的速率来降低代码的整体性能。

我们引入了一种非对称结构，专门纠正2个任意大小的DT错误，从而提供了[2,q-1]-DTEC码结构的示例：

构造 4 令C为域Fq上的[n,n-2]q 线性块码（长度为n，维度为n - 2），校验矩阵由下式给出



其中S = {a1, a2, ..., an}是Fq的不同元素的子集。那么，如果S是一个Sidon集，则C是一个[2,q−1]-DTEC码（一个具有以下性质的集合：对于S中的任何四个不同元素a,b,c,d，a + b≠c + d ）。

注意到这样的码可以纠正动态阈值中的任何2个错误，而一般的2个错误纠正代码需要更大的冗余。这是定制的非对称纠错码的优势。可以修改先前的构造以产生小于q-1的有限幅度r的其他值的代码。

有了这个，我们可以看到另一个如何利用不对称性来引入更优的代数纠错码的例子。

## 非二进制 LDPC 码

接下来，我们将重点从代数代码转移到基于图的代码。基于图形的代码比代数代码有一个很好的优势，因为它可以使用软信息进行解码。换句话说，基于图形的代码解码器可以将小数（而不是整数）值作为输入。然而，代数代码缺乏这种能力。软信息的使用对于闪存等存储设备尤为重要，因为我们可以对数据执行多次读取，以检索更准确的解码器输入。 因此，软信息产生了出色的纠错性能。我们将在本章后面提供有关此概念的更多详细信息。

我们对最重要的一类基于图的码特别感兴趣，即非二进制低密度奇偶校验(NB-LDPC)码。 LDPC码于1960年代首次在Gallager的开创性博士论文中引入，并在90年代重新发现。二进制LDPC码已被广泛研究并已在许多应用中找到用途。

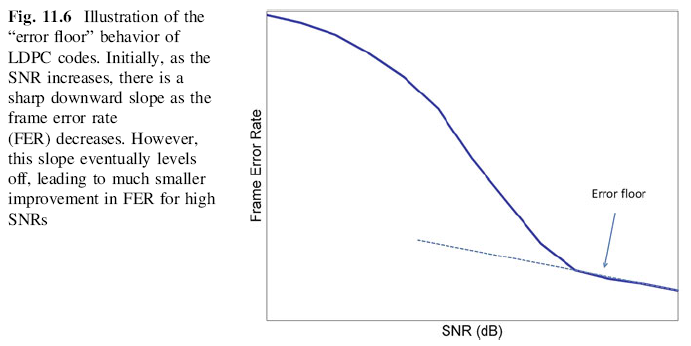
然而，非二进制LDPC码的理解仍然不太好。Davey和MacKay[7]的早期工作表明，与二进制LDPC码相比，非二进制LDPC码具有更好的性能。这种性能随着字段大小参数的增加而增加。然而，这种性能提升是以解码器复杂性为代价的。用于非二进制码的LDPC置信传播解码器的初始植入对于q的字段大小具有O(q2)的复杂度。然而，这种复杂性可以通过基于FFT的解码器实现降低到更易于管理的O(q log q)。已经提出了用于低复杂度解码的其他技术，包括基于线性规划的技术。

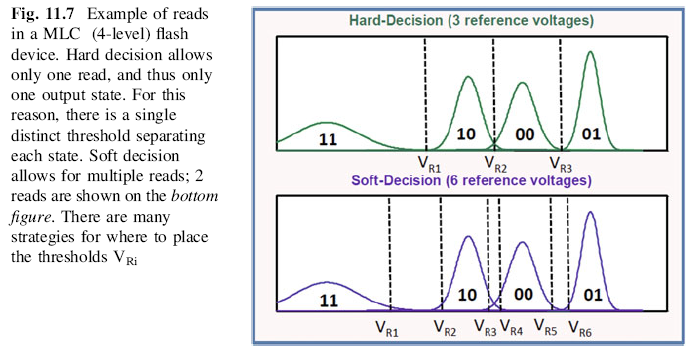
除了改进的解码器复杂性之外，在过去十年中还提出了大量非二进制LDPC码的结构。 这种结构所采用的方法差异很大。例如，构造包括准循环码（一些基于几何方法）、基于原型的码、量子LDPC码和许多其他码[8-10]。在非二进制LDPC研究领域中这种改进工作的激增表明这种代码正在接近普遍的实际应用。一般来说，增加LDPC码的码长会提高其性能； 但是，收益递减。例如，将代码长度从1000位加倍到2000位通常比将代码长度从100,000位加倍到200,000位对性能的积极影响要大得多。

然而，在非二进制LDPC码的普遍应用成为现实之前，还有一个额外的障碍需要处理。 这就是所谓的LDPC“错误下限”。该术语反映了LDPC码的误码率或误帧率与SNR的关系。 最初，随着SNR的增加，BER/FER相应地显着提高；这就是“瀑布机制”。然而，在某个点之后，这些曲线变得越来越平坦，进入了错误下限区域。这个错误下限是一个特别重要的问题，因为LDPC码的许多应用，例如数据存储设备，都在非常高的SNR下运行。例如，对于Flash存储器，所需的输出FER通常超过10−15；对于许多LDPC码，这一点正好位于错误下限区域。图11.6显示了错误下限的图示。

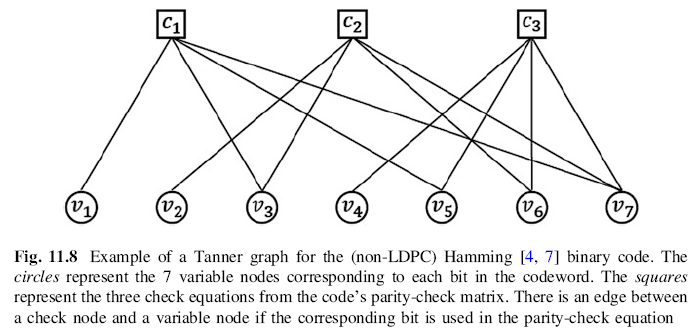
是什么导致了错误下限行为？这是在二进制LDPC码[11, 12]的背景下密切研究的一个重要问题。因此，我们将注意力集中在更高性能的非二进制LDPC 码上。我们首先解释实际 LDPC解码器的操作。例如，在存储设备的情况下，允许对底层设备进行少量探测。如果只允许这样的探测，我们将系统称为硬决策。如果读取不止一次，则系统是软决策，如图11.7底部所示。但是，由于延迟问题，只允许进行少量探测。请注意，一个重要的问题是设置参考阈值（单读情况下为VR1、VR2、VR3，两读情况下为 VR1、…、VR6）[13]中提出了一种基于互信息优化的方法。

由于读取次数较少，存储设备的连续通道已经转变为离散通道。类似地，在数字系统中，信息被量化为有限精度变量。结果，在实际系统中，解码器的行为最终类似于在更简单的通道上运行的解码器，例如离散的无记忆通道。对此类信道上的LDPC码进行了很好的研究。





现有研究的大部分检查二进制码的错误层。错误层实际上与LDPC码的图形结构中的某些对象密切相关。这个图结构就是Tanner图；LDPC码的Tanner图是一个二分图，其中两类节点是变量节点（对应于LDPC码字向量中的分量）和校验节点（对应于校验方程）。如果相应的组件包含在相应的校验方程中，则校验节点和变量节点之间有一条边。汉明[7,4]二进制码的Tanner图示例如下所示（图11.8）。当然，这个码不是低密度的；然而，简单的校验矩阵有助于说明 Tanner 图定义背后的思想。



由于置信传播解码器在此图结构上运行，因此某些节点配置导致解码问题也就不足为奇了。捕获集和吸收集是子图对象的示例，当在特定代码的 Tanner 图中找到它们时，已知它们会导致错误。在二进制LDPC码的情况下，这些对象已被广泛研究。 许多论文提出了LDPC码的设计算法，以避免捕获和吸收集合，从而消除错误下限行为[14, 15]。

然而，这个问题在非二进制LDPC情况下更具挑战性。在本章的这一部分，我们将探讨如何识别、枚举和移除非二进制LDPC码的吸收集。我们首先总结传统二进制LDPC案例中的吸收集。

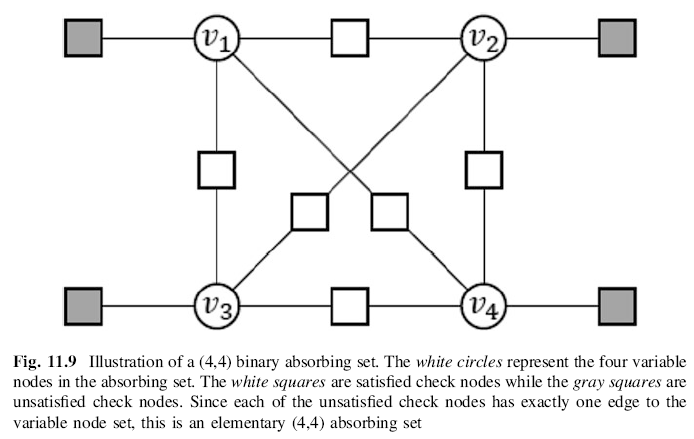
### 二元捕获/吸收集

我们从二进制LDPC码的Tanner图的子图开始。子图包含变量节点集V且|V| =a。V中的变量节点设置为1，而所有其他变量节点设置为0。连接到V中的顶点的校验节点分为集合E和O，其中E包含具有偶数条边到顶点的校验节点V，而O具有奇数个此类边的校验节点。当然，在这个配置中，E 包含满意的校验节点，而O包含不满意的校验节点。现在我们可以引入捕获和吸收集：

定义 9 如果|O|=b，V是一个(a,b)陷阱集。

定义 10 如果|O|=b，且V中每个变量节点在E中的相邻节点(严格)比O中多，那么V就是一个(a,b)吸收集。

定义 11 一个基本的吸收集/陷阱集，其附加条件是每个相邻的满足校验节点都有两条连接到集合的边，而每个相邻的不满足校验节点都有一条连接到集合的边。



我们在图11.9中展示了这样一个集合。

我们看到显示的配置产生了一个(4,4)吸收集。我们有4个变量节点连接到4个未满足的检查节点。不满意的校验节点是灰色方块，而满意的校验节点是白色的。另外请注意，这是一个基本的(4,4)吸收集，因为每个不满足的校验节点都有一条边将其连接到4个变量节点，而每个满足的校验节点都恰好有两条边连接到变量节点。

注意吸收集的基本思想：它是一个变量节点的配置，其中多数逻辑位翻转解码器会出错（这里，我们假设发送了全零码字）但无法从这个错误恢复。这种行为的发生正是因为大多数相邻检查节点都得到满足。

我们还需要一些额外的图论概念。我们可以在无向图的所有循环的集合上定义一个向量空间。对于这样一个G = (V,E) 的图G，以对称集差为加法运算，恒等函数为负运算，空集为附加标识元素。则图G的循环空间是2E 的子空间，以G的循环为元素。现在我们应用线性代数的基本原理：

定义 12 如果G=(V,E)中的一组循环F构成循环空间的基础，则它是 G 的循环跨度。 周期跨度中的周期称为基本周期。

我们还可以引入一个相关的图结构，称为变量节点（VN）图。该图是基于基本吸收集的二分图定义的。变量节点图只包含变量节点；如果这些节点共享一个二度校验节点作为邻居，则这些节点由一条边连接。

### 非二元吸收集

我们现在准备解决非二进制吸收集的问题。由于我们在非二元状态下工作，因此代码的 Tanner 图在连接变量和检查节点的边上放置了权重。该权重等于LDPC码的非二进制校验矩阵中对应的非零值。这为图结构增加了一个额外的方面：有一个拓扑结构（就像二进制码的情况一样），而且我们现在有一个权重结构。 因此，非二元吸收组也必须满足权重条件。

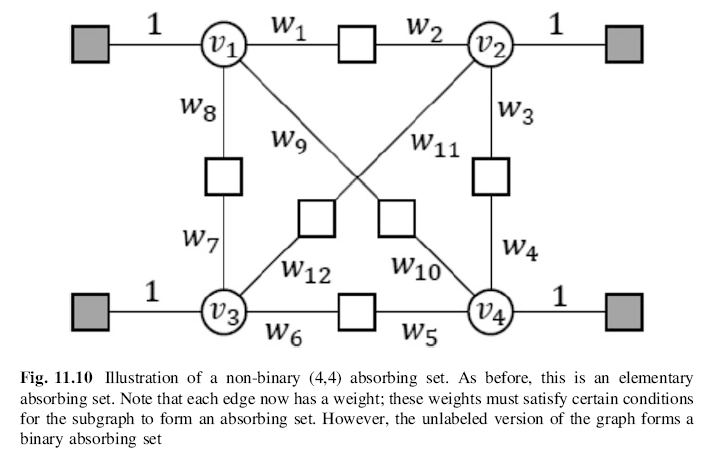
和以前一样，我们寻求一种配置，其中每个变量节点与不满意的节点相比，具有更多满意的相邻检查节点；然而，对于满意/不满意的部分，我们需要权重之间的某些关系。我们在图11.10中展示了这种吸收集的一个例子。

请注意，这里的边权重被视为来自码的有限域GF(q)的非零元素。该吸收集与前面的二元吸收集示例具有相同的拓扑结构。

然而，在一般情况下，为了满足二阶校验节点，我们需要以下关系来保持GF(q)。请注意，权重已在前面的图表中被标记。

v1w1=v2w2, v2w3=v4w4, v4w5=v3w6,

v3w7=v1w8, v2w11=v3w12, v1w9=v4w10.



这些方程中的每一个都直接来自Tanner图的定义。例如，v1和v2之间的校验节点只有（回想一下，除了v1，…，v4之外的所有变量节点都设置为0，而v1，… ,v4设置为1) 在相应的校验方程为0是才会满足：v1w1 + v2w2 = 0。

如果我们的域大小是2的幂，因此q=2p，我们可以消除这些方程中的变量节点，以便专门为权重编写一系列条件：

w1 w7 w11= w2 w8 w12, w3 w5 w12= w4 w6 w11, w2 w4 w9= w1 w3 w10

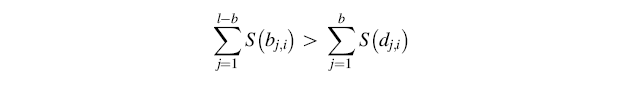
和以前一样，方程被GF(2p)

基本思想可以写成一般形式来定义非二元吸收集：

定义 13 如果存在 (l − b) x 秩为rB的子矩阵B，其元素 bj,i 满足 1 ≤ j ≤ l − b, 1 ≤ i ≤ a 在矩阵 A 中满足条件，则集合V是GF(q)上的(a,b)吸收集合：

1.如果N(B)是B和di的零空间，1≤i≤b是D的第i行，其中D是从A中排除矩阵B得到的，则存在x = [x1 x2 … xa]T 在N(B)中，使得对于{1,…,a}中的所有i，xi 不为零，并且不存在 di x = 0的i。

2. 如果D包含元素dj,i 对于 1 ≤ j ≤ b, 1 ≤ i ≤ a, 那么对于{1,2,…a}中的所有i, 有

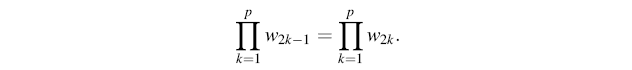


这里函数S是一个指示函数，对于非零x，S(x) = 1，对于x = 0，S(x) = 0。

我们观察到，对非二元情况的类似适应类型也可能用于捕获集。

我们可以通过将与二元吸收集相同的条件添加到基本二元吸收集的情况来定义非二元基本吸收集。在这个基本的吸收集案例中，我们可以进一步操纵上述条件，使其具有以下形式，类似于我们的示例中所示的形式。令Cp是一个循环，在由 (a,b) 非二元吸收集引起的图中包含p个不同的变量节点和p个不同的校验节点。令Cp = c1-v1-c2-v2-...-cp-vp-c1。 权重w2i-1是连接ci 和vi的边上的标签。类似地，w2i 是连接vi和ci+1的边上的标签。然后，我们有以下内容。

引理 1 如果域大小参数q = 2p，那么，每个周期Cp满足如下关系：



在基本非二元吸收集的情况下，我们现在对定义进行简单分解：（未加权的）拓扑结构必须是二元吸收集，此外，权重必须满足引理1中给出的方程。

现在我们已经建立了定义并确定了非二进制吸收集是什么，我们准备研究如何提高非二进制代码的性能。

### 性能分析和影响

我们首先确定可以满足权重条件的频率。这是一个重要的问题，因为如果不满足条件，我们就没有吸收集。这个概念在以下定理中描述。

定理 1 我们有一个(a,b)未标记（二进制）基本吸收集，其中有e个满足的校验节点。然后，

1. (q − 1)a−e−1的一部分边权重分配（接管 GF(q)）产生非二元基本吸收集。

2. e(q − 1)a−e−1(q − 2) 的一部分边缘权重分配（再次接管 GF(q)）产生 (a,b + 1) 个非二进制陷阱集。

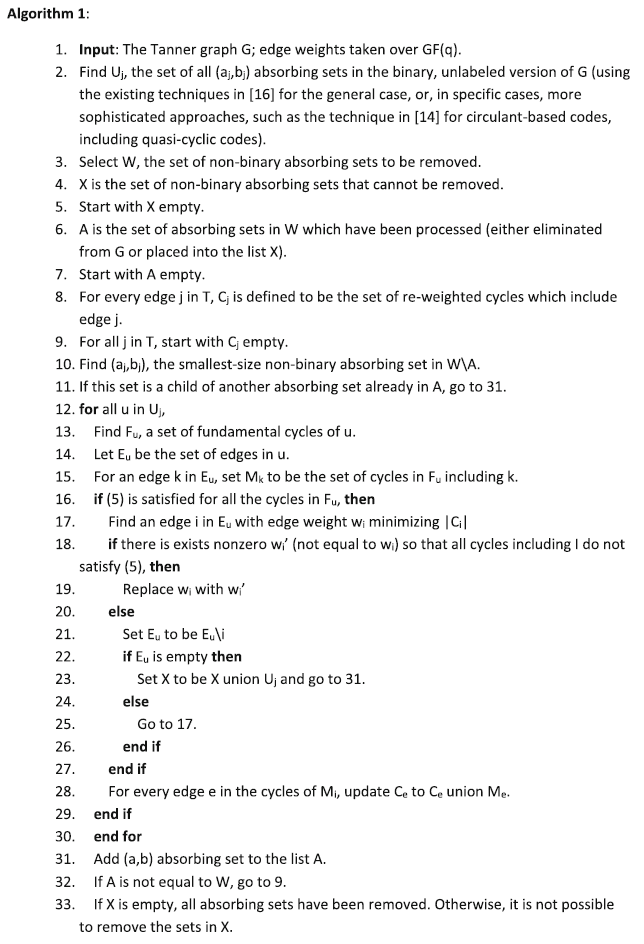
定理的证明依赖于基于已经介绍的图论思想的简单计数参数。

假设代码及其权重是随机生成的，该定理意味着与非二进制吸收集相比，存在更大数量的非二进制陷阱集（乘以e(q - 2)的因子）。然而，在实践中，模拟结果表明错误配置文件不涉及来自陷阱集的任何错误。另一方面，错误配置文件确实显示了由吸收集导致的错误。什么解释了这种行为？这个想法是解码算法中使用的量化导致这些置信传播以类似于多数逻辑位翻转解码器的方式起作用。这样的解码器不会与更一般的陷阱集错误作斗争，但是，正如我们从吸收集的定义中看到的那样，这些解码器在面对吸收集时会产生错误。

出于这个原因，更希望找到一种方法来从非二进制LDPC码的Tanner图中去除或减少吸收集。 首先，我们注意到我们必须针对某些吸收集参数而不是其他参数。在具有高SNR的错误下限机制中，错误通常只包括少量的变量节点。因此，我们着眼于小的吸收集。事实上，LDPC解码器的性能将由最小的吸收集支配，这也是典型的基本吸收集。因此，目标变成最大化最小吸收集的大小。

我们现在准备介绍一种算法，该算法可以从非二进制LDPC码Tanner图中消除有问题的吸收集。如上所述，关键思想是以这样一种方式操纵边权重，使得所涉及的子图不再是吸收集。下面给出算法。我们使用一个附加术语：如果A是B的子图，则吸收集A是吸收集B的子集。我们称B为A的父集。

算法1中的基本概念如下。首先，我们选择我们希望消除的基本吸收集。这些集合必须根据代码参数（例如，列重、周长等）来确定。接下来，在这组集合中，我们检查最小的吸收集合。我们在未标记的（即二进制版本的）Tanner图中寻找该集合的二进制版本。如果这些吸收集的基本循环满足引理1中的公式，我们将其中一条边的权重修改为GF(q)中的某个其他非零元素。我们以这样一种方式做出这个选择，即之前移除的吸收集不会被带回来。一旦当前吸收组被移除，该过程将继续。



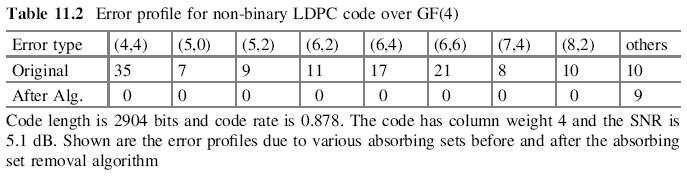
让我们查看特定代码选择的错误概况（由于各种吸收集而给出错误）。我们展示了先前算法对这些错误的影响（表 11.2）。

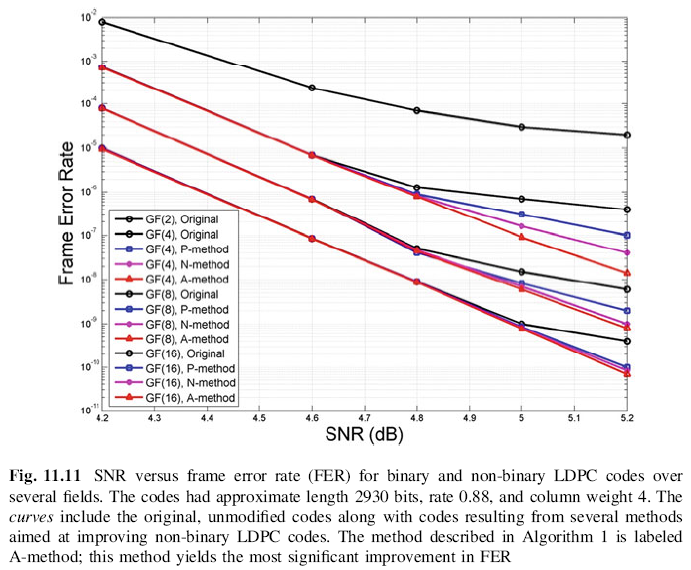
每种错误类型指的是导致错误的 (a,b) 吸收集。该算法删除了所有这些吸收集（最多为 (8,2) 集），从而大大减少了错误的数量。这里，该代码是基于GF(4) 的非二进制LDPC代码。长度为2904比特，信噪比为5.1dB，码率为0.878，列权重为4。

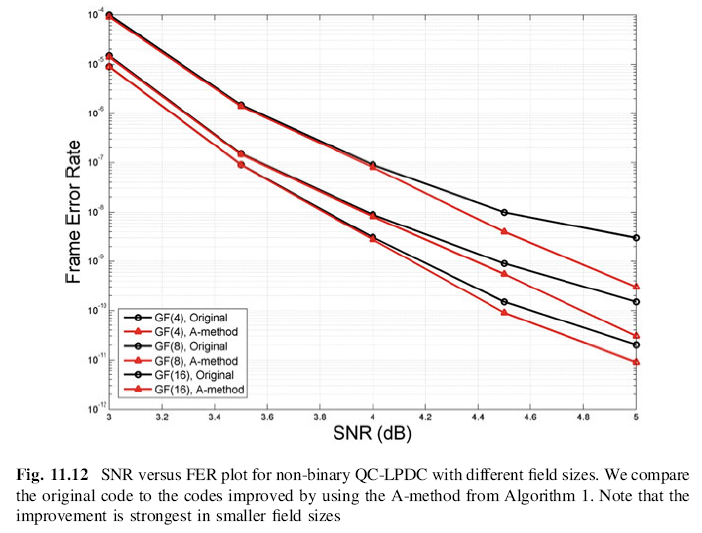
接下来，在图11.11中，我们展示了一个性能图，显示了使用该算法（标记为A方法）的效果。我们还与其他几种算法进行了比较，这些算法也试图通过吸收/陷阱集修改来解决错误下限。特别是，我们与[17]中提出的方法进行比较，我们将其称为“P 方法”。这种方法试图消除Tanner 图中长度为l的所有循环，其中l介于周长g和lmax参数之间。这种消除也具有去除某些吸收集的效果（尤其是非常小的吸收集）。另一种方法我们比较，我们称之为“N-方法”，是在[18]中提出的。

此处，该图显示了原始代码和所讨论的三种方法的帧错误率 (FER) 与 SNR 的关系。 请注意，前面介绍的算法在FER中产生了最佳的整体改进。码长约2930比特，码率0.88，列权重4，解码采用QSPA-FFT解码器。

在非二进制准循环（NB-QC LDPC）码的情况下，这是一类非常实用的非二进制LDPC码，我们得到的结果如图11.12所示。这里的长度约为1400，码率大约0.81，列权重为4，并且再次使用 QSPA-FFT 解码器。







我们注意到，随着域大小q的增加，对基线的改进会减少。这样做的原因是，由于对于较大的域大小有更多的边权重选择，吸收集自然以较小的概率出现，因此通过任何可能的算法都可以去除更少的边缘权重。

## 总结

本章研究了两类非标准代码。第一类由代数码组成，仅依赖硬信息，适用于需要简单高效解码但容错性较宽松的应用，如廉价的数据存储设备。第二类由 LDPC 码组成，解码更复杂，但性能非常好。因此，LDPC码适用于需要极高可靠性的应用。闪存设备占据了这两个端点之间的整个频谱。我们研究的两类高级代码都比传统代码有了显着改进，但同时在构造、设计选择和分析方面提出了更多挑战。

首先，我们检查了非对称代数码。我们通过查看对数据存储设备的物理通道进行建模的不对称通道来推动这项研究。结果表明，使用传统的对称码在码率或纠错能力方面都是浪费的。我们还讨论了两种类型的非对称码：基于张量积运算的分级比特纠错码和依赖于动态阈值边信息技术的基于动态阈值的码。

之后，我们研究了非二进制LDPC码，与经常研究的二进制LPDC码相比，它提供了更好的性能。我们检查了非二进制情况下的错误下限问题，并定义了导致错误下限的底层非二进制吸收集合对象。我们引入了一种算法，该算法可以有效地从非二进制LDPC码的Tanner 图中去除有问题的小吸收集。仿真结果显示出对基线非二进制LDPC码的显着改进。