第三章　信道编码

3.6 卷积码

（第14讲　2007.11.20.）

本节的主要内容

- **3.6.1** 卷积码的产生
- **3.6.2** 卷积码的图示
- **3.6.3** 维特比（**VB**）译码
- **3.6.4** 卷积码的距离特性
- **3.6.5** 软判决VB译码

外语关键词

卷积码：convolution code

移位寄存器：shift register

冲击响应：impulse response

转移函数矩阵：transfer function matrix

格图：trellis diagram

维特比算法：Viterbi algorithm

软判决译码：soft decision decoding

温旧引新：

线性分组码基本概念：

表达方式：(n,k)码，k是信息位数，r是监督位数，$n=k+r$是码长。

编码方法：

已知信息K（k位二进序列），求相应码字的方法是 $C=KG$，G叫生成矩阵，是k行n列的， 一般G具有 $[I_k\ Q]$ 的形式， I_k 是k行k列单位方阵，Q是k行r列的矩阵。

校验方法：

H叫一致监督矩阵，是r行n列的。一般H具有 $[P\ I_r]$ 的形式，I_r 是r 行r列单位方阵，P是r行k列的矩阵， P与Q互为转置关系。

3.6.1 卷积码的产生

1.卷积码示例：

- 输入信息流为$K= a_0a_1a_2a_3\ldots\ldots$，如每一位信息后跟二位监督，信息位$a_i$的监督位为$p_{i1}$和$p_{i2}$，三者构成一个码段。
- 各段依次轮流出现，构成卷积码：

$$C=a_0\,p_{01}\,p_{02}\,a_1\,p_{11}\,p_{12}\,a_2\,p_{21}\,p_{22}\,a_3\,p_{31}\,p_{32}\ldots\ldots$$

- 与分组码不同的是监督位p_{i1}和p_{i2}的值由它前面四个码段中的信息位决定。比如，满足线性方程：

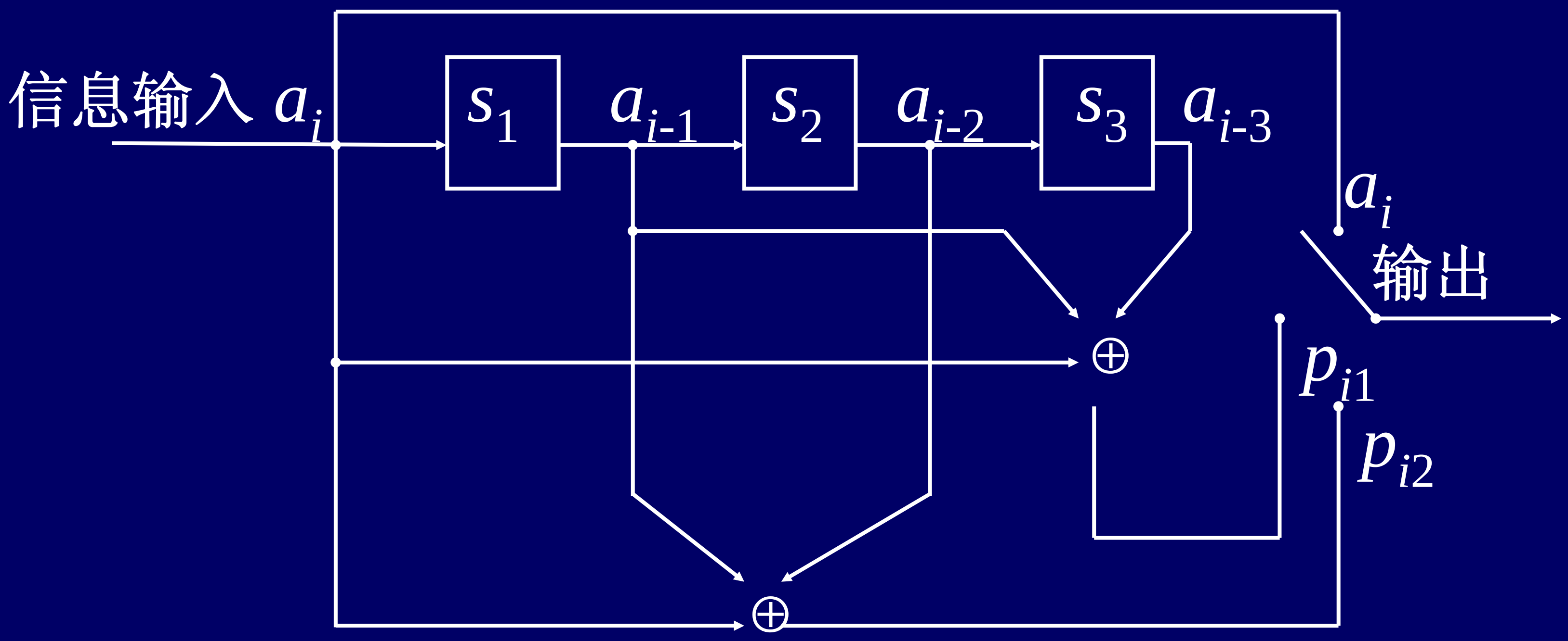
$$\begin{cases} p_{i1} = a_i + a_{i-1} + a_{i-3} \\ p_{i2} = a_i + a_{i-1} + a_{i-2} \end{cases}$$

- 从而使整个码流连锁嵌套，无法再按原来的分组进行分割。

2.比较：

- 线性分组码是“块码”，以k位信息码为单位，编出对应的r 位监督位，构成一个码长为$n=k+r$位的码字。第i个输出码字只取决于第i个输入信息段，各个分组，即各个码字之间没有联系；
- 卷积码在相当于原来的$m+1$个码字的范围内实现信息位与监督位的关联，每个监督位的值由它前面 ($m+1$) 个码字中的信息位共同决定，是有记忆的编码。整个码流所有的码元都已关联起来，形成一个相互制约的锁链，从结构上讲，属于序列“流码”。

(3,1,3)卷积码编码电路原理图



- （***n***，***k***，***m***）卷积码： ***k***位输入（***k*=1**）时，***n***位输出（***n*=*k*+*r* =3**），并由***m***级(***m*=3**)移位寄存器构成，故命名为 **(3, 1, 3)**卷积码。

3.编码器输出对输入的响应：

- 线性系统在时域中输出对输入的依赖关系可用冲击响应描述.
- 视编码器为一线性系统，且移位寄存器初态为(000)，当输入为冲击信号 δ =(1, 0, 0, 0……)时，由于关联长度为m+1=4，所以冲激响应非零长度为4。得到：
  - a_i端（记作c_1）输出为：$g_1 = (1,0,0,0)$;
  - p_{i1}端（记作c_2）输出为：$g_2 = (1,1,0,1)$;
  - p_{i2}端（记作c_3）输出为：$g_3 = (1,1,1,0)$;

- 当输入为任意信号$U=(u_0,u_1,u_2,u_3,u_4\ldots\ldots)$时，三个输出端的响应分别为：

  $C_1=a=U*g_1$； $C_2=p_1=U*g_2$； $C_3=p_2=U*g_3$；

  合写为： $C_j=U*g_j$ $(j=1,2,3)$

- C_j的第l位为：（ $l=0, 1, 2, 3, 4, \ldots\ldots$）

$$C_j^l=\sum_{i=0}^{l} g_j^i u_{l-i}$$

- 输入信息流与输出的编码流之间是的卷积的关系，因而称为卷积码。

- 如：u=(10111……)，根据卷积求和的定义，不难求得：

$$C_1 = u * (1,0,0,0\ ) = (10111\ldots\ldots),$$
$$C_2 = u * (1,1,0,1\ ) = (11110\ldots\ldots),$$
$$C_3 = u * (1,1,1,0\ ) = (11001\ldots\ldots);$$

- 最终输出的卷积码由这三个输出端每次各出一位轮流输出构成：

$$C = (c_1{}^0 c_2{}^0 c_3{}^0 c_1{}^1 c_2{}^1 c_3{}^1 c_1{}^2 c_2{}^2 c_3{}^2\ldots\ldots)$$
$$= (111,011,110,110,101,\ldots\ldots)$$

还可以用转移函数矩阵来表示输出对输入的响应.

- 用D表示移位寄存器的时延，则输出与输入的依赖关系又可表达为转移函数矩阵的形式：

$$G(D)=(1,\ 1+D+D^3,\ 1+D+D^2)$$

- $G(D)$：
  - 是k行n列的矩阵；本例是$k=1$，$n=3$
  - 每个矩阵元是D的m次多项式；本例是$m=3$
  - 多项式的系数分别是 $g_1=(1,0,0,0)$, $g_2=(1,1,0,1)$ 和$g_3=(1,1,1,0)$；
  - 转移函数矩阵表达了编码器的电路结构。

4.卷积码的生成矩阵（可略）

若把输入信息流u=(10111……)与输出的编码流

C= (111,011,110,110,101,……)

之间的关系写为生成方程$C = u\cdot G$的形式，其

中生成矩阵G可表达为：

$$G=\begin{pmatrix} 111 & 011 & 001 & 010 & 000 & 000 & 000 & 000 & \cdots \\ 000 & 111 & 001 & 001 & 010 & 000 & 000 & 000 & \cdots \\ 000 & 000 & 111 & 011 & 001 & 010 & 000 & 000 & \cdots \\ 000 & 000 & 000 & 111 & 011 & 001 & 010 & 000 & \cdots \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \end{pmatrix}$$

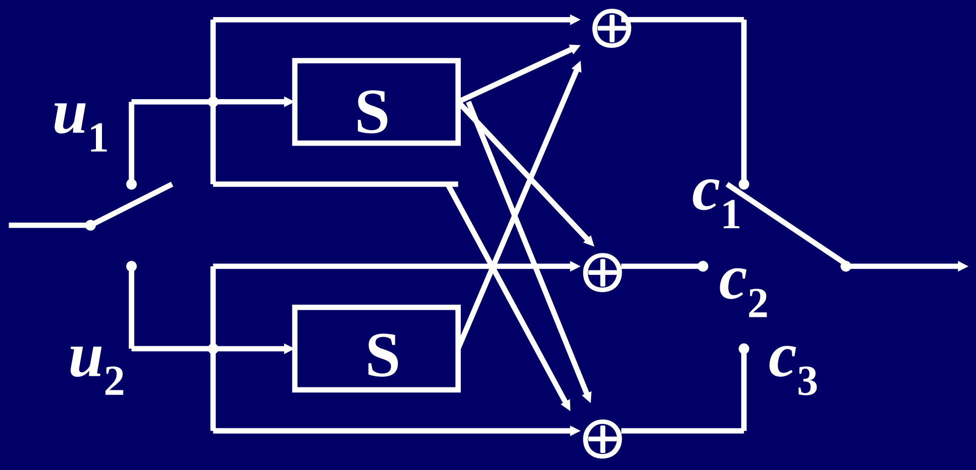
基本生成矩阵 [*g*]:

- *G*是一个半无穷矩阵，但其中有效数字只有：

[*g*] = [111 011 001 010]，它在G矩阵中每行都出现，只是依次错后3位。而G矩阵其余元素都是0。

- [g]由g_1 = (1,0,0,0), g_2 = (1,1,0,1), g_3= (1,1,1,0) 各出一位轮流输出构成。
- [*g*]叫做基本生成矩阵，反映了监督元与信息元之间的约束关系。
- 现在[*g*]的行数是1，由k=1决定；
- 共12个码元，分为四段（由m+1=4决定），每段三个码元（由n=3决定）。

例：(3， 2， 1)卷积码

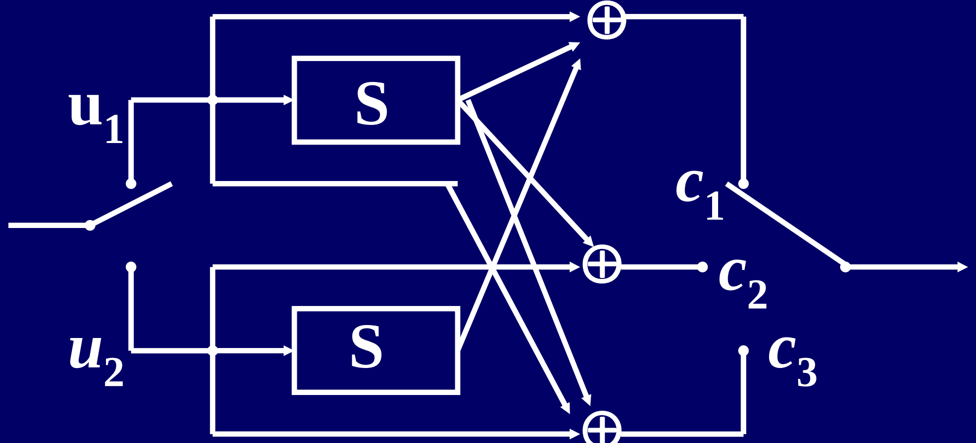


(3,2,1)卷积码编码电路原理图

（3, 2, 1）卷积码：

- 两个输入（*k*=2）
- 三个输出（*n*=*k*+*r*=3）
- 由一级(*m*=1)移位寄存器构成，

- 关联长度为$m+1=2$，所以冲击响应非零长度为2。可直接由编码电路写出冲击响应：



- 当第一个输入端加冲击信号 δ=(1,0,0,…) 时，三个输出分别是： g_{11} = (1,1)，g_{12} = (0,1)，g_{13}= (1,1)；
- 当第二个输入端加冲激信号 δ = (1,0,0,…) 时，三个输出分别是： g_{21} = (0,1)，g_{22} = (1,0)，g_{23}= (1,0)；

- 第一个输入端的冲激响应是：

$$g_{11} = (1,1), \; g_{12} = (0,1), \; g_{13} = (1,1);$$

- 第二个输入端加冲激响应是：

$$g_{21} = (0,1), \; g_{22} = (1,0), \; g_{23} = (1,0);$$

- 转移函数矩阵是***k*=2**行，***n*=3**列的矩阵， 各矩阵元都是***D***的***m*=1**次多项式，g_{ij}为各项系数，即：

$$G(D) = \begin{pmatrix} 1+D & D & 1+D \\ D & 1 & 1 \end{pmatrix}$$

- 基本生成矩阵是各g_{ij}拿出一位轮流输出构成：（可略）

$$[g] = \begin{pmatrix} 101 \; 111 \\ 011 \; 100 \end{pmatrix}$$

- 基本生成矩阵是：

$$[g]=\begin{pmatrix} 101 & 111 \\ 011 & 100 \end{pmatrix}$$

- 当输入为u_1=(101)，u_2=（110）时，输出编码是：

$$C=u\cdot G=\left(11\ \ 01\ \ 10\right)\cdot\begin{pmatrix} 101 & 111 & 000 & 000 \\ 011 & 100 & 000 & 000 \\ 000 & 101 & 111 & 000 \\ 000 & 011 & 100 & 000 \\ 000 & 000 & 101 & 111 \\ 000 & 000 & 011 & 100 \end{pmatrix}=(110\ \ 000\ \ 001\ \ 111)$$

3.6.2 卷积码的图示

1.二元码树图:

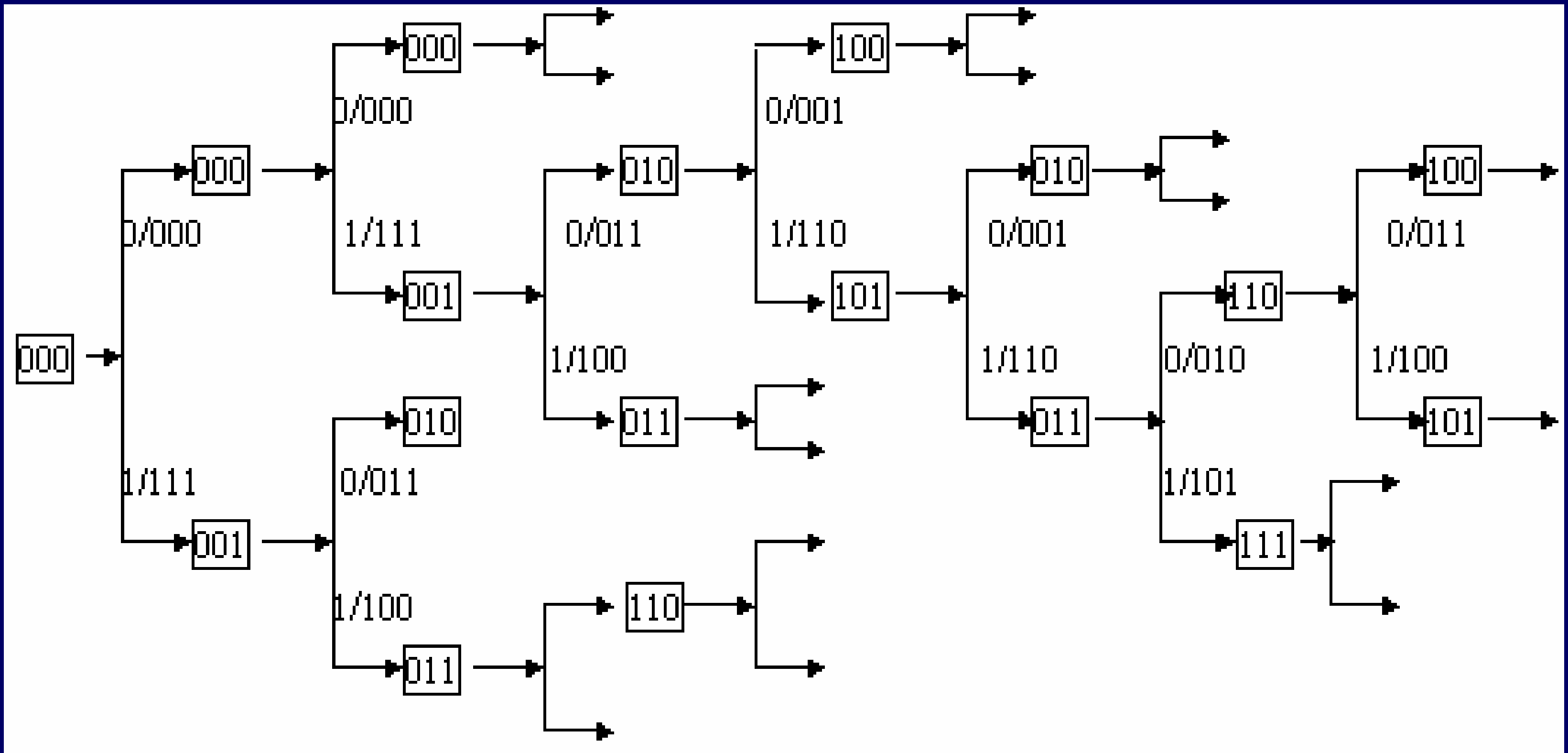
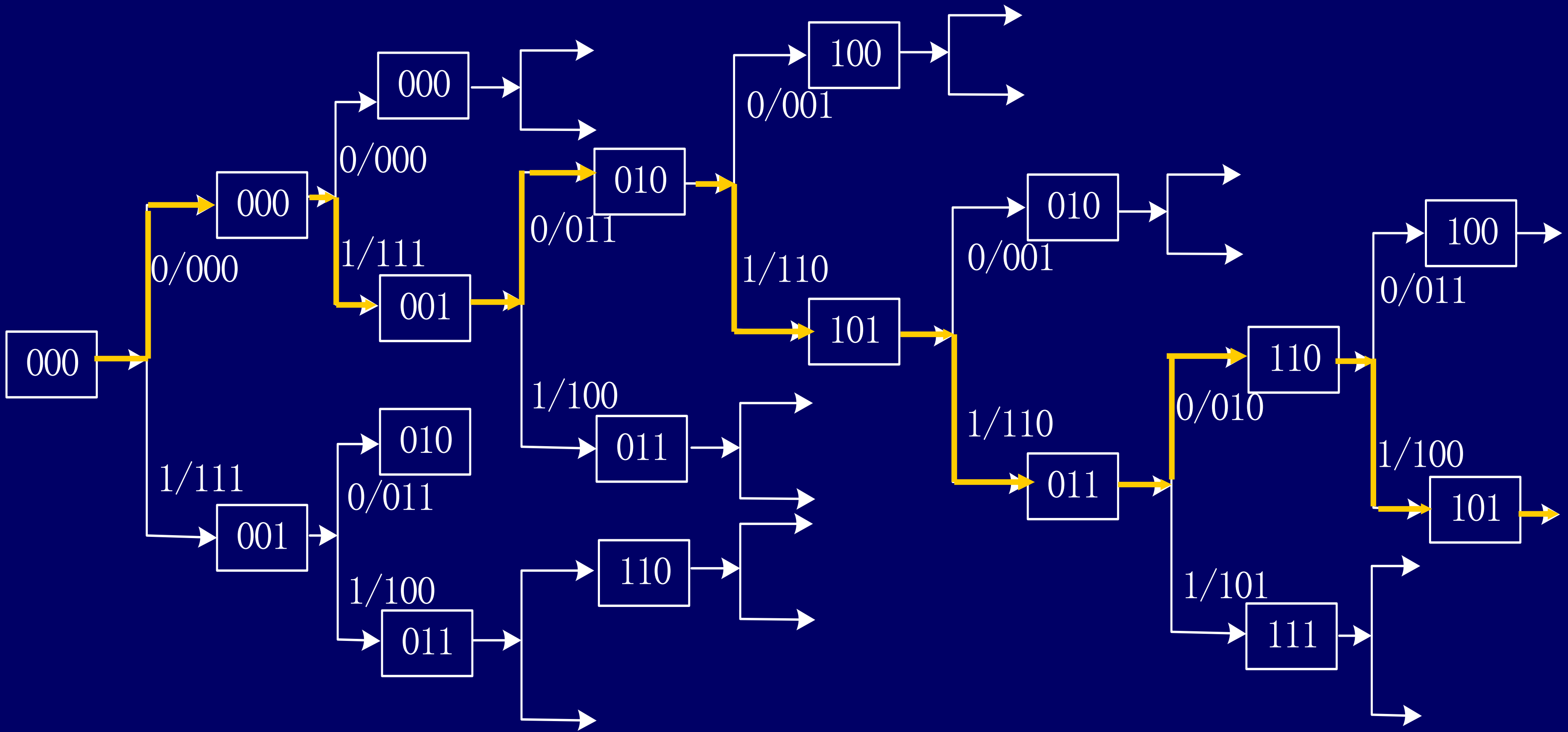


图 3.10　(3,1,3)卷积码的码树

以（**3, 1, 3**）卷积码为例画码树。约定用$\boxed{a_{i-3}a_{i-2}a_{i-1}}$表示发出当前符号$a_i$时的状态。在每条支路旁，把输入的被编符号写在 **“/ ”**左面，输出的已编代码写在**“/ ”**右面。

用码树进行编码

信源发出序列为=(0101101……)；则编码由图得到：



沿着输入 *u*=(0101101……) 的路径，

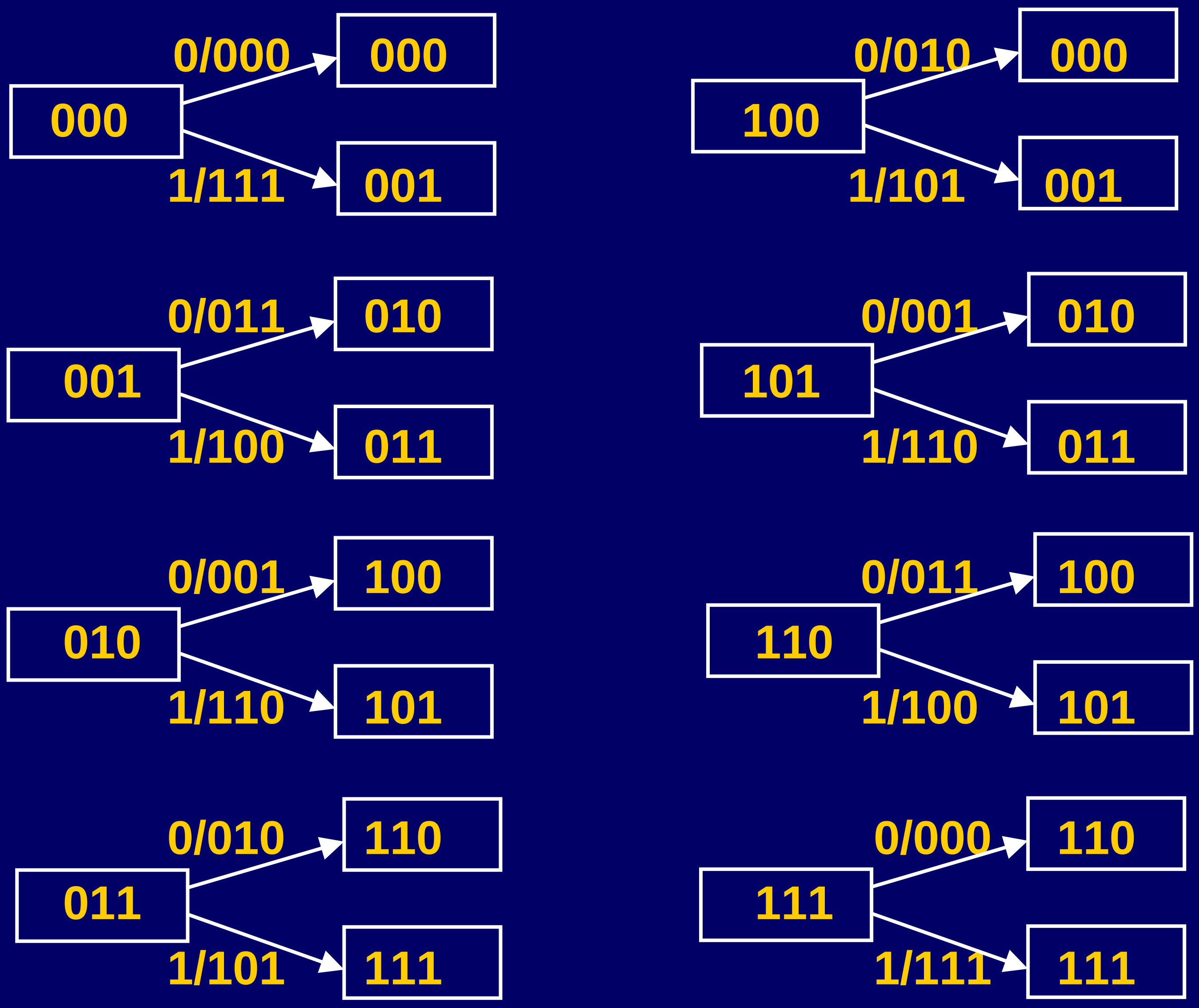
得到编码为：*C*=(000 111 011 110 110 010 100……)；

2. 状态转移图

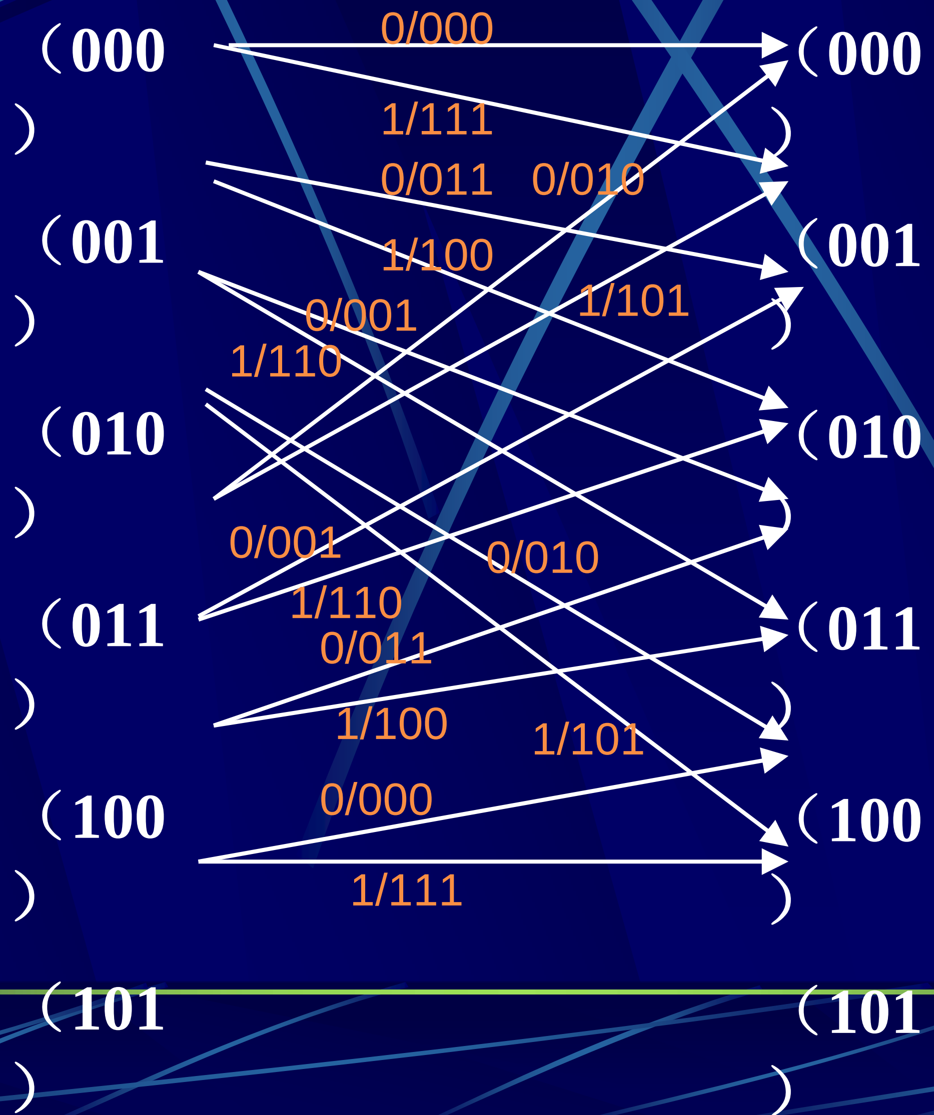
- 由码树看到，只要出现相同的状态，其后的分枝状况就与前面完全重复了。

- 这是因为*m*=3的卷积码状态只有3位，也就是说只有8个不同的状态，每个状态下分别发出0或发出1后，新的状态一定还是这8个状态之中的另一个。

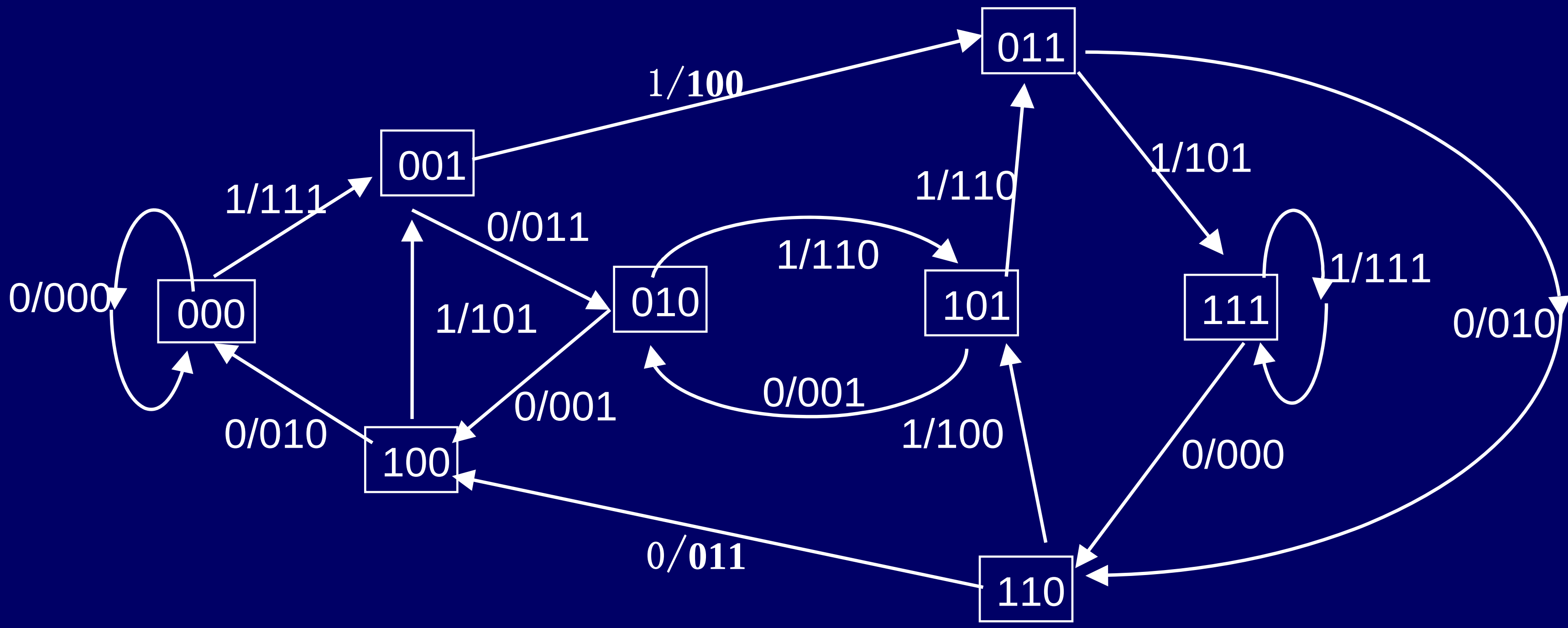
8种状态下发0发1的状态转移关系:



(3,1,3)卷积码的状态转移图



卷积码的状态转移图也可以画成信号流图形式



3．格图（篱笆图）

- 状态转移图概括性强，但是不如码树图那样能显示编码路线。把二者的优点都结合起来，相当于把码树中重复的部分合并，就得到了格图。
- 卷积码的网格图描述：将状态转移图按时间展开，用于描述从第k时刻的编码器状态到第k+1时刻的编码状态的转移情况，以及在转移过程中的输出情况。

(3,1,3)卷积码的格图

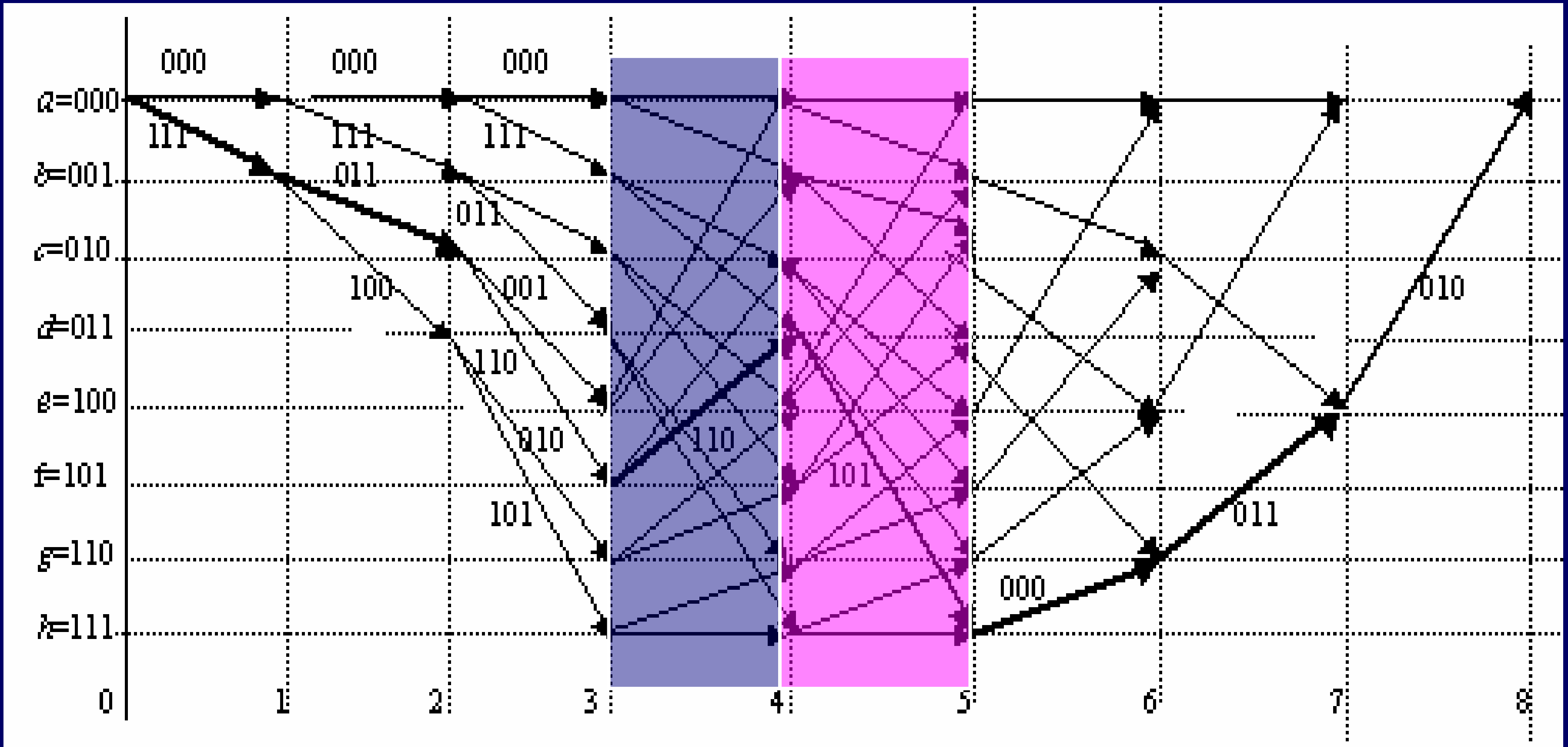


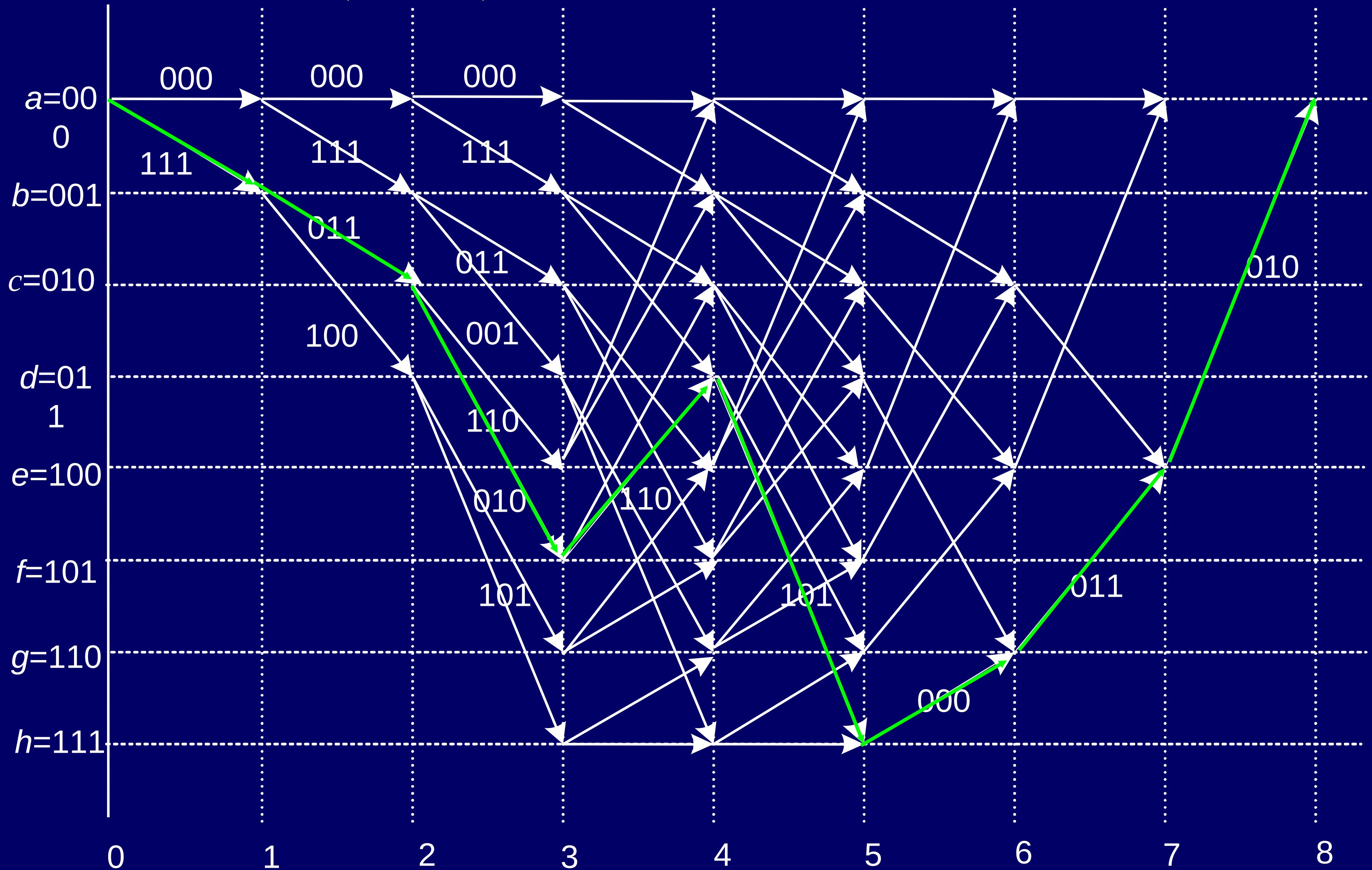
图 3.12 (3,1,3)卷积码的格图

㈠横线表示状态，竖线表示时序，交叉点称为节点。

㈡每个状态只能向它相邻的下一时刻状态转移。对于*k*=1的卷积码，只存在发0和发1两条转移射线，称为支路。

㈢输出代码就标记在相应的支路旁边。

(3,1,3)卷积码的编码



输入$\boldsymbol{u}$=(101110000)　输出C = (111,011,110,110101,000,011,010)

卷积码译码存在的困难：

- *译码比较麻烦*，许用码序列是格图上已存在的连续路径，而含有差错的接收码序列在格图上是不连续的路径（在不同的许用路径之间跳变），译码就是要寻找一条最接近接收码路径的许用路径。而且译码不能等序列全部收到后再进行，应当是边收边译的“流译码”。
- *工作量太大*，因为两序列的接近程度是用汉明距离来度量的，若接收码序列通过100个节点，各种可能的路径数目就高达$2^{100}=1.2\times10^{30}$，全部计算各条路径的汉明距离，工作量太大。

3.6.3 维特比(VB)译码

维特比译码法

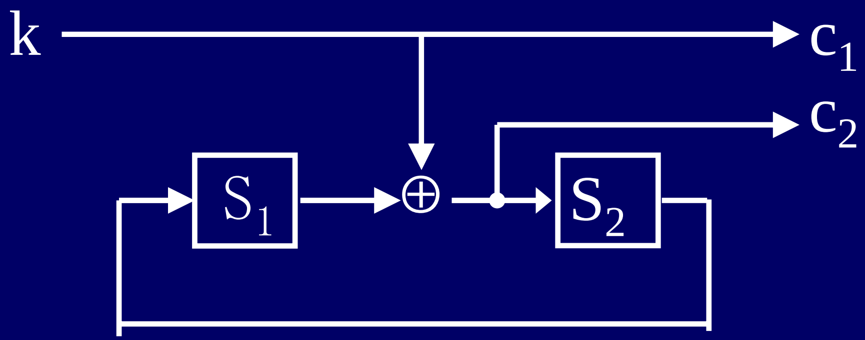
- 维特比译码是一种全局最优的最大似然译码算法。是一种实用的近似计算方法，采用逐步处理的工作方式，每步只对网格图上当前时刻进行计算。边输入边计算，边计算边输出，实时完成序列流译码。

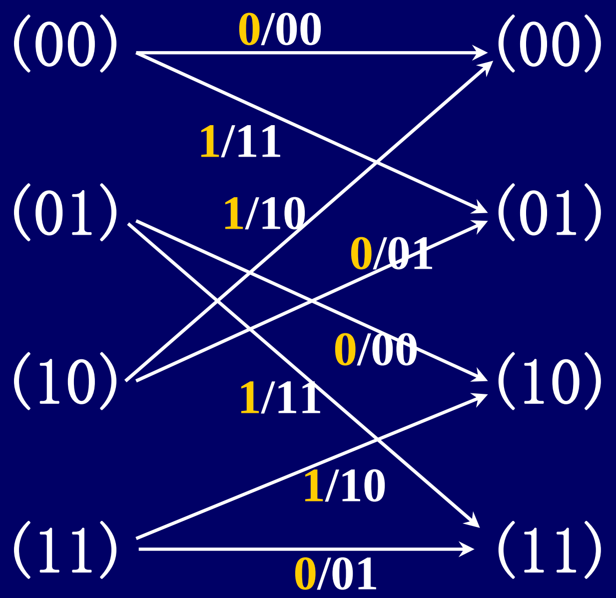
- 某时刻节点与它的上一节点之间的连线称为支路，先算出每条支路的许用码字与该时刻接收码字的汉明距离，称为支路度量。
- 从初始时刻0状态的节点开始，到达指定的节点，许用路径可能有若干条，每条路径与接收码序列的汉明距离被称为路径度量。
- 路径度量采用迭代计算法，初始时刻路径度量为0，每前进一步，只须加上新经过的那条支路的支路度量即可。
- 对于指定的节点，进入该节点的许用路径有若干条，但只须保留路径度量最小的一条作为存留路径，其它路径均不再考虑，这是因为其它路径对于该节点后续路径度量的贡献都比存留路径大。

- **[例]（2, 1, 2）卷积码译码过程**

设转移矩阵为：　$G(D)=(1,\ 1+D^2)$

编码电路和状态转移图 如下：

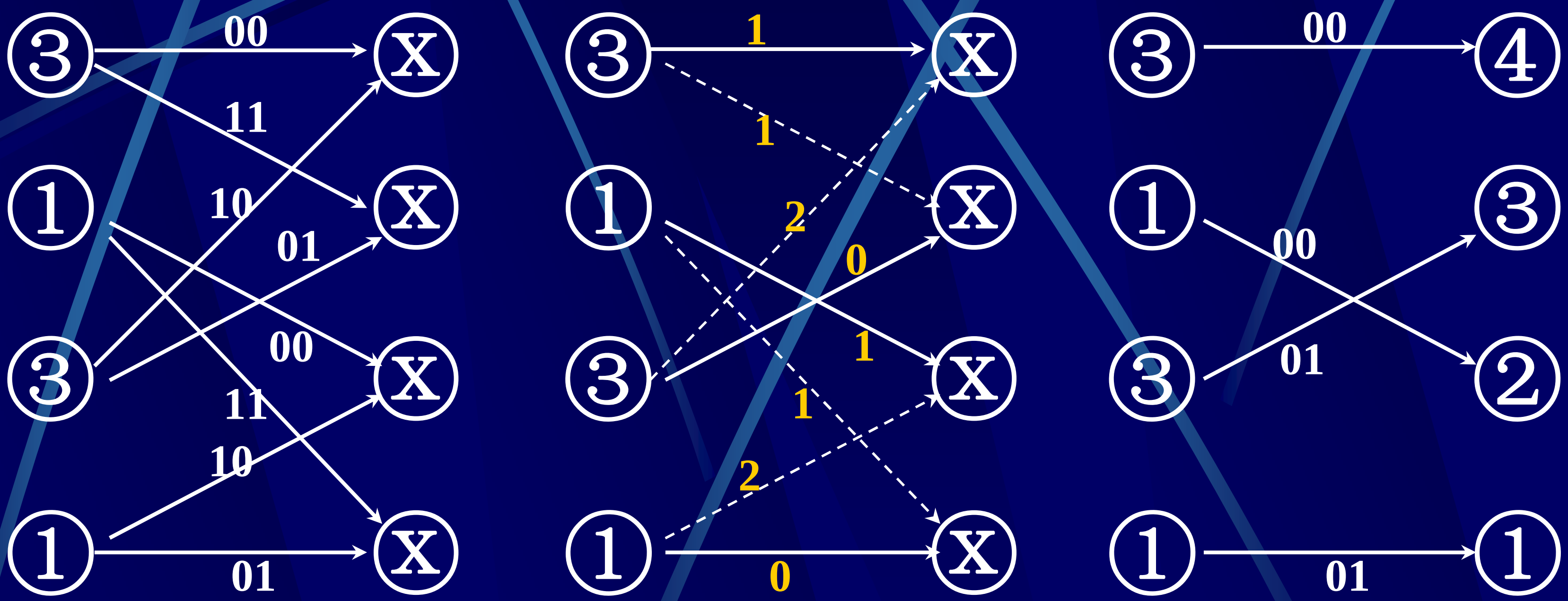




(2,1,2)卷积码的编码电路和状态转移图

- 已知信息流u=(11011)的编码为：C=(11, 11, 01, 10, 10)；设经过有噪信道，接收到的码流为： R=(1<u>0</u>, 11, 01, 1<u>1</u>, 10)；其中2个码元发生了孤立错误，见下划线标记。
- 译码过程从初态00开始，直到第2时刻，进入每个节点的路径都只有一条，它们都是存留路径，圆圈内的数字是该节点存留路径的路径度量。
- 进入第3时刻以后，节点仍然是4个，但每个节点有两条路径进入，经过计算和比较，删去了虚线所示路径，实线是保留路径。

支路度量、路径度量和存留支路：

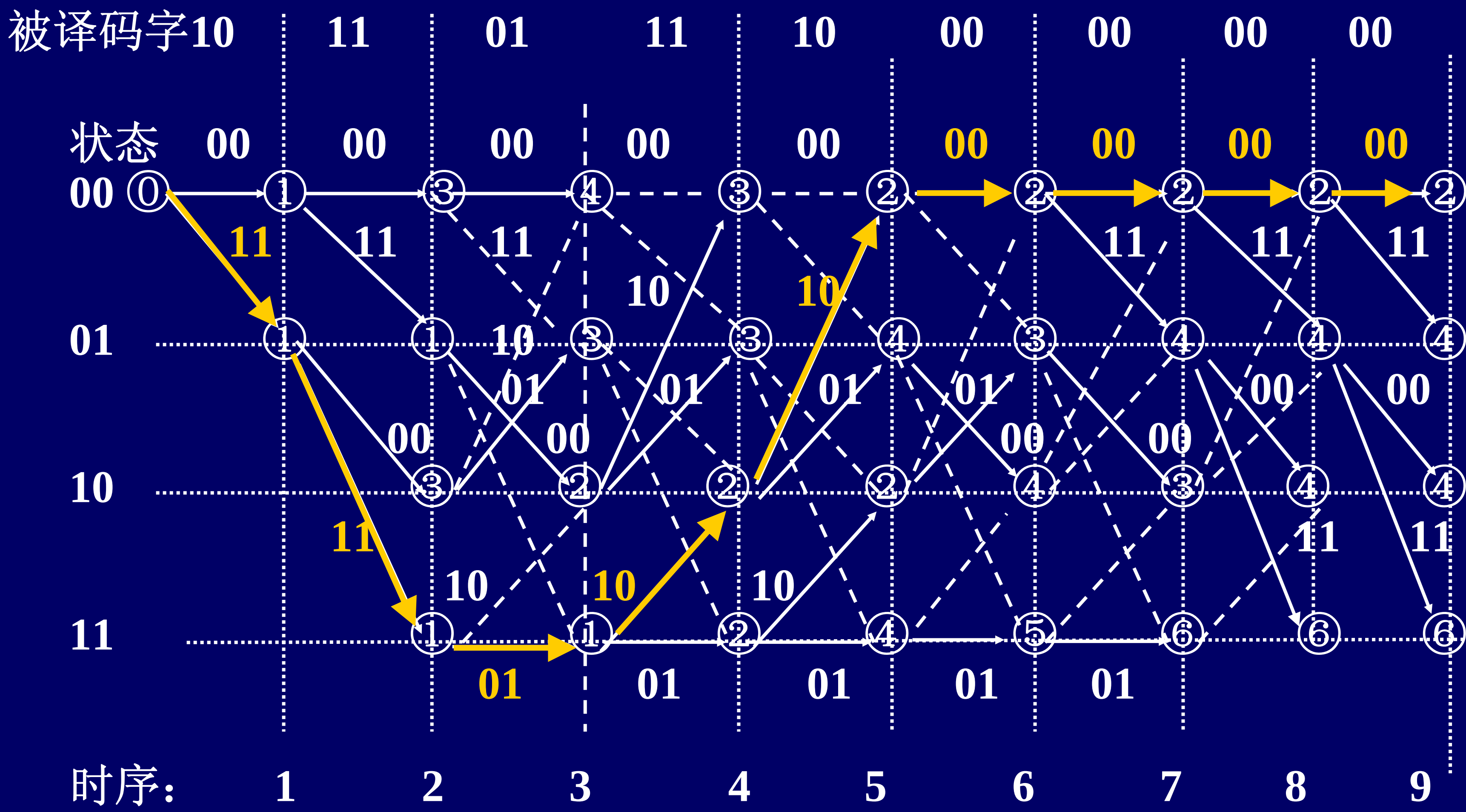


上面的图中③原来存留路径的度量，支路度量用黑色数字01等表示，得到中图所示的支路度量(黄色数字)。

对每个圆圈ⓧ，先算出路径的度量，两条进入的路径保留一条路径度量较小的，得到存留路径，并将路径度量写入右面的圆圈。

- 每个节点都做相同的计算和处理，各自保留唯一的一条优选存留路径。然后转入下一时刻的计算和处理，直到最后输入结束。
- 由于节点数目等于状态数目，每一时刻要计算的节点数是不变的。表明**VB**译码法的计算量不会序列的申延而越变越大。
- 最后，从各条优选存留路径中选出路径度量最小的一条，它就是与接收码流最接近的许用路径，沿此路径的编码序列即为最似然的译码。

(2,1,2)卷积码的格图和译码过程



- 尽管每个节点存留的路径互不相同，然而我们却注意到，第5时刻以后，四条存留路径起始的几段支路已经合并在一起，就是说，经过一定的延时，卷积码的纠错功能发挥了作用，已经筛选出了部分汉明距离最小的许用码字段落。

- 随着计算过程的前进，被合并的部分越来越长，并且已合并的路径不会因为后续的处理而改变。于是可以把被合并的部分作为译码而输出。

- 一般情况下，延时长度大于状态数目的五倍，就完全可以进行输出了。
- 本例很简单，取延时为4个时间单元即可。考虑到延时的影响，应当在被译码的接收码序列后面添加若干个0，同时应当为每个状态各开辟一个4单元的存储器，用来存储4个时延单元（当前时刻及其前三个时刻）的码字。
- 四个状态存储器初始值均取为00：

$$S_{00}(0)=S_{01}(0)=S_{10}(0)=S_{11}(0)=(00,00,00,00)$$

●每进入一个时刻，都进行以下操作：

（1）交换4个存储器的内容：假如存留路径是从 j 状态进入 i 状态，则将 j 存储器的内容替换到 i 状态存储器中。

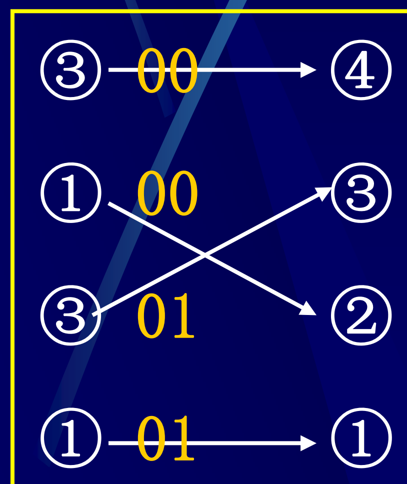
如(2, 1. 2)码从时刻2到时刻3，四个节点的交换情况是

$S_{00}(2)=(00,00,00,00)$

$S_{01}(2)=(00,00,00,11)$

$S_{10}(2)=(00,00,11,00)$

$S_{11}(2)=(00,00,11,11)$



$S_{00}(3)=(00,00,00,00)$

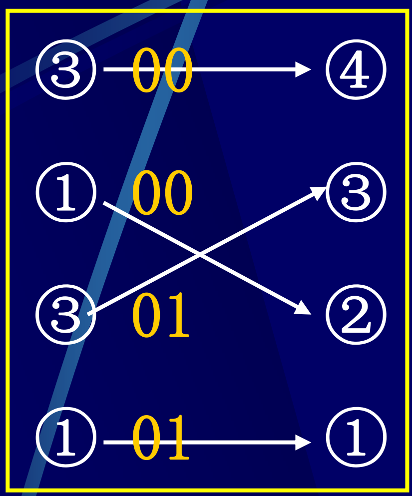
$S_{01}(3)=(00,00,11,00)$

$S_{10}(3)=(00,00,00,11)$

$S_{11}(3)=(00,00,11,11)$

(圆圈是路径度量，存留支路傍边写的是许用码字）

（2）将该存留支路的许用码字分别写入相应存储器的最后2位，同时把其它各位统统左移2位。



$S_{00}(3)=(00,00,00,00)$ 左移 $\cancel{00}(00,00,00,00)$

$S_{01}(3)=(00,00,11,00)$ 左移 $\cancel{00}(00,11,00,01)$

$S_{10}(3)=(00,00,00,11)$ 左移 $\cancel{00}(00,00,11,00)$

$S_{11}(3)=(00,00,11,11)$ 左移 $\boxed{00}(00,11,11,01)$

（3）对于路径度量最小的节点，其状态存储器中移出的码字作为译码输出；其它节点的状态存储器移出的码字则被丢弃。

沿最佳路径各时刻状态存储器的转换与内容变化如下：

$S_{00}(0)=(00,00,00,00)$，是初态； → $S_{01}(1)=(00,00,00,11)$，输出00；

→$S_{11}(2)=(00,00,11,11)$，输出00； → $S_{11}(3)=(00,11,11,01)$，输出00；

→$S_{10}(4)=(11,11,01,10)$，输出00； → $S_{01}(5)=(11,01,10,10)$，输出11；

→$S_{00}(6)=(01,10,10,00)$，输出11； → $S_{00}(7)=(10,10,00,00)$，输出01；

→$S_{00}(8)=(10,00,00,00)$，输出10; → $S_{00}(9)=(00,00,00,00)$，输出10；

如果继续作下去，各节点的路径度量已不再改变，至此，译码结束。得到：$C'=(00,00, 00,00,11,11,01,10,10)$；

去掉前面因延时而带来的8个0，结果与发端编码一致，差错已得到纠正。

3.6.4 卷积码的距离特性

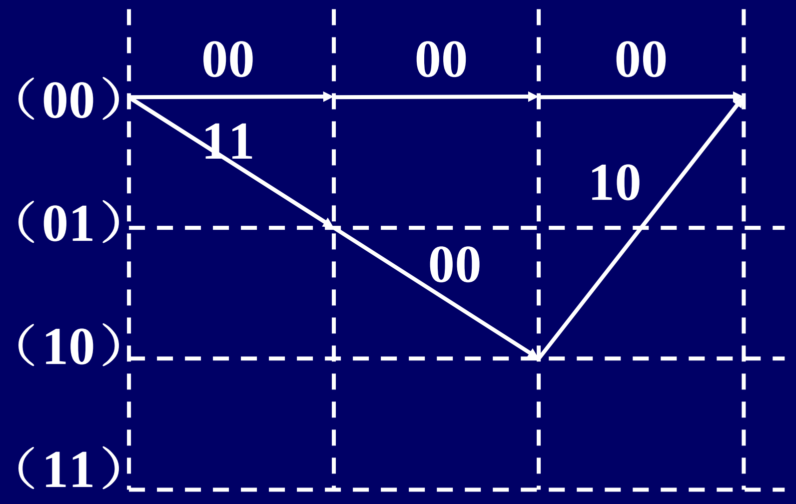
- 分组码的纠错能力取决于码字的最小汉明距离。
- 卷积码的纠错能力也取决于码流序列之间的最小汉明距离。
- 通常，人们把序列趋于无穷长时，格图上所有许用路径之间的最小汉明距离叫做卷积码的自由距离。
- 自由距离是衡量卷积码性能的主要指标，纠错能力和误码概率都与自由距离直接有关，选用或设计卷积码离不开这个参数。

自由距离的确定

* 卷积码属于线性码，所以任意两个码序列按位模二加之和仍然是一个许用码，而它的重量（1的个数）就等于这两个码序列之间的汉明距离。
* 可见，只要在所有的码序列中找到最小重量的许用码，它的重量就是卷积码的最小汉明距离。而任意许用码序列的重量又等于它与全0码之间的汉明距离。
* 所以，只要在格图上找到一条离全零路径最近的、从0状态出发又回到0状态的非全0路径，那么这条路径所代表的码序列的重量就等于自由距离。

简单卷积码的自由距离可以由格图直接得到。

- 例：如上述（2, 1, 2）卷积码
- 格图上重量最小的非全零路径是 (00)→(01)→(10)→(00)，许用码序列为 (11,00,10)，其路径重量为3，故自由距离d_f=3。

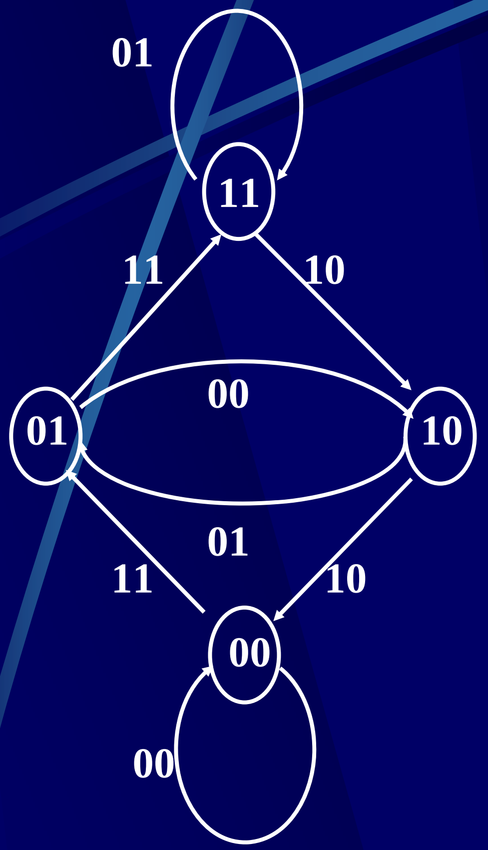


(2,1,2)卷积码的格图和译码过程

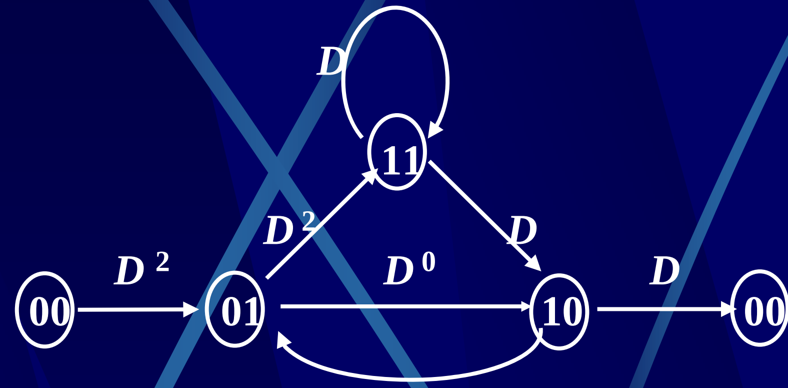
- 复杂卷积码的自由距离可由信号流图来计算或者用计算机搜索。

- 以（**2, 1, 2**）卷积码为例，把状态转移图（下页图**A**）的（**00**）态拆分为一个输入态和一个输出态，把每条支路上的码字重量表达成D的幂次形式：码字重量为n，该支路的权重就是就D^n，这样就得到了信号流图。

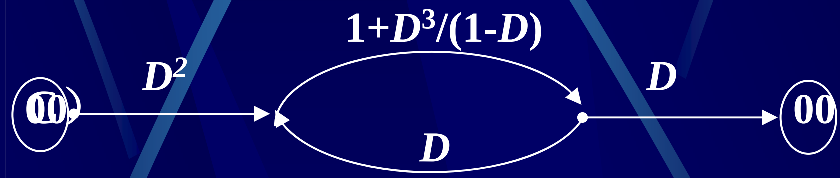
(2,1,2)卷积码的信号流图和化简过程



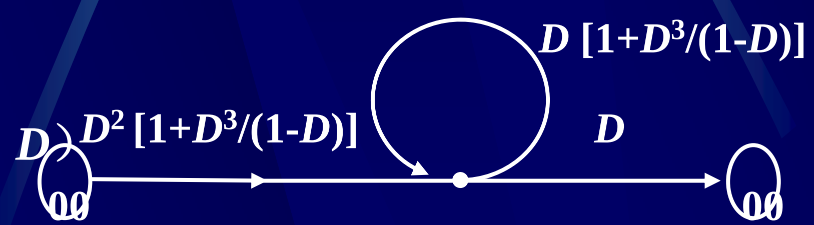
（图 A）



（图B）



（图



（图

（图

- 对信号流图化简，消去自环和支路，得到生成函数：

$$T(D)=\frac{D^2[(1+D^3/(1-D)]}{1-D[1+D^3/(1-D)]}=\frac{D^3-D^4+D^6}{1-2D+D^2-D^4}$$
$$=D^3+D^4+D^5+\cdots\cdots$$

- 一般情况下，生成函数总可以写成下式所示的形式：

$$T(D)=\sum_{d=d_f}^{\infty}A_d D^d$$

此公式表明，该卷积码含有无穷多条不同重量的路径，其中重量为d的非全0路径有A_d条，非全0路径最小重量为d_f。

3.6.5软判决VB译码

1. 概述

- 用最小汉明距离为判据的译码叫做硬判决译码，它只适用于二元对称（**BSC**）信道。
- 对于其它信道，汉明距离最小不一定是最佳。采用**DMC**信道模型可能更恰当。
- **DMC**信道是多元离散无记忆信道，输入和输出可以是不同的**Q**元符号集。理论上**Q>2**时，从**Q**个量化电平来判定输入符号肯定要比用**2**个量化电平判定的可靠性要高。

- 为了适应这种Q进制信号的判决，我们直接由最大似然准则（或最大后验概率准则）出发，寻找一种基于概率的译码算法，也就是所谓软判决译码。
- 设某2进4出的DMC信道的传输矩阵为：

$$P(Y \mid X)=\begin{pmatrix} 0.4 & 0.3 & 0.2 & 0.1 \\ 0.1 & 0.2 & 0.3 & 0.4 \end{pmatrix}$$

式中：$x_i \in X = (0, 1)$， $y_j \in Y = (0_1, 0_2, 1_1, 1_2)$

- 为了方便，用前向概率的对数来描述码元的似然程度，叫做码元度量：

$$M(y_j \mid x_i) = \log P(y_j \mid x_i)$$

- 以10为底时各个概率值对数均为小数：

$$M(Y|X)=\begin{pmatrix}-0.4 & -0.52 & -0.7 & -1.0\\ -1.0 & -0.7 & -0.52 & -0.4\end{pmatrix}$$

- 为了计算方便，在不影响各个码元度量相对大小的情况下，不妨把码元度量的定义加以调整：

$$M(y_j|x_i)=a_2[\log P(y_j x_i)+a_1]$$

取a_1=1，a_2=17.3就能使所有的码元度量近似为整数。

$$M(Y|X)=\begin{pmatrix}10 & 8 & 5 & 0\\ 0 & 5 & 8 & 10\end{pmatrix}$$

式中：行表示许用码元X= (0, 1),

列表示接收码元Y= (0_1, 0_2, 1_1, 1_2)

软判决译码方法

- 软判决用于维特比译码，与硬判决的做法完全相同，区别仅在于支路度量和路径度量由汉明距离改为上述的码元度量。

  比如某时刻接收的码字为（$1_2 0_2$），而支路上标记的许用码字为（**10**），由$\boldsymbol{M(Y|X)}$看到，第一位接收码元$\mathbf{1_2}$与第一位许用码元**1**之码元度量***10***，第二位接收码元$\mathbf{0_2}$与第二位许用码元**0**之码元度量**8**，则支路度量等于**10+8=18**。

- 另外，度量值越大，代表似然概率就越大，因此，在选择保留路径时，是以码元度量最大者为优选。

软判决译码的优点

- 采用软判决之后，译码设备并不比硬判决复杂，但输出端相同误码率情况下所要求输入段提供的信噪比却可以减小**2~3**分贝。原因在于软判决更充分地利用了信道的统计性质，因此，目前实用的**VB**译码器几乎都用软判决。

本节要点

1. 卷积码的基本概念：
   - 卷积码的生成
   - 冲击响应
   - 转移函数矩阵
2. 卷积码的图示：
   - 码树图
   - 状态转移图
   - 网格图
3. 卷积码的维特比译码：
   - 支路度量和路径度量
   - 唯一的存留路径
   - 时延与输出
   - 软判决VB译码

- 思考：

卷积码为什么有纠正多位错的强大功能？

- 作业：

P115页：20题

第三章 信道编码

3.7 纠正突发错误的编码

（第15讲 2007.11.27.）

本节的主要内容

- 3.7.1 交织码
- 3.7.2 Fire码
- 3.7.3 RS码
- 3.7.4 级连码

外语关键词

交织码：Cross interleave code (CIC)

突发错误：burst error

级连码：Nested codes

移动通信：mobile communication

有限域：finite field

交叉置乱：cross scrambling

交插置乱：interlace scrambling

温旧引新：

- ## 卷积码基本概念：

表达方式：(*n*,*k*,*m*)码。*k*位输入(信息)，*n*位输出(信息加监督)，*m*是移位寄存器位数。要求对逻辑电路、监督方程、冲击响应与转移矩阵四种描述方式能够互换；

- ## 编码方法：

状态转移图与网格图。

- ## 译码方法：

维特比译码：支路度量、路径度量、存留路径 。

引言

- 信道中常有冲击性的干扰（如雷电、火花等），造成连续若干个码元的成片错误，通信中成为“突发错误”。
- 定义连续发生的错误码元长度叫做突发长度，用b表示。
- 纠正突发性错误一般有两种方案。
- 其一是直接设计能够纠正连续多个错位的编码，比如本节后面要讨论的 **Fire码与RS码**。
- 其二是设法将成串的错误分散开来，使它转化位孤立的单个错误，就可以采用纠正独立错误的编码来纠正。这就是马上要介绍的交织码。
- 实践中往往是两种方案结合进行。

3.7.1 交织码

交织编、译码流程：

原理：将信道中的突发错误分散开来，从而能被纠正孤立错误的编码系统来纠正。

交织器

$x_1x_2x_3x_4x_5x_6x_7x_8x_9x_{10}x_{11}x_{12}x_{13}x_{14}x_{15}x_{16}x_{17}x_{18}x_{19}x_{20}x_{21}x_{22}x_{23}x_{24}x_{25}$

按行写入

x_5	x_4	x_3	x_2	x_1
x_{10}	x_9	x_8	x_7	x_6
x_{15}	x_{14}	x_{13}	x_{12}	x_{11}
x_{20}	x_{19}	x_{18}	x_{17}	x_{16}
x_{25}	x_{24}	x_{23}	x_{22}	x_{21}

按列读出

$x_1x_6x_{11}x_{16}x_{21}x_2x_7x_{12}x_{17}x_{22}x_3x_8x_{13}x_{18}x_{23}x_4x_9x_{14}x_{19}x_{24}x_5x_{10}x_{15}x_{20}x_{25}$

解交织器

$x_1x_6x_{11}x_{16}x_{21}x_2x_7x_{12}x_{17}x_{22}x_3x_8x_{13}x_{18}x_{23}x_4x_9x_{14}x_{19}x_{24}x_5x_{10}x_{15}x_{20}x_{25}$

按列写入

x_5	x_4	x_3	x_2	x_1
x_{10}	x_9	x_8	x_7	x_6
x_{15}	x_{14}	x_{13}	x_{12}	x_{11}
x_{20}	x_{19}	x_{18}	x_{17}	x_{16}
x_{25}	x_{24}	x_{23}	x_{22}	x_{21}

按行读出

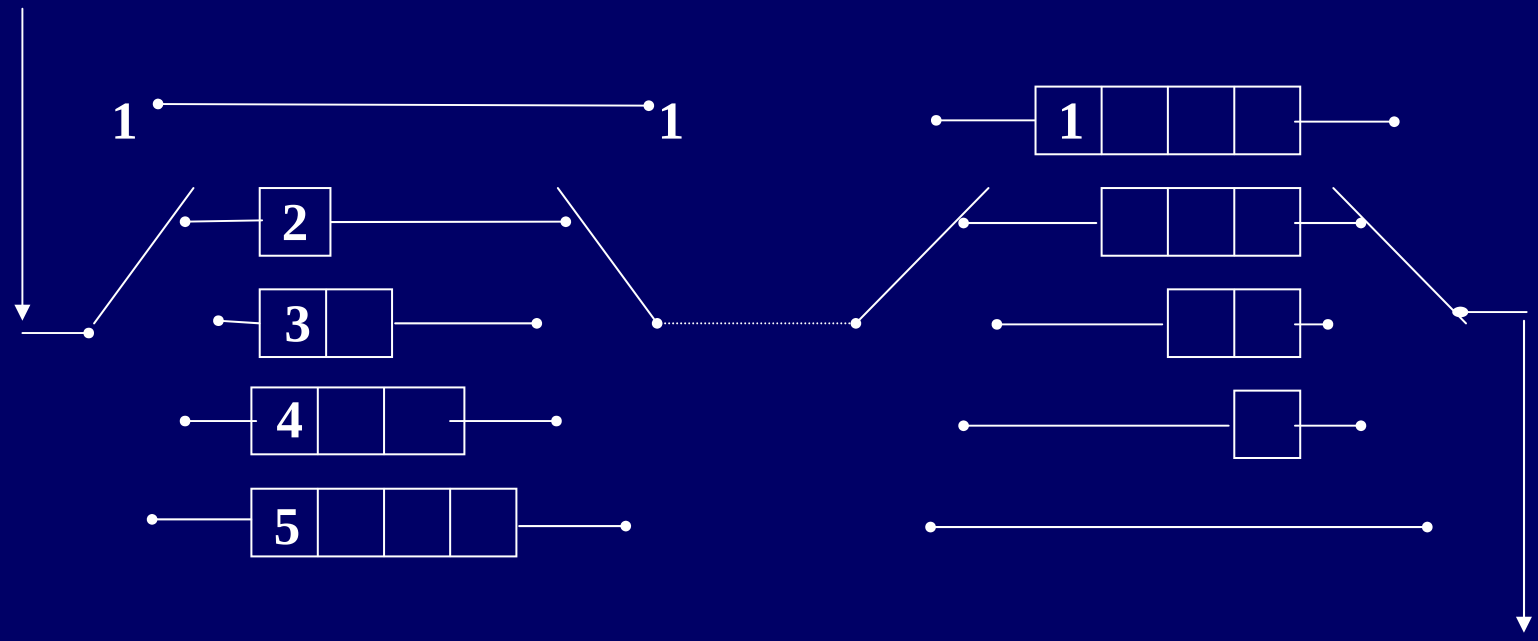
$x_1x_2x_3x_4x_5x_6x_7x_8x_9x_{10}x_{11}x_{12}x_{13}x_{14}x_{15}x_{16}x_{17}x_{18}x_{19}x_{20}x_{21}x_{22}x_{23}x_{24}x_{25}$

- 这种“交叉置乱”的交织方法，在发端和收端各需要**25**个存储单元将数据 “缓存”，以等待交织处理，这样一来就使通信发生了**50**个节拍的延时。

- 经改进得到的“卷积交织器”将交织器对角拆开，分别置于发端和收端，从而存储单元减少一半以上，延时也缩短一半以上。

卷积交织器

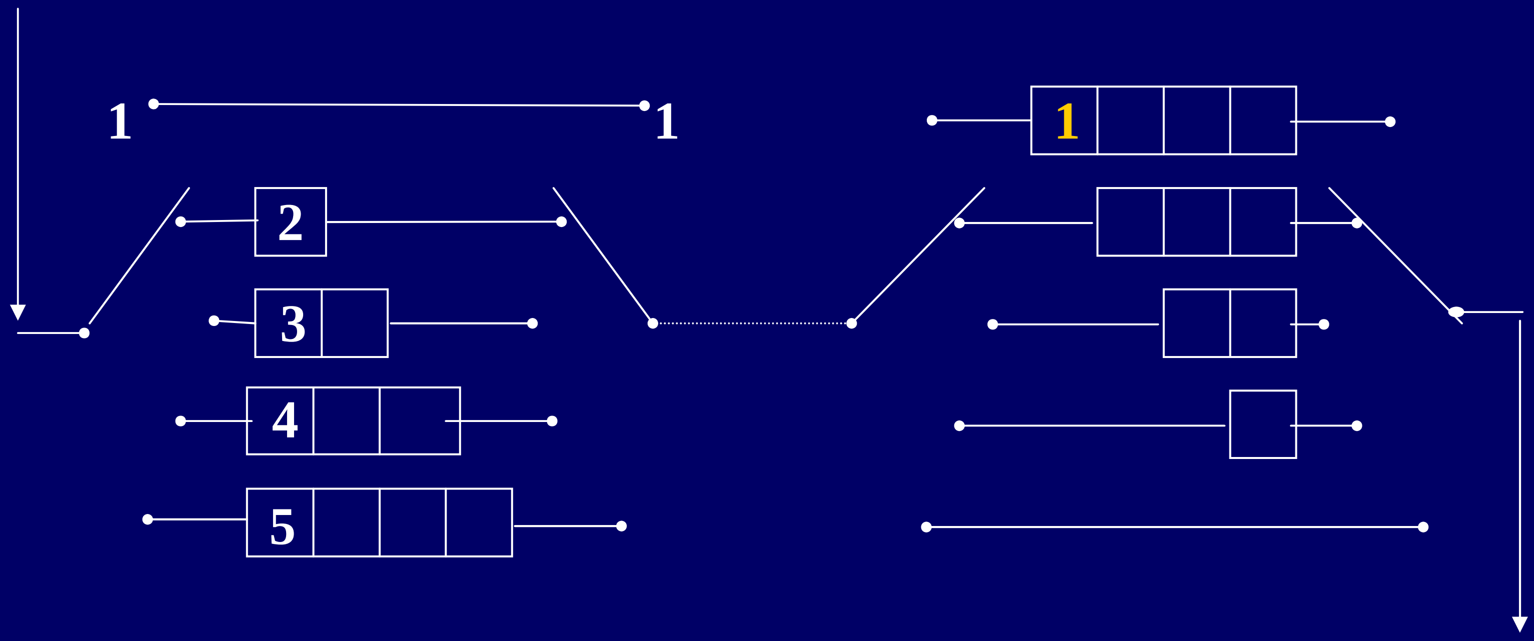
$x_1x_2x_3x_4x_5x_6x_7x_8x_9x_{10}x_{11}x_{12}x_{13}x_{14}x_{15}x_{16}x_{17}x_{18}x_{19}x_{20}x_{21}x_{22}x_{23}x_{24}x_{25}$



信道输出：x_1 □ □ □ □

卷积交织器

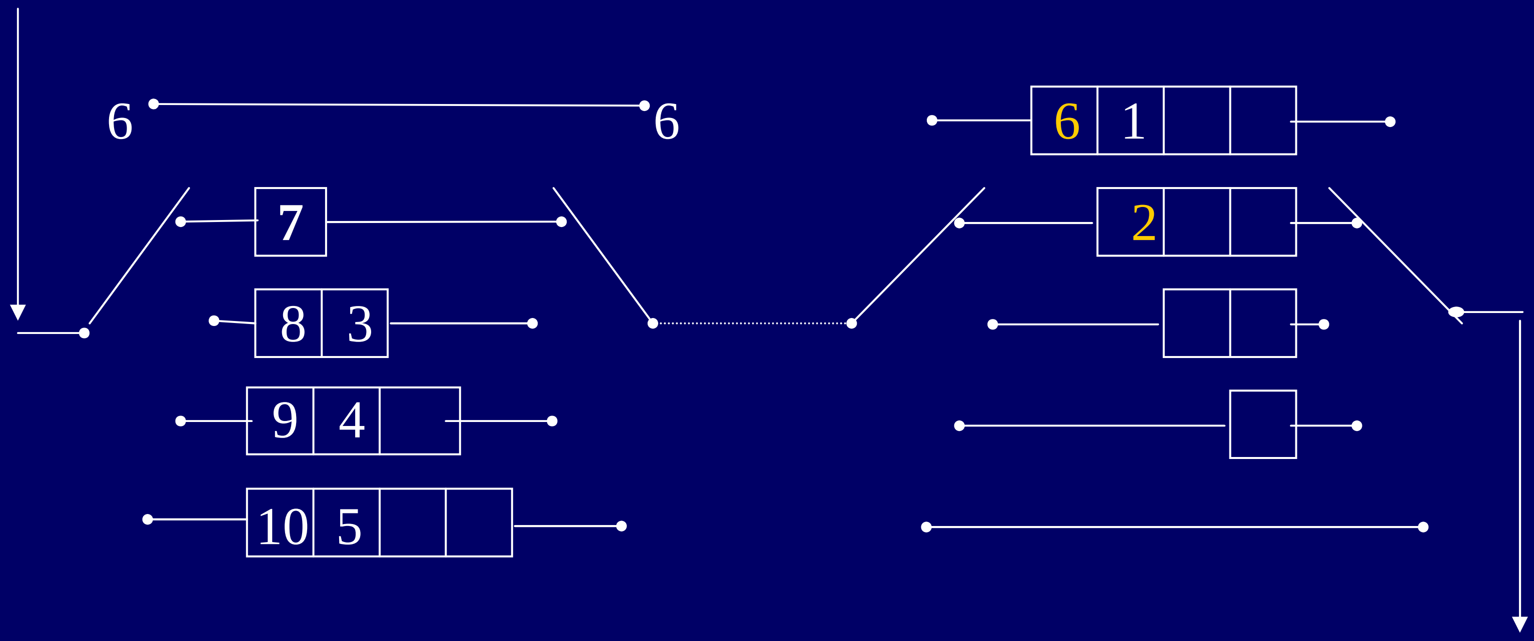
$x_1x_2x_3x_4x_5x_6x_7x_8x_9x_{10}x_{11}x_{12}x_{13}x_{14}x_{15}x_{16}x_{17}x_{18}x_{19}x_{20}x_{21}x_{22}x_{23}x_{24}x_{25}$



第1轮信道输出：x_1□□□□

卷积交织器

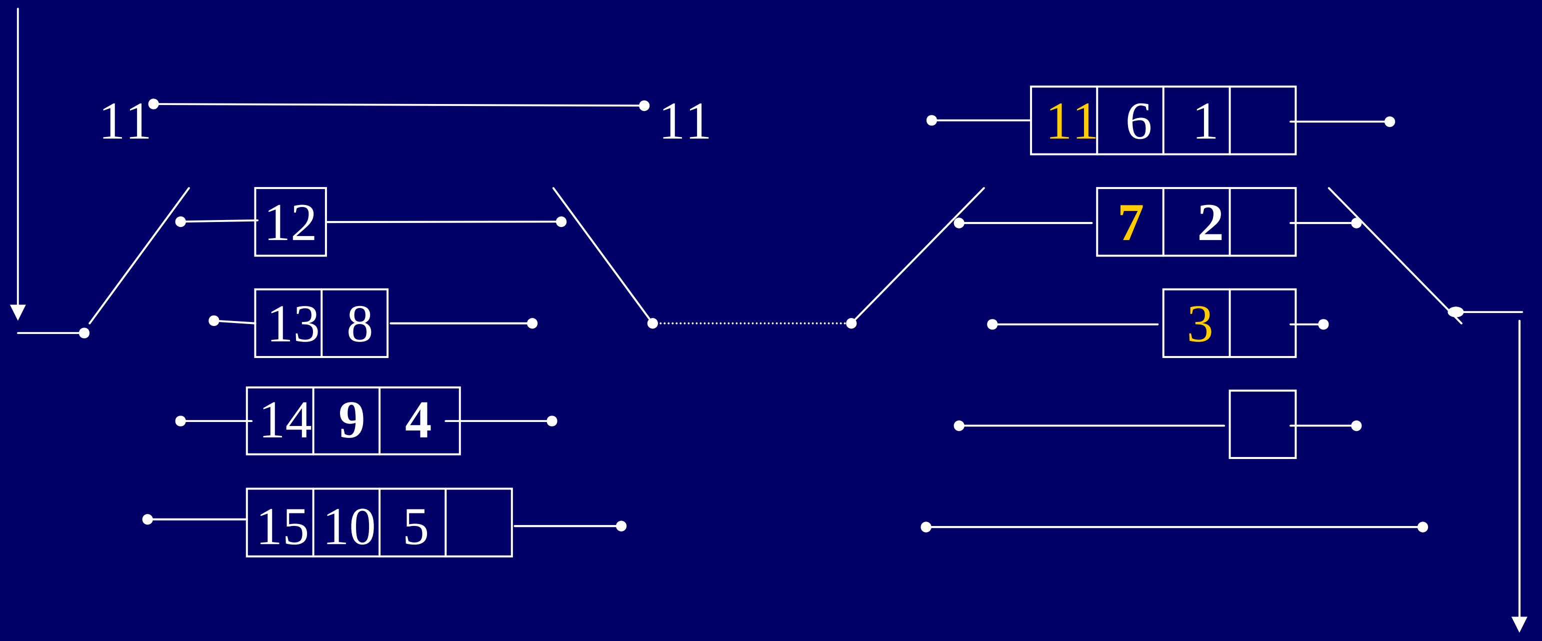
$x_1x_2x_3x_4x_5x_6x_7x_8x_9x_{10}x_{11}x_{12}x_{13}x_{14}x_{15}x_{16}x_{17}x_{18}x_{19}x_{20}x_{21}x_{22}x_{23}x_{24}x_{25}$



第2轮信道输出：x_1□□□□x_6x_2□□□

卷积交织器

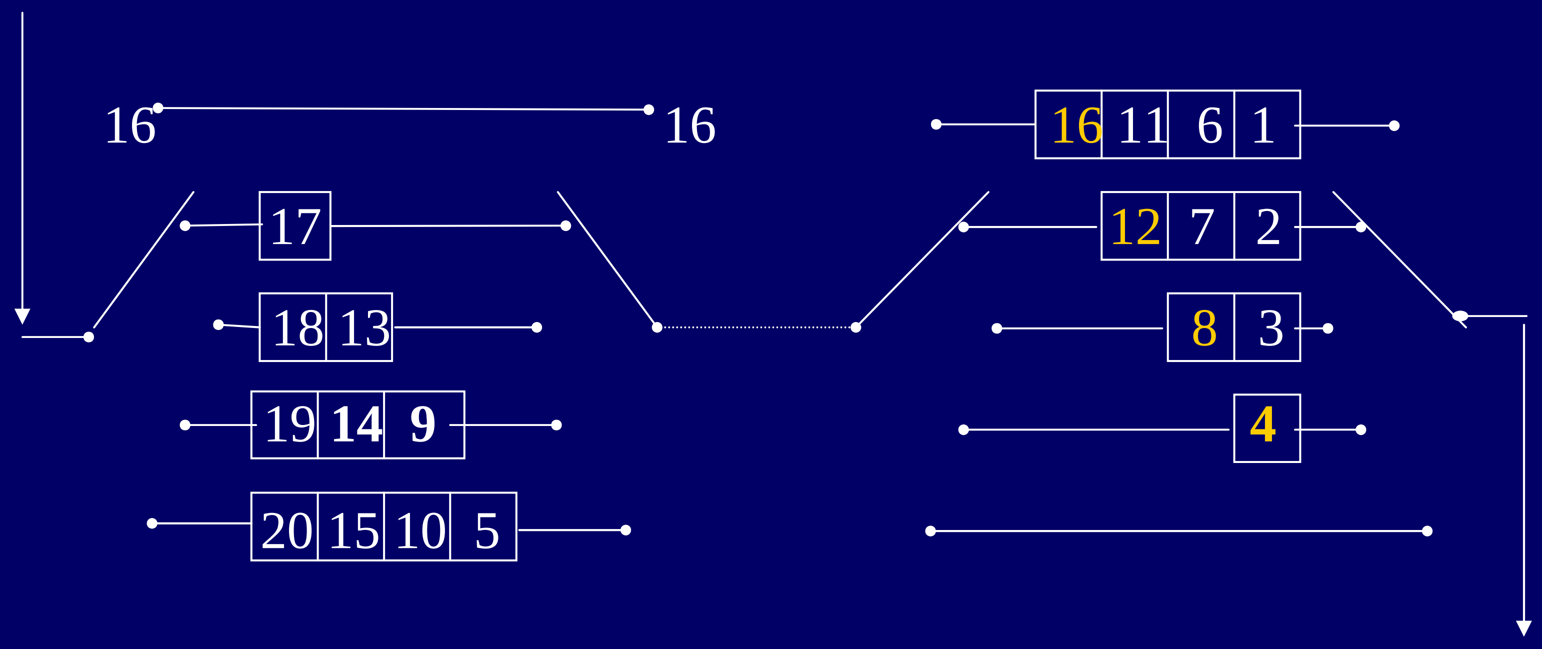
$x_1x_2x_3x_4x_5x_6x_7x_8x_9x_{10}x_{11}x_{12}x_{13}x_{14}x_{15}x_{16}x_{17}x_{18}x_{19}x_{20}x_{21}x_{22}x_{23}x_{24}x_{25}$



第3轮信道输出：x_1□□□□x_6x_2□□□$x_{11}x_7x_3$□□

卷积交织器

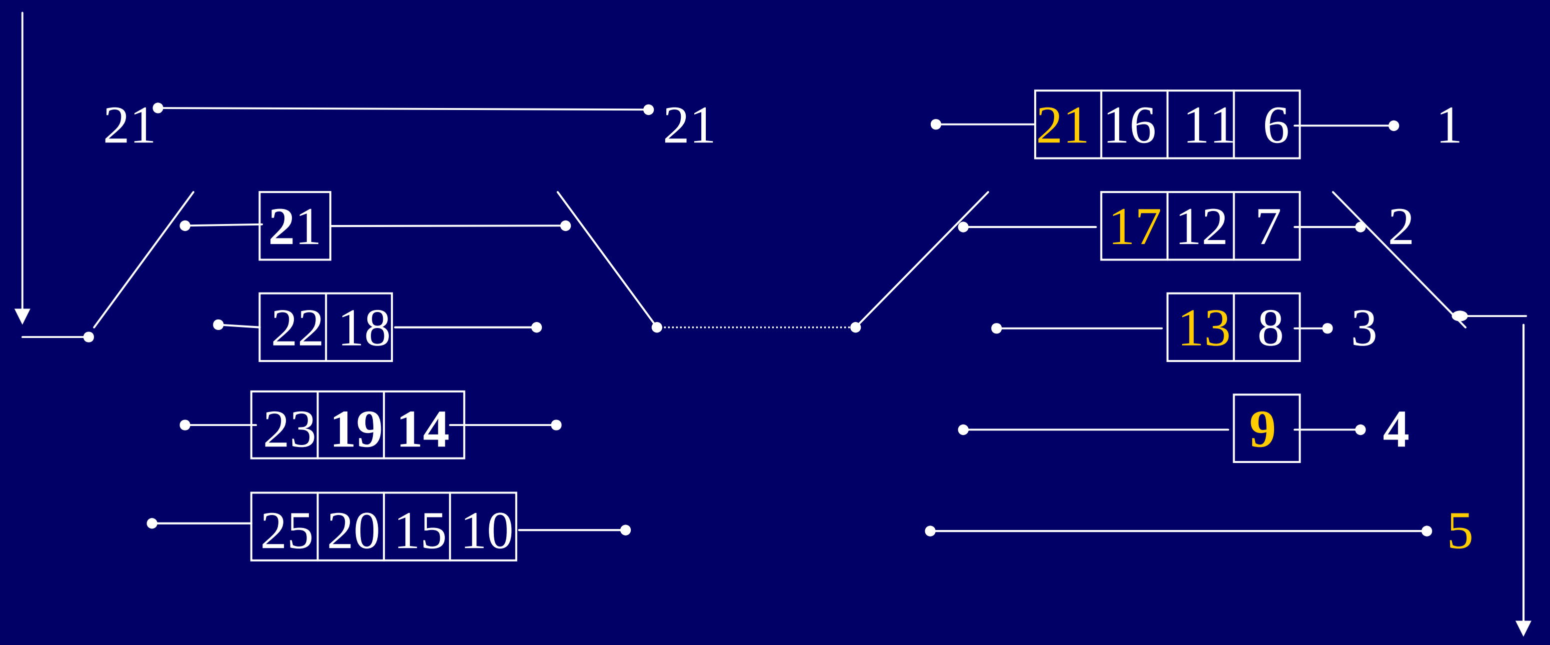
$x_1x_2x_3x_4x_5x_6x_7x_8x_9x_{10}x_{11}x_{12}x_{13}x_{14}x_{15}x_{16}x_{17}x_{18}x_{19}x_{20}x_{21}x_{22}x_{23}x_{24}x_{25}$



第4轮信道输出：x_1□□□□x_6x_2□□□$x_{11}x_7x_3$□□$x_{16}x_{12}x_8x_4$□

卷积交织器

$x_1x_2x_3x_4x_5x_6x_7x_8x_9x_{10}x_{11}x_{12}x_{13}x_{14}x_{15}x_{16}x_{17}x_{18}x_{19}x_{20}x_{21}x_{22}x_{23}x_{24}x_{25}$

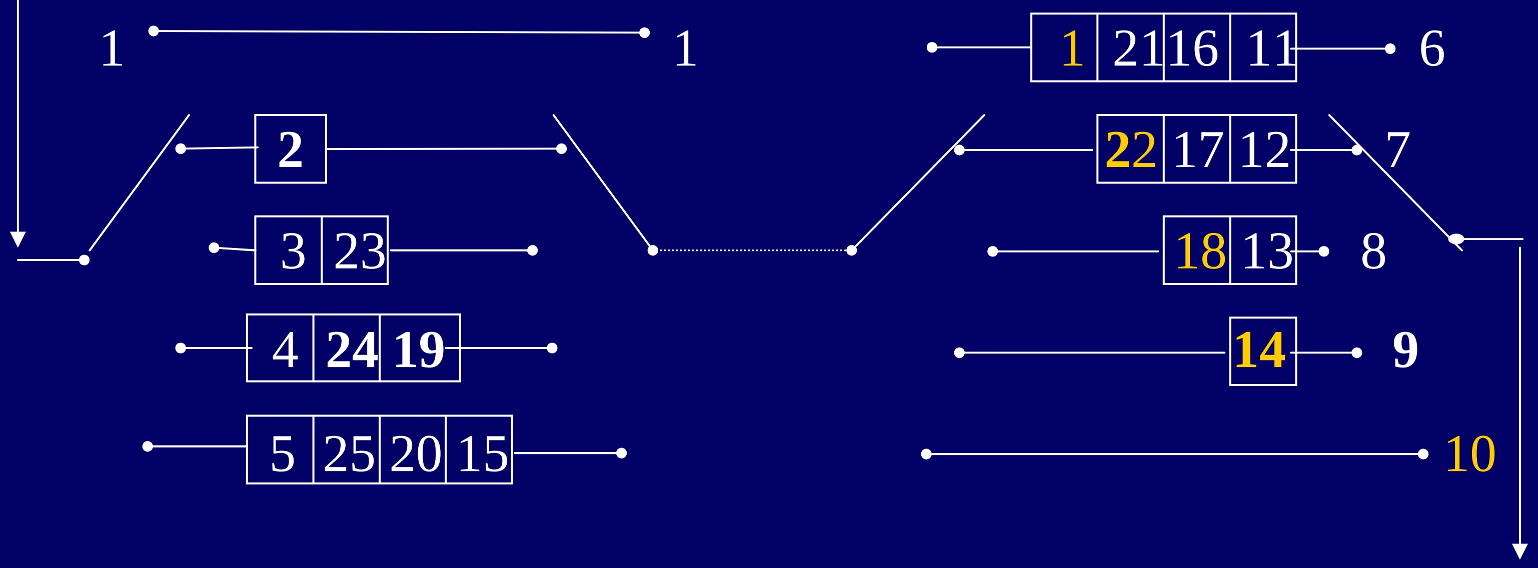


第5轮信道输出：x_1□□□□x_6x_2□□□$x_{11}x_7x_3$□□$x_{16}x_{12}x_8x_4$□

$x_{21}x_{17}x_{13}x_9x_5$

卷积交织器

$x_1x_2x_3x_4x_5x_6x_7x_8x_9x_{10}x_{11}x_{12}x_{13}x_{14}x_{15}x_{16}x_{17}x_{18}x_{19}x_{20}x_{21}x_{22}x_{23}x_{24}x_{25}$



第6轮信道输出： x_1□□□□x_6x_2□□□$x_{11}x_7x_3$□□$x_{16}x_{12}x_8x_4$□

$x_{21}x_{17}x_{13}x_9x_5x_1x_{22}x_{18}x_{14}x_{10}$

卷积交织器对突发错误的分散情况：

- 经过卷积交织后，数据序列按照依次提前**4** 个位置的规律重新排列，称之为“交插置乱”。

$X_1X_{22}X_{18}X_{14}X_{10}X_6X_2X_{23}X_{19}X_{15}X_{11}X_7X_3X_{24}X_{20}X_{16}X_{12}X_8X_4X_{25}$
$X_{21}X_{17}X_{13}X_9X_5$

- 如果信道中发生**2**处突发错误：

$X_1X_{22}X_{18}X_{14}X_{10}X_6X_2X_{23}X_{19}X_{15}X_{11}X_7X_3X_{24}X_{20}X_{16}X_{12}X_8X_4X_{25}$
$X_{21}X_{17}X_{13}X_9X_5$

- 那么经过解交织后，序列恢复原来次序，但错误码元已被分散开来：

$X_1X_2X_3X_4X_5X_6X_7X_8X_9X_{10}X_{11}X_{12}X_{13}X_{14}X_{15}X_{16}X_{17}X_{18}X_{19}X_{20}$
$X_{21}X_{22}X_{23}X_{24}X_{25}$

3.7.2 Fire码

（1）纠错位数与突发长度的关系：

- **BCH**码是能够纠正多个错位的循环码。
- 纠正t位错误，本身就包括t位连在一起的情况。也就是说它必能纠正一个突发长度 $b=t$ 位的突发错误。
- 然而突发长度为b，只不过表明这b位已经不可信，存在随机错误，但是未必b位全错，所以一般 $b \leqslant t$
- 如果总是b位全错，倒好办了：只要找到错误位置，把b位统统求反即可。

- 理论上目前还得不到 ***t***与***b***的简单关系，只能给出一个范围：

$$d-2 \leqslant \boldsymbol{b} \leqslant (n-\mathrm{k})/2$$

式中d是最小码距， n-k分别是码长与信息位长度。

- 理论上还限制了$\boldsymbol{p}=(\boldsymbol{d}-1)/2$个突发错误的总长度不大于 $3d-4p-4$ 与 $(d-1)/2$ 之中的较大者。
- 就是说，若只纠正一个突发错误，则突发长度可以长一些；若要纠正多个突发错误，则每个都比较短一些，以保持总突发长度基本不变。

（2）弗尔(Fire)码概述：

- 弗尔(**Fire**)码是用分析方法构造出来的能够纠正单个突发错误的二进制循环码。
- **Fire**码是循环码的推广，编译码方法与循环码相同。
- **Fire**码的生成多项式是：$g(x)=(x^{2b-1}+1)\cdot p(x)$

  式中是b突发长度，$p(x)$是m ($m \geq b$)次的即约多项式。可见$g(x)$的幂次是 $r=2b-1+m$
- 取码长n=LCM $[2^m-1, 2b-1]$，则得到（n，$n-r$) **Fire**码。它能纠正单个长度不大于b的突发错误。

[例1] 试构造一个能纠正五连突发错误的Fire码。

解：b=5，不妨也取m=5，于是2^m-1=31，2b-1=9；

取：$p(x)=x^5+x^2+1$

生成多项式为：$g(x) = (x^{2b-1}+1)\cdot p(x)= (x^9+1)\cdot(x^5+x^2+1)$

$= x^{14}+x^{11}+ x^9+x^5+ x^2+1$

码长：n=LCM $[2^m-1, 2b-1]$= n=LCM [31, 9]=279

信息位为：$k=n-r$ =279−14=265

应构造的循环码是：（279，265）码；

监督多项式为：$r(x) = x^{14}K(x) \bmod g(x)$；

码多项式为：$C(x) = x^{14}K(x)+r(x)$

用计算机搜索得到的一些Fire码

n	k	t	b_F	b_C	k/n	$g(x)$
15	6	2	3	4	0.4	23
21	8	3	4	6	0.381	127
33	12	5	6	9	0.364	3040
39	14	6	7	12	0.359	13617
45	18	7	8	12	0.4	10011

实际上，绝大多数Fire码实际所具有的纠错能力b_C远大于设计的纠错能力b_F。

3.7.3 RS码

（1）概述：

- RS是里德-索罗蒙（Reard-Solomen）的字头。
- RS码是q进制本原BCH码。码字$C=c_1c_2\ldots\ldots c_n$中每位码元c_i都是一个q进制符号。取$q=2^m$，纠正1位q进制符号相当于纠正了m位2进制符号，因此RS码自然被用来纠正突发错误。
- 严格讲，纠正t位q进制符号与纠正$m\cdot t$位2进制符号还是有区别的：$m\cdot t$位2进制符号可能会夸接在$t+1$个q进制符号上，必须纠正$t+1$位q进制符号才能保证任意$m\cdot t$位2进制符号正确。
- 因此RS码可纠正的二元符号突发长度是：$b\leqslant(t-1)m+1$

（2）*q*进制符号的选择：

- 例如，以8进制符号来表达3位二元码：

  (000, 001, 010, 011, 100, 101, 110, 111）；

- 以往，曾经借用十进制符号的前8个符号来表示它们：

  (0, 1, 2, 3, 4, 5, 6, 7);

- 现在，为了把2元BCH编码的理论方法推广到*q*元BCH码，必须用$x^7-1=0$的7个复数根$(e^{j2\pi/7})^i=\alpha^i$ $(i=0,1,2,\ldots,6)$ 加上0，即借助$GF(2^3)$域的8个域元素来表示它们：

  $(0, 1, \alpha, \alpha^2, \alpha^3, \alpha^4, \alpha^5, \alpha^6)$；

这是因为它们满足循环码所必须的循环移位运算法则。

- 结论：用$GF(2^m)$域的$q=2^m$个域元素作为*q*进制符号的表达。

（3）$GF(2^m)$域元素与二元码的对应关系

- 仍以$GF(2^3)$域为例。
- 设 α 是$x^7-1=0$的一个本原根，满足：

$$x^7-1=(x-1)(x-\alpha)(x-\alpha^2)(x-\alpha^3)(x-\alpha^4)(x-\alpha^5)(x-\alpha^6)$$

- 当然 α 也是本原多项式x^3+x+1的根，这是因为：

$$m_1(x)=(x-\alpha)(x-\alpha^2)(x-\alpha^4)=x^3+x+1$$

- α 既然满足 $\alpha^3+\alpha+1=0$，因此模 $\alpha^3+\alpha+1$ 运算下就有：

$$\alpha^3=\alpha+1; \quad \alpha^4=\alpha(\alpha+1)=\alpha^2+\alpha;$$

$$\alpha^5=\alpha^2(\alpha+1)=\alpha^3+\alpha^2=\alpha^2+\alpha+1;$$

$$\alpha^6=(\alpha+1)(\alpha+1)=\alpha^2+\alpha+\alpha+1=\alpha^2+1$$

借助码多项式的形式，建立了三位二元码与 $GF(2^3)$域元素的一一对应关系：

$GF(2^3)$域表达	码多项式	二元码
0	0	000
$\alpha^0 = 1$	1	001
$\alpha^1 = \alpha$	x	010
$\alpha^2 = \alpha^2$	x^2	100
$\alpha^3 = \alpha + 1$	$x+1$	011
$\alpha^4 = \alpha^2 + \alpha$	x^2+x	110
$\alpha^5 = \alpha^2 + \alpha + 1$	x^2+x+1	111
$\alpha^6 = \alpha^2 + 1$	x^2+1	101

值得注意，二元码的排列次序已不是按照自然数的顺序！

同理，四位二元码与$GF(2^4)$域16个元素的一一对应关系可借助本原多项式 $\alpha^4+\alpha+1=0$ 得到 ：

本原域元素	码多项式	二元码	本原域元素	码多项式	二元码
0	0	0000	$\alpha^7=\alpha^3+\alpha+1$	x^3+x+1	1011
$\alpha^0=\alpha^{15}=1$	1	0001	$\alpha^8=\alpha^2+1$	x^2+1	0101
$\alpha^1=\alpha$	x	0010	$\alpha^9=\alpha^3+\alpha$	x^3+x	1010
$\alpha^2=\alpha^2$	x^2	0100	$\alpha^{10}=\alpha^2+\alpha+1$	x^2+x+1	0111
$\alpha^3=\alpha^3$	x^3	1000	$\alpha^{11}=\alpha^3+\alpha^2+\alpha$	x^3+x^2+x	1110
$\alpha^4=\alpha+1$	$x+1$	0011	$\alpha^{12}=\alpha^3+\alpha^2+\alpha+1$	x^3+x^2+x+1	1111
$\alpha^5=\alpha^2+\alpha$	x^2+x	0110	$\alpha^{13}=\alpha^3+\alpha^2+1$	x^3+x^2+1	1101
$\alpha^6=\alpha^3+\alpha^2$	x^3+x^2	1100	$\alpha^{14}=\alpha^3+1$	x^3+1	1001

（4）$GF(2^m)$域元素的运算法则

❖$GF(2^m)$域的运算法则有三条：

①位运算遵循模2加法则：如与1011 ⊕ 1110=0101对应的是：

$\alpha^7+\alpha^{11}=(\alpha^3+\alpha+1)+(\alpha^3+\alpha^2+\alpha)=\alpha^2+1=\alpha^8$

(加法运算以多项式表达进行比较方便)

②乘运算（循环移位）遵循模 $\alpha^{q-1}=1$运算法则：（$q=2^m$）

$(\alpha^3+\alpha+1)\cdot(\alpha^3+\alpha^2+\alpha)=\alpha^7\cdot\alpha^{11}=\alpha^{18}=\alpha^3$;

所以1011⊙1110=1000

(乘法运算以本原根形式进行比较方便)

③随时通过模$P(x)$运算实现幂次变换。

例如 $\alpha^{11}=\alpha^{11}\bmod(\alpha^4+\alpha+1)=\alpha^3+\alpha+1$ (商是 $\alpha^7+\alpha^4+\alpha^3+\alpha$)

（5）RS码的编码

- 码长为n的q 进制码也可以表达为码多项式：

$$C(x)=c_{n-1}x^{n-1}+c_{n-2}x^{n-2}+\dots\dots+c_2x^2+c_1x+c_0$$

- 现在c_i不是0或1，而是一个q 进制符号。
- 信息元为k位q进制符号，监督元为$r=n-k$位q进制符号时，相应的码多项式分别为：

$$K(x)=m_{k-1}x^{k-1}+m_{k-2}x^{k-2}+\dots\dots+m_2x^2+m_1x+m_0$$

$$r(x)=r_{r-1}x^{r-1}+r_{r-2}x^{r-2}+\dots\dots+r_2x^2+r_1x+r_0$$

- 它们合起来构成系统码，并满足关系式：

$$C(x)=x^rK(X)+r(x)=g(x)Q(x);$$

式中$g(x)$是生成多项式，$Q(x)$是$g(x)$整除$C(x)$所得的商式。

- RS码的生成多项式$g(x)$仍然应从多项式x^n-1中分解出一个r次的因式得到。
- 在$GF(2^m)$域中，x^n-1可分解为：

$$x^n-1=(x-1)(x-\alpha)(x-\alpha^2)\cdots\cdots(x-\alpha^{n-1})=\prod_{i=0}^{n-1}(x-\alpha^i)$$

- 若要求纠错符号数目为t，则生成多项式$g(x)$应由2 t个这样的一次因式相乘。
- 一般取：　$g(x)=(x-\alpha^0)(x-\alpha^1)\ \ldots\ldots(x-\alpha^{2t-1})$

或：　$g(x)=(x-\alpha)(x-\alpha^2)\ \ldots\ldots(x-\alpha^{2t})$;

(共2t个一次因子相乘, 表明$g(x)$的幂次为$r=2t$)

[例2]已知信息位15比特（**001 101 011 100 001**），试构造一个能纠正三连突发错误的RS码。

解：采用$GF(2^3)$域的RS码。纠正3位二进制符号突发错误相当于纠正t=1位q=8进制符号，15比特信息可作为5个八进制符号，监督位长度$r=2t=2$，所以应构造(7, 5) RS码。

生成多项式为：$g(x) = (x-\alpha^0)(x-\alpha^1)$

信息多式为：$K(x) = m_4x^4+m_3x^3+m_2x^2+m_1x+m_0$；

监督多项式为：$r(x) = r_1x+r_0$；

则码多项式为：$C(x) = x^2K(x)+r(x)$；

设商式为$Q(x)$, 则：$x^2K(x)+r(x)=Q(x)g(x)$；

即：$m_4x^6+m_3x^5+m_2x^4+m_1x^3+m_0x^2+r_1x+r_0=(x-1)(x-\alpha)Q(x)$；

令$x=1$得：　$m_4+m_3+m_2+m_1+m_0+r_1+r_0=0$ ---------------------- ①

令$x=\alpha$得：$m_4\alpha^6+m_3\alpha^5+m_2\alpha^4+m_1\alpha^3+m_0\alpha^2+r_1\alpha+r_0=0$

---------------------- ②

以$GF(2^3)$的本原多项式$\alpha^3+\alpha+1$为模，并注意$\alpha^7=1$

联立即可解得：　$r_1=m_4\alpha^6+m_3\alpha+m_2\alpha^2+m_1\alpha^5+m_0\alpha^3$；

$r_0=m_4\alpha^2+m_3\alpha^3+m_2\alpha^6+m_1\alpha^4+m_0\alpha$；

已知：$m_4=001=1$，$m_3=101=\alpha^2+1=\alpha^6$，$m_2=011=\alpha+1=\alpha^3$，

$m_1=100=\alpha^2$，$m_0=001=1$；

则求得：　$r_1=1=001$，$r_0=\alpha+1=011$；

于是所编的RS码字为：$C=(001,101,011,100,001,001,011)$；

（6）RS码的译码

- 仍以[例2]的（7, 5）RS码来讨论。
- 根据①②两式：

$$0 = m_4 + m_3 + m_2 + m_1 + m_0 + r_1 + r_0;$$

$$0 = m_4\alpha^6 + m_3\alpha^5 + m_2\alpha^4 + m_1\alpha^3 + m_0\alpha^2 + r_1\alpha + r_0$$

当收到的码字为 $R=(R_6, R_5, R_4, R_3, R_2, R_1, R_0)$时，

通过与发送码 $C=(m_4, m_3, m_2, m_1, m_0, r_1, r_0)$ 相比较，

可令校验子s_0和 s_1为：

$$s_0 = R_6 + R_5 + R_4 + R_3 + R_2 + R_1 + R_0;$$

$$s_1 = R_6\alpha^6 + R_5\alpha^5 + R_4\alpha^4 + R_3\alpha^3 + R_2\alpha^2 + R_1\alpha + R_0$$

❖无错时应当 $s_0=s_1=0$;

❖当发生错误时：$s_0 \neq 0$;　$s_1 \neq 0$;

❖本例设计只纠1位q 进制码的错，假设错发生在R_i位, 因为

$R_i=C_i \oplus E_i$，C_i是正确码，E_i是误差值，正确码使:

$$C_6+C_5+C_4+C_3+C_2+C_1+C_0=0;$$

$$C_6\alpha^6+C_5\alpha^5+C_4\alpha^4+C_3\alpha^3+C_2\alpha^2+C_1\alpha+C_0=0;$$

因此，s_0 和s_1 不等于0 的部分完全来自E_i。结论是:

（1）$s_0=E_i$;　s_0给出错位的误差值:

（2）$s_1=E_i\alpha^i$;　再加上s_1就可以给出错误所在的位置:

- 例如收到码字是R=(001,101,111,100,001,001,011)；
- 本原元素幂形式为：$R=(\alpha^0, \alpha^6, \alpha^5, \alpha^2, \alpha^0, \alpha^0, \alpha^3)$；
- 首先按位模二加算出 $s_0=100=\alpha^2$；
- 设错误发生在R_i位，则非零的s_0就等于R_i的误差值E_i

  即 $E_i=s_0=\alpha^2$。
- 利用域运算法则，代入R值可以计算出s_1：

  $s_1=\alpha^6+\alpha^{11}+\alpha^9+\alpha^5+\alpha^2+\alpha+\alpha^3=\alpha^6$;
- 因为错误是在R_i位，所以非零的s_1就等于该位的误差值，

  $E_i\alpha^i=s_1$；即 $\alpha^2\cdot\alpha^i=\alpha^6$，由此便知 $i=4$；
- 结论是错在α^4位；其系数R_4=111是错的，应纠正为

  $R_4\oplus E_4=111\oplus 100=011$；

（7）RS码在CD光盘中的应用

- ISO/IEC10149标准规定，CD-ROM扇区中采用GF(2^8)域上的RS码进行差错控制。
- 数据以字节为单位编码，即8*bit*作为一个符号看待，每个扇区中第12—2075字节为信息数据块，共2064个符号，排列成两个24×43的矩阵。
- 首先每列进行（26,24）的RS编码(实际上是（31,29）的截短循环码)，每24个符号添加两个监督符号，构成码长26的码字，称为*P* 校验。43列共找到172个字节的*P* 校验符号。
- 然后，对矩阵行、列进行交织变换。

❖变换后再对每一列进行（45, 43）的RS编码（实际上是（63, 61）的截短循环码），每43个符号添加两个监督符号，构成码长45 的码字，称为Q 校验。26列共找到104个字节的Q 校验符号。

❖最后，2064个字节的信息与276个字节的校验构成两个45行26列的阵列。

❖经过交织编码，突发性字节错误被分散开来。无论P 校验码还是Q 校验码，校验字节都是r=2，均可纠错 t=1字节的错误符号，即8位连续二元错位。

❖经这种双向RS交织校验，字节误码率可小于10^{-13}，完全满足了计算机读写数据的要求。

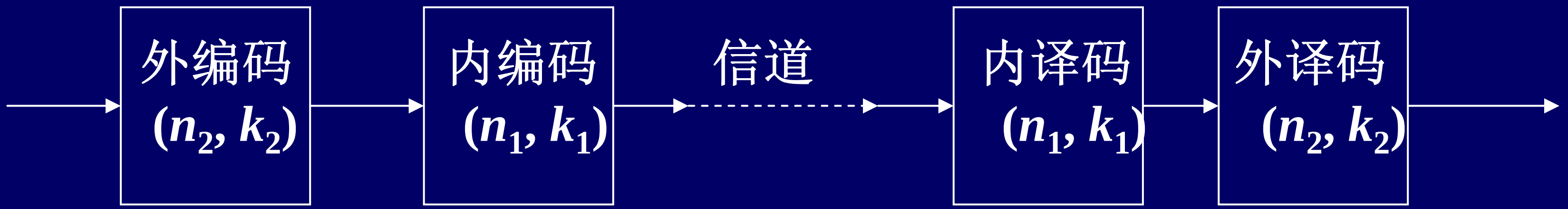
- 如果错误在二维阵列上完全分散开来，可以分别算出各行和各列的校验子，校验子为零的行或列都是无错的行或列。
- 然而有时错误在二维阵列上不能完全分散开来，我们可以先对只有一个错误字节的行进行纠错，这样就会使一些本来有两个或更多字节错误的列变成单个错误，从而在随后按列进行的纠错中得以纠正。
- 反复交替地进行行纠错和列纠错，最后如果还剩下一些尚未纠正的错误，那么它们一定是分布在“井”字形的交叉点上。
- 由于已经知道了非零校验子所在交叉点的行列序号，就等于已经知道了这些错误字节所在的位置。在此情况下，联立求解校验子的方程组，双字节错误也能得到纠正。

3.7.4 级连码

（1）原理：

- 实际信道中出现的差错，往往既不是单纯的独立差错，也不全是突发错误，而是兼而有之。因此需要将纠正孤立差错的编码与纠正突发错误的编码结合使用。
- 采用二元线性分组码(n_1, k_1)为内编码，用非二元的(n_2, k_2)码为外编码，内编码解决孤立差错校验，外编码负责突发错误纠正。
- 长码一般具有较强的纠错能力，但需要较复杂的编码和译码设备。经内、外两种编码级连，总码长达n_1n_2，总信息为k_1k_2。因此，级连码是一种由短码构造长码的有效方法。

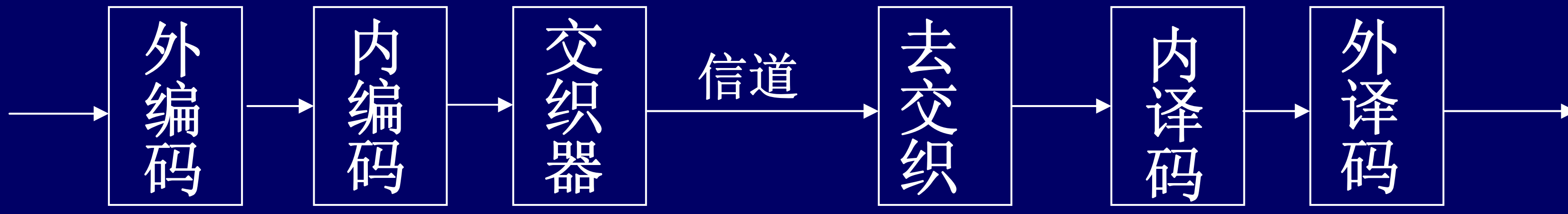
（2）流程与方案



- **1984**年，美国**NASA**采用（**2,1,7**）卷积码为内码，（**255, 223**）**RS**码为外码，加上交织编码，设计出用于空间飞行器数据网的编码方案，后来成为级连码系统的技术标准。依此为基础，以后又设计出若干性质优良的级连码。
- 若干性质优良的级连码的编码方案

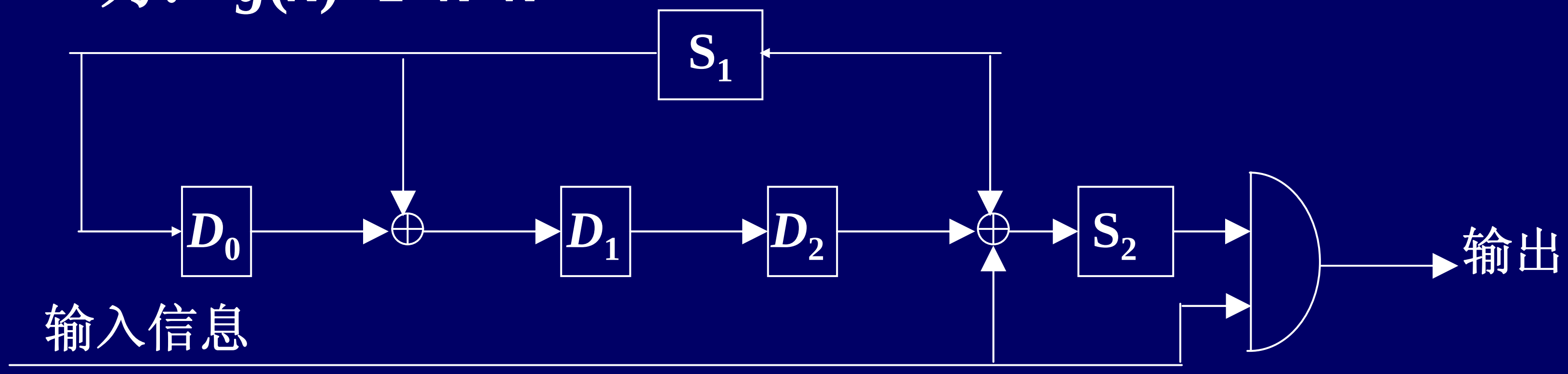
内码	**(4,1,5)**	**(5,1,14)**	**(6,1,14)**	**(6,1,14)**	**(6,1,15)**	**(5,1,13)**
外码	**(255,223)**	**(255,231)**	**(255,233)**	**(1023,959)**	**(1023,959)**	**(255.229)**

（3）级连码在GSM移动通信中的应用



①外编码采用循环码：

- 首先将待编数据分成每20毫秒260*bit*为一帧，其中前50*bit*，进行(53, 50)的截短循环码编码，其生成多项式为：$g(x)=1+x+x^3$



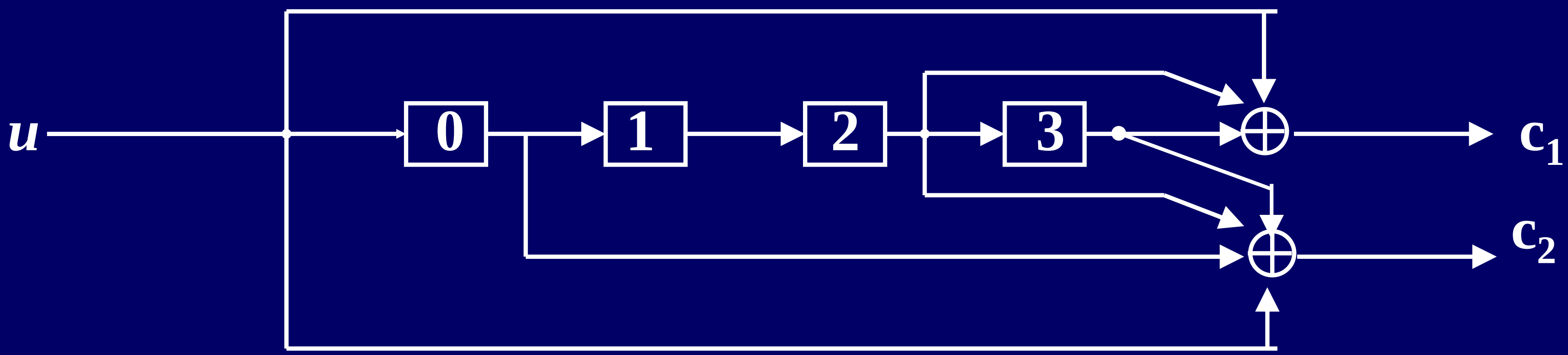
- 编码结果是在50*bit*信息后添加3个奇偶校验位**$p(0)$**、**$p(1)$**和**$p(2)$**；
- 接着将280比特中的前182*bit*，3个奇偶校验位，再添入4个0，共189*bit*进行重排。结果排成:

u=[$d(0)$, $d(2)$, … (原偶序列)…, $d(180)$, $p(0)$, $p(1)$, $p(2)$, $d(181)$, $d(179)$, … (原奇序列)…, $d(3)$, $d(1)$, 0, 0, 0, 0]

②内编码采用卷积码:

- 采用(2, 1, 4)卷积码, 使码率加倍。
- 转移函数矩阵为：$G(D) =(1+D^3+D^4 , 1+D+D^3+D^4)$

- (2, 1, 4)卷积码的电路为：



- 寄存器初态为（0，0，0，0）

  输入的189*bit*为：u =[u (0)，u (1)，……,u (188)]；

  输出两路分别为 c_1 =[c(0)，c (2) ……,c(376)]

  和 c_2 =[c (1)，c (3) ……,c(377)]；

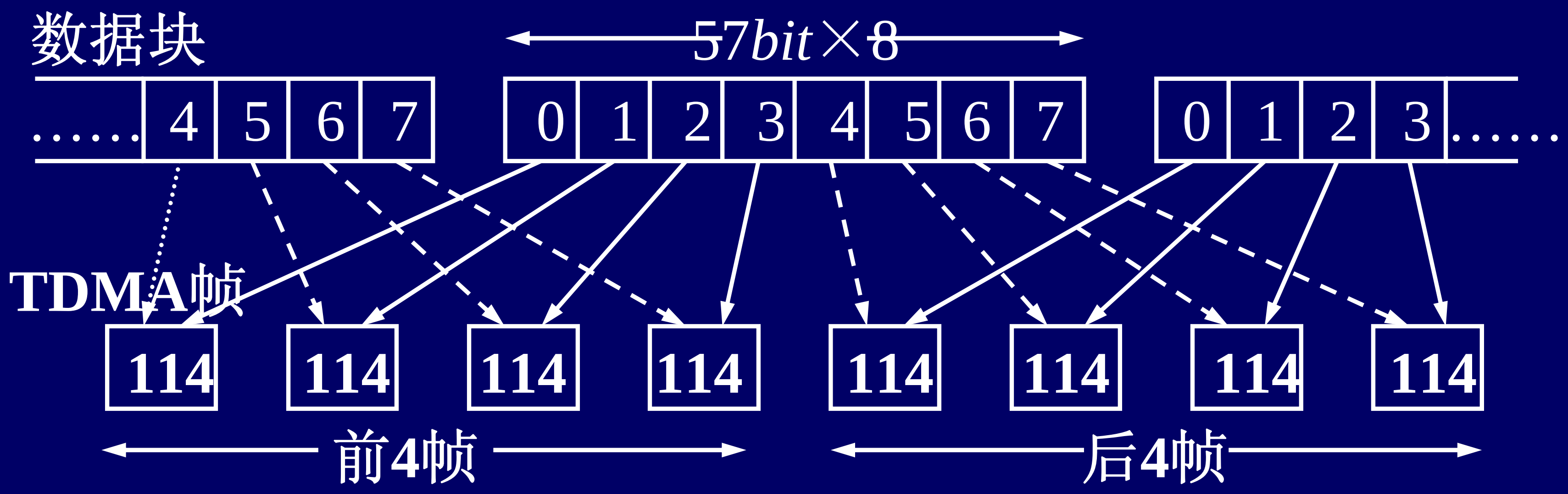
- 穿插合并成为378*bit*，最后接上剩下的78*bit*成为：

  c=[c (0)，c (1)，……,c(377)，……,c(455)]；

- 至此，每帧260*bit*变为456*bit*，码率由13*kb/s*变为22.8*kb/s*；

③重排交织：

- 每帧456*bit*按顺序分成8个数据子块，记做*x*=0, 1, 2, ……7；每块中的数据位记做*y*=0, 1, 2, ……, 56（共57位）。
- GSM采用时分多址(TDMA)复用技术。每个TDMA帧114*bit*。
- 8个子块中的数据被分配到8个TDMA帧中，前4个子块（*x*=0, 1, 2, 3）中的228bit放入前4个TDMA帧的偶数位，后4个子块（*x*=4, 5, 6, 7）中的228*bit*放入后4个TDMA帧的奇数位。



- 与此同时，前4个TDMA帧的奇数位中还接收了前面更早数据块中的数据，后4个TDMA帧的偶数位则还接收了下一组数据块中的数据。
- 将TDMA帧依次送入交织器，被交织的数据显然是来自两个相邻456*bit*块的数据。
- 图中实线为偶数位数据流向，虚线为奇数位数据流向。

本节小结

- 交织码
- RS码
- Fire码
- 级连码
- CD-ROM中的应用
- GSM移动通信中的应用

- 思考题：

  1. 交织码的内编码与外编码是怎样配合工作的？

  2. 构造检、纠错编码所追求的目标是什么？

- 作业题：

  阅读3.8节:信道编码的新进展。