第5章 卷积码

分组码在编码时，先将输入信息码元序列分为长度为 k 的段，然后按照编码规则，给每段附加上 r 位监督码元，构成长度为 n 的码组。各个码组间没有约束关系，即监督码元只监督本码组的码元有无错码。因此在解码时各个接收码组也是独立地进行解码的。从信息论角度而言，一方面信息流分割为独立码块而不能利用组间的相关信息，且编码定理表明分组码的码长 n 越长越好，另一方面译码运算量却随着码长 n 的增加而增加。

为了解决上述矛盾，提出了另外一种编码，称为卷积码。卷积码的特点是信息进行编码时，信息组之间不是独立编码的，而是具有一定的相关性，系统译码时可以利用这种相关性进行译码。

卷积码是一种非常重要的差错控制编码。卷积码编码时虽然也是将 k 比特的信息段编成 n 比特的码组，但是监督码元不仅和当前的信息段有关，而且与前面若干时刻输入至编码器的信息段有关。同样，卷积码译码时也要结合当前时刻和以前各时刻接收到的码段来提取有关信息。由于卷积码充分利用了前后码段之间的相关性，故与分组码比较，卷积码的性能更好，译码更容易。

5.1 卷积码的基本概念

卷积码编码时，首先将信息序列划分为长度为 k 的组，当前时刻编码输出不仅取决于当前输入的信息组，而且与前若干时刻的信息组有关。为了表示这种关联性，卷积码一般表示为(n,k,m)，其中，k 为信息组的长度，n 表示每组信息对应输出的码长度，m 表示与此前输入的 m 个信息关联，$N=m+1$ 称为信息组约束长度。与分组码一样，(n,k,m)卷积码的码率为 $R=k/n$。

卷积码编码器的一般框图如图 5-1 所示。输入的信息序列被分成长度为 k 的段，并经过串/并转换输入到离散线性系统的 k 个输入端。该系统的输出端为 n 个(一般 $n>k$)，且系统最大延时为 m。输出的 n 个编码数字经过并/串转换送入信道就完成了编码过程，这就是可表示为(n,k,m)码的典型卷积码。一般选 n 和 k 较小，但 m 值较大(m 为 10 左右)。

卷积码的纠错性能随 m 的增大而增大，而复杂度随 m 的增大呈指数增加。在编码器复杂性相当的情况下，卷积码的性能优于分组码。但卷积码没有分组码那样严密的数学分析手段，目前大多通过计算机进行好码的搜索。

卷积码编码器是一个由 k 个输入端、n 个输出端及 m 个移位寄存器所构成的有限状态

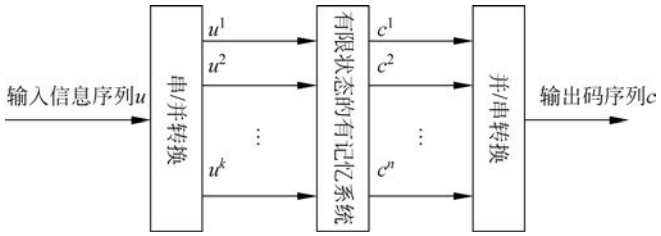


图 5-1 卷积码编码器原理图

有记忆系统。

描述卷积码的方法很多，大致可划分为两大类：解析法和图形法。解析法包括离散卷积码法、矩阵生成法及码多项式法等，主要用于编码部分的描述；图形法包括状态图法、树图法及网格法等，主要用于译码的描述和说明。

5.2 卷积码的编码过程

图 5-2 给出了一个二进制卷积码的编码器。若每一时间单位输入编码器一个新的信息元 u_i，且存储器内的数据往右移一位，则 u_i 一方面直接输出至信道，另一方面与前两个单位时间送入的信息元 u_{i-1} 和 u_{i-2} 按图 5-2 中线路所确定的规则进行运算，得到此时刻的两个输出(校验元)c_i^1 和 c_i^2，跟随在 u_i 后面组成一个子码 $\boldsymbol{c}_i=(u_i c_i^1 c_i^2)$ 送入信道。由图 5-2 可知

$$\begin{cases} c_i^1 = u_i \oplus u_{i-1} \\ c_i^2 = u_i \oplus u_{i-2} \end{cases} \tag{5-1}$$

下一个时间单位输入的信息元为 u_{i+1}，其相应的两个输出(校验元)

$$\begin{cases} c_{i+1}^1 = u_{i+1} \oplus u_i \\ c_{i+1}^2 = u_{i+1} \oplus u_{i-1} \end{cases}$$

组成第二个子码 $\boldsymbol{c}_{i+1}=(u_{i+1}c_{i+1}^1 c_{i+1}^2)$ 送至信道，如此循环。在每一时间单位，送入编码器 k(这里为 1)个信息元，编码器就送出相应的 n(这里为 3)个码元组成一个子码 $\boldsymbol{c}_i$ 送入信道，在卷积码中，这 n 个码元组成的子码 $\boldsymbol{c}_i$ 有时也称卷积码的一个码段或子组。

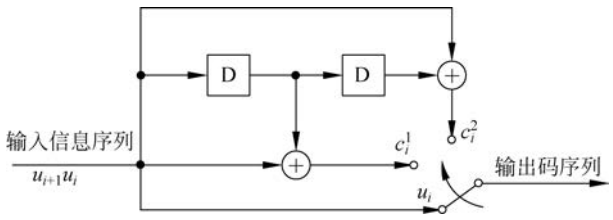


图 5-2 (3,1,2)卷积码编码器

由式(5-1)和图 5-2 可知，用这种卷积码编码器输出的每一子码中的校验元，是此时刻

输入的信息元与前 m（这里为 2）个子码中信息元的模 2 加，它们是线性关系，所以由这类编码器编出的卷积码是线性码。

当 $m=0$ 时，卷积码就可以被看作一个分组码，此时编码系统就是一个无记忆系统。

下面用具体的例子来说明二元域上卷积码的编码过程。

例 5-1 如图 5-3 给出一个(2,1,3)卷积码编码器结构，此时 $n=2$，$k=1$，$m=3$，编码速率 $R=k/n=1/2$，约束长度 $N=m+1=4$。求该卷积码编码器的输出。

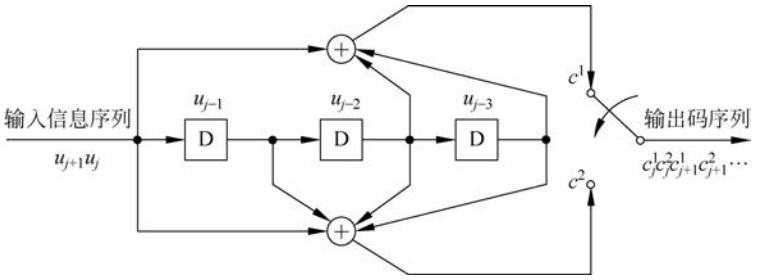


图 5-3 (2,1,3)卷积码编码器结构

卷积码编码时，在信息输入的同时，移位寄存器组进行信息移位，原来最低位置的信息移往下一个寄存器组，最后一个寄存器组的信息移出。移位操作结束后，编码器输出寄存器内容的运算结果，经过 n 节拍即可输出编码器当前的编码码字。

该编码器由三个移位寄存器和两个模 2 加法器组成，每输入一个码元就会产生两个输出，输出端第 j 时刻分别由下式确定：

$$\begin{cases} c_j^1 = u_j \oplus u_{j-2} \oplus u_{j-3} \\ c_j^2 = u_j \oplus u_{j-1} \oplus u_{j-2} \oplus u_{j-3} \end{cases} \tag{5-2}$$

假设输入信息序列为

$$\boldsymbol{u} = (u_0 u_1 u_2 \cdots) \tag{5-3}$$

则对应输出的两码组为

$$\begin{cases} \boldsymbol{c}^1 = (c_0^1 c_1^1 c_2^1 \cdots) \\ \boldsymbol{c}^2 = (c_0^2 c_1^2 c_2^2 \cdots) \end{cases} \tag{5-4}$$

最终的编码输出是在上下两码组中交替取值

$$\boldsymbol{c} = (c_0^1 c_0^2 c_1^1 c_1^2 c_2^1 c_2^2 \cdots) \tag{5-5}$$

假设输入为 $\boldsymbol{u}=(10111)$，编码开始前先对移位寄存器进行复位（即置零），输入的顺序为 10111，则编码的过程结合式(5-2)、式(5-3)、式(5-4)为

（1）输入 $u_j=1$，则 $c_j^1 = u_j \oplus u_{j-2} \oplus u_{j-3} = 1 \oplus 0 \oplus 0 = 1$，则

$$c_j^2 = u_j \oplus u_{j-1} \oplus u_{j-2} \oplus u_{j-3} = 1 \oplus 0 \oplus 0 \oplus 0 = 1$$

编码输出为 11。

（2）输入 $u_{j+1}=0$，则 $c_{j+1}^1 = u_{j+1} \oplus u_{j-1} \oplus u_{j-2} = 0 \oplus 0 \oplus 0 = 0$，则

$$c_{j+1}^2 = u_{j+1} \oplus u_j \oplus u_{j-1} \oplus u_{j-2} = 0 \oplus 1 \oplus 0 \oplus 0 = 1$$

编码输出为 01。

(3) 输入 $u_{j+2}=1$，则 $c_{j+2}^1=u_{j+2}\oplus u_j\oplus u_{j-1}=1\oplus 1\oplus 0=0$，则

$$c_{j+2}^2=u_{j+2}\oplus u_{j+1}\oplus u_j\oplus u_{j-1}=1\oplus 0\oplus 1\oplus 0=0$$

编码输出为 00。

(4) 输入 $u_{j+3}=1$，则 $c_{j+3}^1=u_{j+3}\oplus u_{j+1}\oplus u_j=1\oplus 0\oplus 1=0$，则

$$c_{j+3}^2=u_{j+3}\oplus u_{j+2}\oplus u_{j+1}\oplus m_j=1\oplus 1\oplus 0\oplus 1=1$$

编码输出为 01。

(5) 输入 $u_{j+4}=1$，则 $c_{j+4}^1=u_{j+4}\oplus u_{j+2}\oplus u_{j+1}=1\oplus 1\oplus 0=0$，则

$$c_{j+4}^2=u_{j+4}\oplus u_{j+3}\oplus u_{j+2}\oplus u_{j+1}=1\oplus 1\oplus 1\oplus 0=1$$

编码输出为 01。

所以编码器的输出为 1101000101，由于是 1/2 码率，所以共有 10 位输出。

如果考虑信息序列输入完移位寄存器的清零位，即增加 3 位零输入，则有如下几种：

(6) 输入 $u_{j+5}=0$，则 $c_{j+5}^1=u_{j+5}\oplus u_{j+3}\oplus u_{j+2}=0\oplus 1\oplus 1=0$，则

$$c_{j+5}^2=u_{j+5}\oplus u_{j+4}\oplus u_{j+3}\oplus u_{j+2}=0\oplus 1\oplus 1\oplus 1=1$$

编码输出为 01。

(7) 输入 $u_{j+6}=0$，则 $c_{j+6}^1=u_{j+6}\oplus u_{j+4}\oplus u_{j+3}=0\oplus 1\oplus 1=0$，则

$$c_{j+6}^2=u_{j+6}\oplus u_{j+5}\oplus u_{j+4}\oplus u_{j+3}=0\oplus 0\oplus 1\oplus 1=0$$

编码输出为 00。

(8) 输入 $u_{j+7}=0$，则 $c_{j+7}^1=u_{j+7}\oplus u_{j+5}\oplus u_{j+4}=0\oplus 0\oplus 1=1$，则

$$c_{j+7}^2=u_{j+7}\oplus u_{j+6}\oplus u_{j+5}\oplus u_{j+4}=0\oplus 0\oplus 0\oplus 1=1$$

编码输出为 11。

故加入清零的码元后，编码器的最终输出为 1101000101010011。

例 5-2 图 5-4 是(3,2,2)系统卷积码编码器，分析输入为 $\boldsymbol{u}=(10\quad 00\quad 00)$和 $\boldsymbol{u}=(01\quad 00\quad 00)$时的输出码序列。

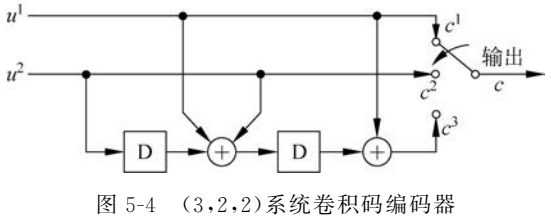


图 5-4 (3,2,2)系统卷积码编码器

例 5-2

由图 5-4 可知

$$c_i^1=u_i^1,\quad c_i^2=u_i^2,\quad c_i^3=u_i^1\oplus u_{i-1}^1\oplus u_{i-1}^2\oplus u_{i-2}^2$$

第 i 时刻的信息段 $\boldsymbol{u}_i=(u_i^1u_i^2)$，码段 $\boldsymbol{c}_i=(c_i^1c_i^2c_i^3)$，编码器由两个移位寄存器构成，所以是(3,2,2)码，输入 $\boldsymbol{u}=(10\quad 00\quad 00)$的编码过程如表 5-1 所示，输出码序列 $\boldsymbol{c}=(101\quad 001\quad 000)$。

表 5-1 (3,2,2)卷积码编码过程

输入 $\boldsymbol{u}$	移存器初态	移存器次态	输出 $\boldsymbol{c}$
10	00	01	101
00	01	00	001
00	00	00	000

同理可得输入 $\boldsymbol{u}=(01\quad 00\quad 00)$ 的编码输出 $\boldsymbol{c}=(010\quad 001\quad 001)$。可以看出码段的左边两个码元和输入的两个信息元始终一致，是系统码。

5.3 卷积码的数学描述

5.3.1 卷积码的码多项式法描述

卷积码的一般编码器如图 5-5 所示。在某一时刻 i，输入到编码器的是由 k 个信息元组成的信息组 u_i，相应的输出序列是由 n 个码元组成的子码 c_i。若输入的信息序列 $u_0u_1u_2u_3\cdots u_i\cdots$ 是一个半无限序列，则由卷积码编码器输出的序列，也是一个由各子码 $c_0c_1c_2c_3\cdots c_i\cdots$ 组成的半无限序列，称此序列为卷积码的一个码序列或码字。

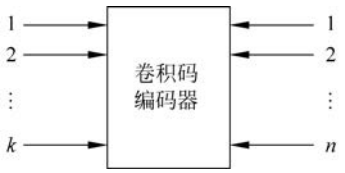


图 5-5 卷积码编码器

类似于前面的表示，可以将输入的序列对应写成多项式的形式。

若 $\boldsymbol{u}=(u_0u_1u_2u_3\cdots)$，则

$$u(x)=u_0+u_1x+u_2x^2+u_3x^3+\cdots \tag{5-6}$$

所以有

$$\boldsymbol{u}=(10111)\rightarrow u(x)=1+x^2+x^3+x^4 \tag{5-7}$$

在分析中，可以用

$$\begin{aligned}\boldsymbol{g}_k^1&=(g_{k,0}^1g_{k,1}^1g_{k,2}^1\cdots g_{k,m}^1)\\ \boldsymbol{g}_k^2&=(g_{k,0}^2g_{k,1}^2g_{k,2}^2\cdots g_{k,m}^2)\\ &\vdots\\ \boldsymbol{g}_k^n&=(g_{k,0}^ng_{k,1}^ng_{k,2}^n\cdots g_{k,m}^n)\end{aligned} \tag{5-8}$$

来表示第 k 个输入端在输出端 $c^1,c^2,\cdots,c^n$ 的求和式的系数，则对于图 5-3 的卷积码编码器，$k=1$，为了书写简便，可以忽略 k 的角标。由式(5-2)可得

$$\begin{cases}c_j^1=1\cdot u_j\oplus 0\cdot u_{j-1}\oplus 1\cdot u_{j-2}\oplus 1\cdot u_{j-3}\\ c_j^2=1\cdot u_j\oplus 1-u_{j-1}\oplus 1\cdot u_{j-2}\oplus 1\cdot u_{j-3}\end{cases}$$

所以有

$$\boldsymbol{g}^1=(1011),\quad \boldsymbol{g}^2=(1111)$$

对应的多项式为

$$g^1(x)=1+x^2+x^3,\quad g^2(x)=1+x+x^2+x^3$$

编码后的多项式为(乘积后合并也是模 2 加)

$$\begin{cases}c^1=u(x)\cdot g^1(x)\\ c^2=u(x)\cdot g^2(x)\end{cases} \tag{5-9}$$

所以有

$$\begin{aligned}c^1&=u(x)\cdot g^1(x)=(1+x^2+x^3+x^4)(1+x^2+x^3)\\ &=1+x^2+x^3+x^4+x^2+x^4+x^5+x^6+x^3+x^5+x^6+x^7=1+x^7\\ c^2&=u(x)\cdot g^2(x)=(1+x^2+x^3+x^4)(1+x+x^2+x^3)\end{aligned}$$

$$=1+x^2+x^3+x^4+x+x^3+x^4+x^5+x^2+x^4+x^5+x^6+x^3+x^5+x^6+x^7$$
$$=1+x+x^3+x^4+x^5+x^7$$

故对应的码序列为

$$\begin{cases}\boldsymbol{c}^1=(10000001)\\ \boldsymbol{c}^2=(11011101)\end{cases}$$

则编码器的输出为 1101000101010011。

5.3.2 卷积码的矩阵生成法描述

以如图 5-6 所示的(2,1,2)卷积码为例来讨论生成矩阵。

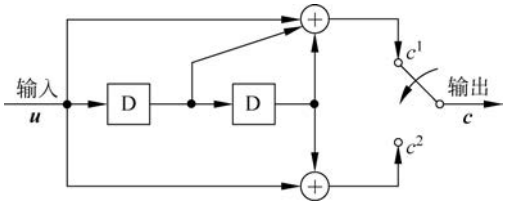


图 5-6 (2,1,2) 卷积码编码器

由图 5-6 可列出表达式。

设 $$\boldsymbol{u}=(u_0u_1u_2\cdots)$$

则

$$c_i^1=u_i\oplus u_{i-1}\oplus u_{i-2},\quad c_i^2=u_i\oplus u_{i-2},\quad \boldsymbol{c}^1=c_0^1c_1^1c_2^1\cdots,\quad \boldsymbol{c}^2=c_0^2c_1^2c_2^2\cdots$$
$$\boldsymbol{c}=c_0^1c_0^2c_1^1c_1^2c_2^1c_2^2\cdots$$

假设移位寄存器初始状态全为 0,当输入信息序列 $\boldsymbol{u}=(100)$时,编码器的工作过程如表 5-2 所示。

表 5-2 (2,1,2)卷积码编码过程

输入 u	移存器初态	移存器次态	输出 $\boldsymbol{c}$
1	00	10	11
0	10	01	10
0	01	00	11

输入信息序列 $\boldsymbol{u}=(100)$时,输出码字 $\boldsymbol{c}=(111011)$。

假设移位寄存器初始状态全为 0,则有

$$c_0^1=u_0 \qquad c_0^2=u_0$$
$$c_1^1=u_1\oplus u_0 \qquad c_1^2=u_1$$
$$c_2^1=u_2\oplus u_1\oplus u_0 \qquad c_2^2=u_2\oplus u_0$$
$$c_3^1=u_3\oplus u_2\oplus u_1 \qquad c_3^2=u_3\oplus u_1$$
$$\vdots$$

上述方程组的矩阵形式为

$$\boldsymbol{c}=[c_0^1c_0^2c_1^1c_1^2c_2^1c_2^2\cdots]=[u_0u_1u_2u_3\cdots]\cdot\begin{bmatrix}11101100\cdots\\00111011\cdots\\00001110\cdots\\00000011\cdots\\\vdots\end{bmatrix}$$

即

$$\boldsymbol{c}=\boldsymbol{u}\boldsymbol{G}_\infty \tag{5-10}$$

$\boldsymbol{G}_\infty$称为(2,1,2)卷积码的生成矩阵，这是一个半无限矩阵，重写为

$$\boldsymbol{G}_\infty=\begin{bmatrix}11&10&11&&&&\\&11&10&11&&\mathbf{0}&\\&&11&10&11&&\\\mathbf{0}&&&11&10&11&\\&&&&&&\ddots\end{bmatrix}$$

仔细观察$\boldsymbol{G}_\infty$可发现它的每一行都是前一行右移n位的结果，也就是说，它完全是由矩阵的第一行确定的。将第一行取出并表示为

$$\boldsymbol{g}_\infty=[11\quad10\quad11\quad00\quad\cdots] \tag{5-11}$$

$\boldsymbol{g}_\infty$称为该码的基本生成矩阵，通过与表5-2比较，$\boldsymbol{g}_\infty$其实就是当$\boldsymbol{u}=(100\cdots0)$，即输入信息序列为冲激序列时卷积码编码器的冲激响应。

令输入信息序列为$\boldsymbol{u}=(10101)$，则输出码字

$$\boldsymbol{c}=\boldsymbol{u}\boldsymbol{G}_\infty=[10101]\begin{bmatrix}11&10&11&&&&&\\&11&10&11&&\mathbf{0}&&\\&&11&10&11&&&\cdots\\&\mathbf{0}&&11&10&11&&\\&&&&11&10&11&\end{bmatrix}$$

$$=[11\quad10\quad00\quad10\quad00\quad10\quad11\quad\cdots]$$

再如例5-2所示的(3,2,2)系统卷积码，当$\boldsymbol{u}=(10\quad00\quad00\quad\cdots)$时，相应的冲激响应为$\boldsymbol{c}=(101\quad001\quad000\quad\cdots)$；当$\boldsymbol{u}=(01\quad00\quad00\quad\cdots)$时，$\boldsymbol{c}=(010\quad001\quad001\quad\cdots)$。

由$\boldsymbol{u}=(10\quad00\quad00\quad\cdots)$和$\boldsymbol{u}=(01\quad00\quad00\quad\cdots)$的冲激响应，得该码的基本生成矩阵为

$$\boldsymbol{g}_\infty=\begin{bmatrix}101&001&000&\cdots\\010&001&001&\cdots\end{bmatrix}$$

将$\boldsymbol{g}_\infty$作为生成矩阵$\boldsymbol{G}_\infty$的最上面两行，并经位移得该码的生成矩阵为

$$\boldsymbol{G}_\infty=\begin{bmatrix}101&001&000&&&&&\\010&001&001&&&&&\\&101&001&000&&\mathbf{0}&&\\&010&001&001&&&&\\&&101&001&000&&&\\&&010&001&001&&&\\&\mathbf{0}&&101&001&000&&\\&&&010&001&001&&\\&&&&&&&\ddots\end{bmatrix}$$

显然，若输入信息序列 $\boldsymbol{u}=(10\quad 11\quad 01\quad 11\quad \cdots)$，则相应的输出码序列应为

$$\boldsymbol{c}=\boldsymbol{u}\boldsymbol{G}_{\infty}=[10\quad 11\quad 01\quad 11\quad \cdots]\begin{bmatrix}101 & 001 & 000 & 000 & 000 & 000\\ 010 & 001 & 001 & 000 & 000 & 000\\ 000 & 101 & 001 & 000 & 000 & 000\\ 000 & 010 & 001 & 001 & 000 & 000\\ 000 & 000 & 101 & 001 & 000 & 000\cdots\\ 000 & 000 & 010 & 001 & 001 & 000\\ 000 & 000 & 000 & 101 & 001 & 000\\ 000 & 000 & 000 & 010 & 001 & 001\\ & & & \vdots & & \end{bmatrix}$$

$$=[101\quad 110\quad 010\quad 111\quad 001\quad 001\quad \cdots]$$

一般情况下，(n,k,m)卷积码的生成矩阵可表示为

$$\boldsymbol{G}_{\infty}=\begin{bmatrix}\boldsymbol{G}_0 & \boldsymbol{G}_1 & \boldsymbol{G}_2 & \cdots & \boldsymbol{G}_m & & & \\ & \boldsymbol{G}_0 & \boldsymbol{G}_1 & \cdots & \boldsymbol{G}_{m-1} & \boldsymbol{G}_m & & \boldsymbol{0} \\ & & \boldsymbol{G}_0 & \cdots & \boldsymbol{G}_{m-2} & \boldsymbol{G}_{m-1} & \boldsymbol{G}_m & \\ & \boldsymbol{0} & & & & & & \ddots \end{bmatrix} \tag{5-12}$$

基本生成矩阵为

$$\boldsymbol{g}_{\infty}=[\boldsymbol{G}_0\quad \boldsymbol{G}_1\quad \boldsymbol{G}_2\quad \cdots\quad \boldsymbol{G}_m\quad \boldsymbol{0}\quad \cdots] \tag{5-13}$$

其中，生成子矩阵为

$$\boldsymbol{G}_l=\begin{bmatrix}g_{1,l}^1 & g_{1,l}^2 & \cdots & g_{1,l}^n\\ g_{2,l}^1 & g_{2,l}^2 & \cdots & g_{2,l}^n\\ \vdots & \vdots & \ddots & \vdots\\ g_{k,l}^1 & g_{k,l}^2 & \cdots & g_{k,l}^n\end{bmatrix}\quad (0\leqslant l\leqslant m) \tag{5-14}$$

生成矩阵中每一行的分组数(即码段数)为编码的约束长度，矩阵的总行数取决于输入信息序列的长度。

例 5-3 由例 5-1 给出的(2,1,3)卷积码编码器结构，$n=2,k=1,m=3$，可知

$$\begin{cases}c_j^1=u_j\oplus u_{j-2}\oplus u_{j-3}\\ c_j^2=u_j\oplus u_{j-1}\oplus u_{j-2}\oplus u_{j-3}\end{cases}$$

则有

$$\boldsymbol{g}^1=(1011),\quad \boldsymbol{g}^2=(1111)$$

$$\boldsymbol{G}_l=\begin{bmatrix}g_{1,l}^1 & g_{1,l}^2 & \cdots & g_{1,l}^n\\ g_{2,l}^1 & g_{2,l}^2 & \cdots & g_{2,l}^n\\ \vdots & \vdots & \ddots & \vdots\\ g_{k,l}^1 & g_{k,l}^2 & \cdots & g_{k,l}^n\end{bmatrix}=\begin{bmatrix}g_{1,l}^1 & g_{1,l}^2\end{bmatrix}$$

$$\boldsymbol{G}_0=[g_{1,0}^1 g_{1,0}^2]=[11],\quad \boldsymbol{G}_1=[g_{1,1}^1 g_{1,1}^2]=[01]$$

$$\boldsymbol{G}_2=[g_{1,2}^1 g_{1,2}^2]=[11],\quad \boldsymbol{G}_3=[g_{1,3}^1 g_{1,3}^2]=[11]$$

所以基本生成矩阵为

$$\boldsymbol{g}_{\infty}=[\boldsymbol{G}_0 \quad \boldsymbol{G}_1 \quad \boldsymbol{G}_2 \quad \cdots \quad \boldsymbol{G}_m \quad 0 \quad \cdots]=[11 \quad 01 \quad 11 \quad 11\cdots]$$

所以生成矩阵为

$$\boldsymbol{G}=\begin{bmatrix} 11 & 01 & 11 & 11 & & & & \\ & 11 & 01 & 11 & 11 & & & \\ & & 11 & 01 & 11 & 11 & & \\ & & & 11 & 01 & 11 & 11 & \\ & & & & 11 & 01 & 11 & 11 \end{bmatrix}$$

当输入为 $\boldsymbol{u}=(10111)$ 时，有

$$\boldsymbol{c}=\boldsymbol{u}\boldsymbol{G}=[10111]\begin{bmatrix} 11 & 01 & 11 & 11 & & & & \\ & 11 & 01 & 11 & 11 & & & \\ & & 11 & 01 & 11 & 11 & & \\ & & & 11 & 01 & 11 & 11 & \\ & & & & 11 & 01 & 11 & 11 \end{bmatrix}$$

$$=[1101000101010011]$$

例 5-4 如图 5-7 是一个(3,2,1)卷积码编码器结构，$n=3,k=2,m=1$。编码速率为 $R=2/3$，假设输入为 $\boldsymbol{u}=(110110)$，求编码器的输出。

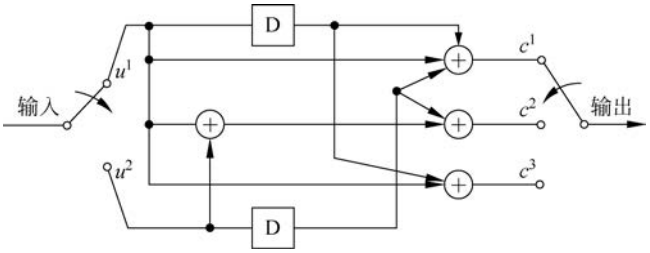


图 5-7 (3,2,1)卷积码编码器结构

图 5-7 中的编码器有两个输入端，所以分析会更复杂一些，但是分析过程更具有代表性。

(1) 矩阵生成法。

第一个输入端与三个输出端的关系为

$$c_j^{1,1}=u_j^1\oplus u_{j-1}^1,\quad c_j^{1,2}=u_j^1,\quad c_j^{1,3}=u_j^1\oplus u_{j-1}^1$$

第二个输入端与三个输出端的关系为

$$c_j^{2,1}=u_{j-1}^2,\quad c_j^{2,2}=u_j^2\oplus u_{j-1}^2,\quad c_j^{2,3}=0$$

也就是

$$g_1^1=u_j^1\oplus u_{j-1}^1,\quad g_1^2=u_j^1,\quad g_1^3=u_j^1\oplus u_{j-1}^1$$

$$g_2^1=u_{j-1}^2,\quad g_2^2=u_j^2\oplus u_{j-1}^2,\quad g_2^3=0$$

可知

$$g_{1,0}^1=1,\quad g_{1,1}^1=1$$

$$g_{1,0}^2=1,\quad g_{1,1}^2=0$$

$$g_{1,0}^{3}=1,\quad g_{1,1}^{3}=1$$
$$g_{2,0}^{1}=0,\quad g_{2,1}^{1}=1$$
$$g_{2,0}^{2}=1,\quad g_{2,1}^{2}=1$$
$$g_{2,0}^{3}=0,\quad g_{2,1}^{3}=0$$

因为 $k=2$,所以

$$\boldsymbol{G}_l=\begin{bmatrix} g_{1,l}^{1} & g_{1,l}^{2} & g_{1,l}^{3} \\ g_{2,l}^{1} & g_{2,l}^{2} & g_{2,l}^{3} \end{bmatrix}\quad (0\leqslant l\leqslant m)$$

因为 $m=1$,所以

$$\boldsymbol{G}_0=\begin{bmatrix}111\\010\end{bmatrix},\quad \boldsymbol{G}_1=\begin{bmatrix}101\\110\end{bmatrix}$$

于是

$$\boldsymbol{G}_{\infty}=\begin{bmatrix} \boldsymbol{G}_0 & \boldsymbol{G}_1 & \boldsymbol{G}_2 & \cdots & \boldsymbol{G}_m & & & \\ & \boldsymbol{G}_0 & \boldsymbol{G}_1 & \cdots & \boldsymbol{G}_{m-1} & \boldsymbol{G}_m & & \boldsymbol{0} \\ & & \boldsymbol{G}_0 & \cdots & \boldsymbol{G}_{m-2} & \boldsymbol{G}_{m-1} & \boldsymbol{G}_m & \\ & \boldsymbol{0} & & & & & & \ddots \end{bmatrix}=\begin{bmatrix} \boldsymbol{G}_0 & \boldsymbol{G}_1 & & \\ & \boldsymbol{G}_0 & \boldsymbol{G}_1 & \\ & & \boldsymbol{G}_0 & \boldsymbol{G}_1 \end{bmatrix}$$

$$=\begin{bmatrix} 111 & 101 & & \\ 010 & 110 & & \\ & 111 & 101 & \\ & 010 & 110 & \\ & & 111 & 101 \\ & & 010 & 110 \end{bmatrix}$$

所以

$$\boldsymbol{c}=\boldsymbol{u}\boldsymbol{G}=[110110]\begin{bmatrix} 111 & 101 & & \\ 010 & 110 & & \\ & 111 & 101 & \\ & 010 & 110 & \\ & & 111 & 101 \\ & & 010 & 110 \end{bmatrix}=[101001001101]$$

(2) 多项式生成法。

因为输入序列为 $\boldsymbol{u}=(110110)$,所以经过串/并转换可以得到两个输入子序列 $\boldsymbol{u}^1=(101)$,$\boldsymbol{u}^2=(110)$,所以得到输入序列的多项式形式

$$\begin{cases}\boldsymbol{u}^1=(101)\rightarrow u^1(x)=1+x^2\\ \boldsymbol{u}^2=(110)\rightarrow u^2(x)=1+x\end{cases}$$

由于

$$g_1^1=u_j^1\oplus u_{j-1}^1,\quad g_1^2=u_j^1,\quad g_1^3=u_j^1\oplus u_{j-1}^1$$
$$g_2^1=u_{j-1}^2,\quad g_2^2=u_j^2\oplus u_{j-1}^2,\quad g_2^3=0$$

得

$$\begin{cases} \boldsymbol{g}_1^1=(11)\rightarrow g_1^1(x)=1+x \\ \boldsymbol{g}_1^2=(10)\rightarrow g_1^2(x)=1 \\ \boldsymbol{g}_1^3=(11)\rightarrow g_1^3(x)=1+x \\ \boldsymbol{g}_2^1=(01)\rightarrow g_2^1(x)=x \\ \boldsymbol{g}_2^2=(11)\rightarrow g_2^2(x)=1+x \\ \boldsymbol{g}_2^3=(00)\rightarrow g_2^3(x)=0 \end{cases}$$

$$\boldsymbol{G}=\begin{bmatrix} g_{1,l}^1 & g_{1,l}^2 & g_{1,l}^3 \\ g_{2,l}^1 & g_{2,l}^2 & g_{2,l}^3 \end{bmatrix}=\begin{bmatrix} 1+x & 1 & 1+x \\ x & 1+x & 0 \end{bmatrix}$$

$$\boldsymbol{c}=\begin{bmatrix} u^1(x) \\ u^2(x) \end{bmatrix}^{\mathrm{T}}\cdot\boldsymbol{G}(\boldsymbol{x})=\begin{bmatrix} 1+x^2 & 1+x \end{bmatrix}\cdot\begin{bmatrix} 1+x & 1 & 1+x \\ x & 1+x & 0 \end{bmatrix}$$

$$=\begin{bmatrix} 1+x^3 & 0 & 1+x+x^2+x^3 \end{bmatrix}$$

所以有

$$\begin{cases} c^1(x)=1+x^3\rightarrow\boldsymbol{c}^1=(1001) \\ c^2(x)=0\rightarrow\boldsymbol{c}^2=(0000) \\ c^3(x)=1+x+x^2+x^3\rightarrow\boldsymbol{c}^3=(1111) \end{cases}$$

所以，卷积码编码器的输出为 $\boldsymbol{c}=(101001001101)$，最终的输出结果与矩阵法输出结果相同。

5.3.3 卷积码的离散卷积法描述

对于例 5-1，编码后多项式表示为

$$\begin{cases} c^1(x)=u(x)\cdot g^1(x) \\ c^2(x)=u(x)\cdot g^2(x) \end{cases} \tag{5-15}$$

其对应的编码方程为

$$\begin{cases} \boldsymbol{c}^1=\boldsymbol{u}*\boldsymbol{g}^1 \\ \boldsymbol{c}^2=\boldsymbol{u}*\boldsymbol{g}^2 \end{cases} \tag{5-16}$$

式中，“$*$”表示卷积运算，卷积码因此而得名。$\boldsymbol{g}^1$ 和 $\boldsymbol{g}^2$ 表示编码器的两个冲激响应。

由前面的分析知 $\boldsymbol{g}^1=(1011)$，$\boldsymbol{g}^2=(1111)$，且 $\boldsymbol{u}=(10111)$，最后求得

$$\begin{cases} \boldsymbol{c}^1=\boldsymbol{u}*\boldsymbol{g}^1=(10111)*(1011)=(10000001) \\ \boldsymbol{c}^2=\boldsymbol{u}*\boldsymbol{g}^2=(10111)*(1111)=(11011101) \end{cases}$$

交织后可得编码器的输出为 1101000101010011。

5.4 卷积码的图形描述

5.4.1 状态图

编码过程可以用状态图来表示。在卷积码编码器中，寄存器任一时刻存储的数据称为编码器的一个状态，随着信息序列的不断输入，编码器的状态在不断变化，同时输出的码元序列也相应地发生改变。所谓状态图就是反映编码器中寄存器存储状态转移的关系图，它用编码器中寄存器的状态及其随输入序列而发生的转移关系来描述编码过程。

例 5-5 图 5-6 的(2,1,2)卷积码编码器由两级移位寄存器组成，因此状态只有 4 种可能，即 00、10、01、11，用符号 S_i 表示，分别将其对应为 S_0、S_1、S_2 和 S_3。

表 5-3 和图 5-8 分别为(2,1,2)卷积码的状态表和状态图。

状态图中，用实线表示信息 0 输入，虚线表示 1 输入，若输入信息 $\boldsymbol{u}=(10101)$，状态转移为 $S_0 \rightarrow S_1 \rightarrow S_2 \rightarrow S_1 \rightarrow S_2 \rightarrow S_1 \rightarrow S_2 \rightarrow S_0$，相应输出码元序列为(11 10 00 10 00 10 11)。

编码器输出的码元序列是在信息序列的第一个码元输入直到最后一个码元完全移出移位寄存器所产生的。要求有用信息序列输入完毕后，应再向编码器输入 mk 个全零码元，所以最终状态应回到初始状态 S_0。

表 5-3 (2,1,2)卷积码的状态表

输入 $\boldsymbol{u}$	初态 S_i	次态 S_j	输出 $\boldsymbol{c}$
0	00	00	00
1	00	10	11
0	10	01	10
1	10	11	01
0	01	00	11
1	01	10	00
0	11	01	01
1	11	11	10

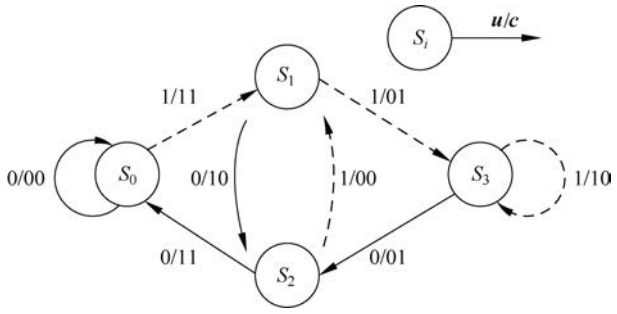


图 5-8 (2,1,2)卷积码的状态图

5.4.2 树图

树图结构是由状态图按时间展开的。即将输入信息序列 $\boldsymbol{u}$ 的输入顺序按时间顺序($l=$

0,1,2,…)展开,并展开所有可能的输入输出情况。

如果(n,k,m)卷积码编码器的输入信息序列是半无限长序列,则它的输出码元序列也应是半无限长序列,这种半无限长序列的输入、输出编码过程可用半无限码树来表示。

如图 5-9 所示为图 5-6 的(2,1,2)卷积码的树图。

图 5-9 中,节点处符号为移存器状态,分支上的数据为输出码段,上分支为输入信息 0,下分支为输入信息 1。设移位寄存器初始状态 $S_0=00$,当输入信息元 $u_0=0$ 时,由树根出发走上支路,移存器右移一位,状态仍为 S_0,输出码段 $c_0=00$;当 $u_0=1$ 时,由树根出发走下支路,移存器状态转为 $S_1=10$,此时输出码段为 $c_0=11$。

在输入第二位信息元时,编码器已处于 1 阶节点处,若在 S_0 点,则输入 $u_1=0$ 时走上分支,输出 $c_1=00$,新状态为 S_0;输入 $u_1=1$ 时下分支,输出 $c_1=11$,新状态为 S_1;若在 S_1 点,则输入 $u_1=0$ 时走上分支,输出 $c_1=10$,新状态为 $S_2=10$;输入 $u_1=1$ 时走下分支,输出 $c_1=01$,新状态为 $S_3=11$。

再输入 u_2,编码器从 2 阶节点处出发,此时起始状态有 4 种:S_0、S_1、S_2 和 S_3,按输入 0 走上分支,输入 1 走下分支码段的规则,得到相应的输出 c_2 和新状态。依次类推,输入无限长信息序列,就可以得到一个无限延伸的树结构图。

从码树图上观察,输入无限长信息序列,就可以得到一个无限延伸的树结构图。输入不同的信息序列,编码器就走不同的路径,输出不同的码元序列。

在树图中,编码的过程相当于以输入信息序列为指令沿码树游走,在树图中所经过的路径代码就是相应输出的码序列。

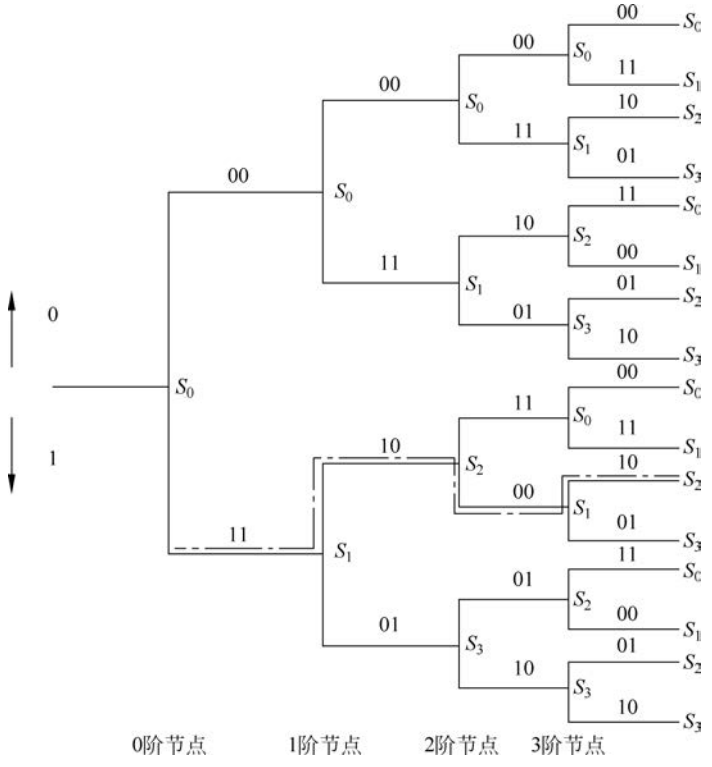


图 5-9 (2,1,2)卷积码的树图

树图最大特点是按时间顺序展开的($l=0,1,2,\cdots$),且能将所有时序状态表示为不相重合的路径。

一般地,对于二元(n,k,m)卷积码来说,从每个节点发出 2^k 条分支,每条分支上标有 n 长输出数据,最多可有 2^{km} 种不同状态。

状态图从状态上看最为简洁,但时序关系不清晰。码树的最大特点是时序关系清晰,且对于每一个输入信息序列都有一个唯一的不重复的树枝结构相对应,它的主要缺点是进行到一定时序后,状态将产生重复,且树图越来越复杂。

若输入信息为 $\boldsymbol{u}=(10101)$,则由图 5-9 可知卷积编码序列为 11100010,如图 5-9 中点划线所示。由于树图只展示到第 3 个节点,所以第 4 个节点输入信息的编码看不到,这也正是树图的缺点所在。

5.4.3 网格图

网格图又称篱笆图,它综合了状态图和树图的特点,是将码树中处于同一级节点合并而成的,是一个纵深宽度或者高为 2^{km} 的网格图,结构简单,而且时序关系清晰。

网格图的最大特点是保持了树图的时序展开性,同时又克服了树图中太复杂的缺点,它将树图中产生的重复状态合并起来。

树图中,从某一阶节点开始所长出的分支从纵向看是周期重复的。如图 5-9(2,1,2)码的树图中,当节点数大于 $m+1=3$ 时,状态 S_0、S_1、S_2 和 S_3 重复出现,因此在第($m+1$)阶节点以后,将树图上处于同一状态的同一节点折叠起来加以合并,就可以得到网格图。

图 5-10 是信息序列长度 l 为 5 的(2,1,2)码网格图,实线表示输入为 0 的分支,虚线表示输入为 1 的分支。分支上标准的 n 位数据表示相应的编码输出 $\boldsymbol{c}$。从第 m 至 l 节点,编码器处于稳定的状态转移中,并且各节点的网格结构均相同。在 l 节点后 m 个移存器尚需转移 m 个状态,才能回到初始状态 S_0。由于 l 到($l+m$)节点的过程中输入 0,所以只有实线分支。

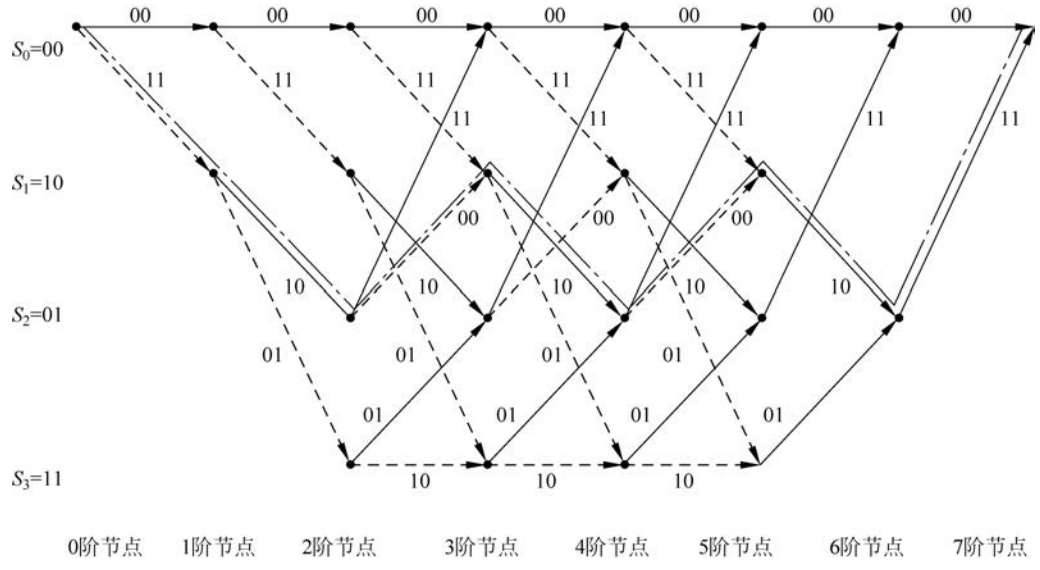


图 5-10 (2,1,2)卷积码网格图

与树图一样，网格图中每一种信息序列有唯一的网格编码路径，图中输入信息序列 $\boldsymbol{u}=(10101)$路径对应的输出码元序列 $\boldsymbol{c}=(11\quad 10\quad 00\quad 10\quad 00\quad 10\quad 11)$，如图 5-10 中点划线所示，编码后面多出了 1011 四位，是因为在输入信息序列后加入了两位移位寄存器清零码元，因此实际上输入为 $\boldsymbol{u}=(1010100)$。

5.5 卷积码的译码

卷积码的译码可分为三大类：序列译码、门限译码和 Viterbi 译码。序列译码基于码的树图结构，能很好地处理约束度很长的卷积码，缺点是它的译码时间是可变的；门限译码基于码的代数结构，通过计算伴随式集合实现，缺点是在误码率方面表现较差；Viterbi 译码基于码的网格图结构，是一种极大似然译码方法。

5.5.1 卷积码的硬判决和软判决

图 5-11 所示为卷积码编译码系统的简要框图。信息序列 $\boldsymbol{u}$ 编码成为传输序列 $\boldsymbol{c}$，经过有噪声的信道后接收端接收到序列 $\boldsymbol{y}$，由卷积译码器给出译码序列$\hat{\boldsymbol{u}}$。

在图 5-11 中，噪声信道的输入序列 $\boldsymbol{c}$ 是一个二进制符号序列，对其输出序列 $\boldsymbol{y}$ 如果也按二进制数据进行判决，给出译码序列$\hat{\boldsymbol{u}}$，则一般称为硬判决（硬量化）卷积译码。如果为了充分利用信道输出序列的数据信息以提高译码可靠性，可将信道输出的数据做多电平量化，例如 8 电平量化，再进行卷积译码，则通常称为软判决（软量化）卷积译码。对 AWGN 信道来说，软判决译码比硬判决译码可获得 2dB 的性能改善。

图 5-11 卷积码编译码系统框图

5.5.2 硬判决的 Viterbi 译码

硬判决的 Viterbi 算法是一种汉明距离译码。通常所说的 Viterbi 译码就是硬判决的 Viterbi 译码。

Viterbi 译码方法采用分段处理，每个码段根据接收的码元序列，按照极大似然译码准则，寻找发送端编码器在网格图上所经过的最佳路径，也就是在网格图上寻找与接收码相比差距最小的可行路径。对于 BSC 信道，这种寻找可等价为确定与接收码段具有最小汉明距离的路径。由于接收序列通常很长，所以 Viterbi 译码时最大似然译码作了简化，即它把接收码字分段累接处理，每接收一段码字，计算比较一次，保留码距最小的路径，直至译完整个序列。

Viterbi 译码具有以下优点：

(1) 有固定的译码时间；

(2) 适于译码器的硬件实现，运行速度快；

(3) 译码的错误概率可以达到很小；

(4) 容易实现，成本低。

1. Viterbi 译码算法的步骤

对于(n,k,m)卷积码，假设已接收 l 个码段，Viterbi 译码算法可归纳为以下步骤：

(1) 在 $j=m$ 节点处，对进入每一状态的长度为 $j=m$ 的部分路径，计算输出数据与对应接收的 j 个 n 长码段的汉明距离。将部分路径存储作为被留选的幸存路径(部分路径是指前 m 个码段的路径，是全路径的一部分，而不是前 m 个码段路径的不同分支路径)。

(2) j 增加 1，把此时进入每一状态的所有分支与前一阶节点处留选的幸存路径累积，计算这些路径与相应接收码段的汉明距离，每个状态留选汉明距离最小者为对应的幸存路径，其余路径则删除。

(3) 重复步骤(2)，直至 $j=l+m$，最终整个网格图中只剩一条幸存路径，译码结束。

由 m 至 l 阶节点，网格图中 2^{km} 个状态中每一个状态有一条幸存路径；但在 l 阶节点后，状态数目减少，幸存路径随之相应减少；至 $l+m$ 阶节点时，仅剩一条幸存路径，它就是译码器输出的最佳估值码元序列路径。

如果在某阶节点时，某状态的两条路径具有相同的汉明距离，这时需观察下一阶节点累积的汉明距离，再选定最小距离的路径。

2. Viterbi 译码过程

对于图 5-6 的编码器，设输入信息序列 $\boldsymbol{u}=(10101)$，通过 BSC 后送入译码器的序列 $\boldsymbol{y}=(11\ \ 10\ \ 01\ \ 11\ \ 00\ \ 10\ \ 11)$，采用 Viterbi 译码算法对信息序列和码序列进行估值。

前面已经推导出，当输入信息序列 $\boldsymbol{u}=(10101)$ 时，正确的输出码序列是 $\boldsymbol{c}=(11\ \ 10\ \ 00\ \ 10\ \ 00\ \ 10\ \ 11)$，与实际接收序列 $\boldsymbol{y}$ 比较有 2 个码元错误。根据图 5-10 的网格图，Viterbi 译码器对接收序列 $\boldsymbol{y}$ 的译码过程示于图 5-12 中，y_i 为接收码段，d 为汉明距离，$\hat{u}$ 为信息估值。

在图 5-12(a)中，从初始状态到达 $j=2$ 阶节点的 4 种状态有 4 条路径，这 4 条路径与接收序列 $\boldsymbol{y}_0\boldsymbol{y}_1=(11\ \ 10)$的汉明距离分别为 3、3、0、2，依次作为 4 种状态的幸存路径。

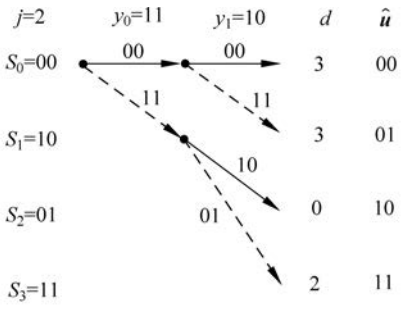
当 $j=3$ 时，如图 5-12(b)所示，沿前一阶节点的幸存路径到达 S_0 状态有两条路径 $S_0 \xrightarrow{00} S_0 \xrightarrow{00} S_0 \xrightarrow{00} S_0$ 和 $S_0 \xrightarrow{11} S_1 \xrightarrow{10} S_2 \xrightarrow{11} S_0$，与 $\boldsymbol{y}_0\boldsymbol{y}_1\boldsymbol{y}_2=(11\ \ 10\ \ 01)$的汉明距离分别为 4 和 1，选取距离最小者为 S_0 状态的幸存路径。同样 S_1、S_2、S_3 状态也都有两条路径，分别留选距离最小者为幸存路径，其余路径则删除。

接着由 $j=3$ 的 S_0、S_1、S_2 状态转移到 $j=4$ 的 S_0、S_1、S_2、S_3 状态，如图 5-12(c)所示，有 4 条路径与 $\boldsymbol{y}_0\boldsymbol{y}_1\boldsymbol{y}_2\boldsymbol{y}_3=(11\ \ 10\ \ 01\ \ 11)$比较，具有最小汉明距离，被留选下来。

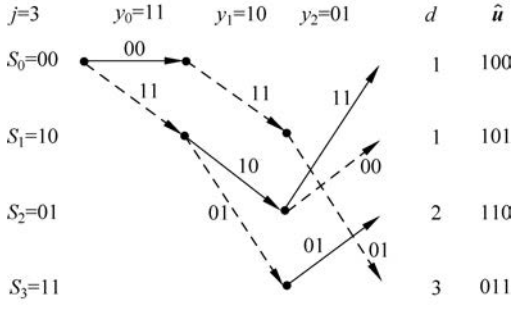
同样由 $j=4$ 到 $j=5$ 节点，如图 5-12(d)所示，每一种状态都各自留选了一条幸存路径。

在图 5-12(e)当 $j=6$ 时，有用信息已输入完毕，输入端补零至编码器，所以只剩下 S_0 和 S_2 两种状态，而 S_0 状态的两条路径与 $\boldsymbol{y}_0\boldsymbol{y}_1\boldsymbol{y}_2\boldsymbol{y}_3\boldsymbol{y}_4\boldsymbol{y}_5=(11\ \ 10\ \ 01\ \ 11\ \ 00\ \ 10)$的距离都是 3，因此都被留存。

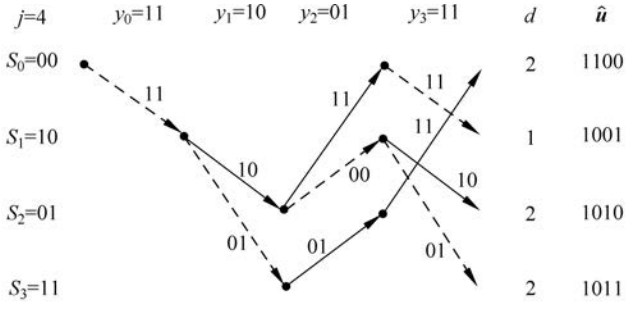
当 $j=7$ 时，如图 5-12(f)回到初始状态 S_0，只剩唯一的一条幸存路径，其对应的输出码序列就是接收码序列的最佳估值 $\hat{\boldsymbol{y}}=(11\ \ 10\ \ 00\ \ 10\ \ 00\ \ 10\ \ 11)$，相应的信息序列估值 $\hat{\boldsymbol{u}}=(10101)$。



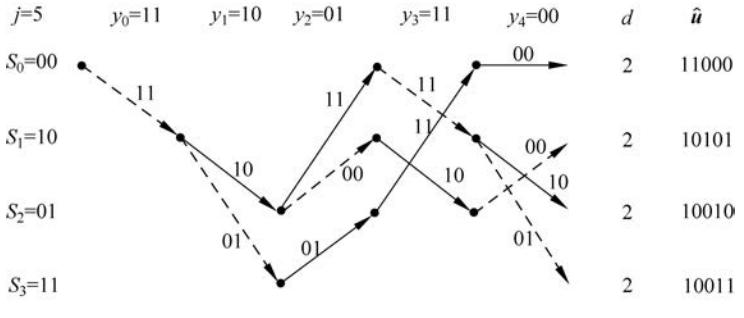
(a)



(b)



(c)



(d)

图 5-12　Viterbi 译码过程

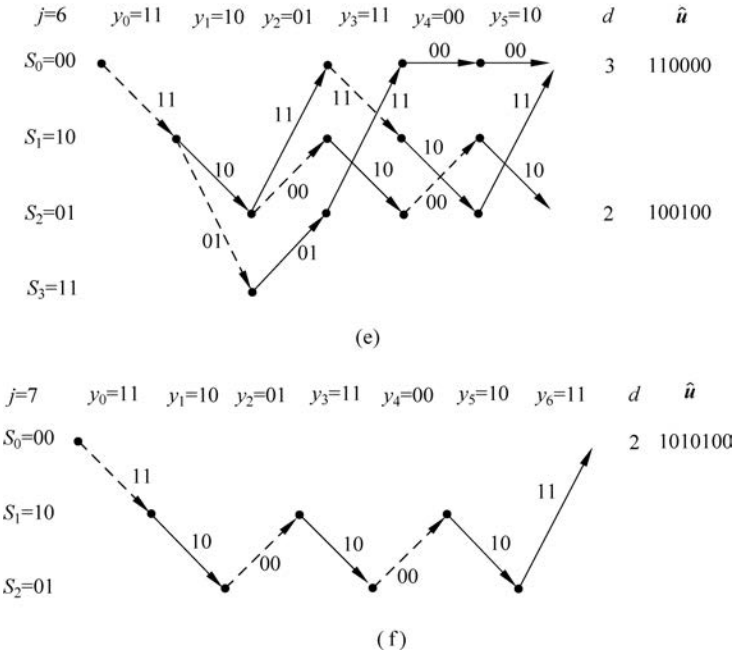


图 5-12 （续）

可见，译码结果与编码器的输出结果一致，实现了正确译码。

由于幸存路径长度为 L，共需 $L \cdot 2^m$ 个段存储单元存储全部幸存路径，因此对实际中几乎无穷大的传送序列，若记 $L=L_d$ 为译码输出时刻，L_d 的值不可能太大，通常 L_d 选择为约束长度 N 的 5～10 倍，称为译码深度，$L_d=(5\sim10)N$。当实际序列长度 $L \gg L_d$ 时，译码器可以是逐 L_d 段长进行译码。

5.5.3 软判决的 Viterbi 译码

为了充分利用接收信号的全部信息，提高译码性能，可将硬判决改为软判决。即对接收信号进行多电平判决或进行多进制 Viterbi 译码。

硬判决 Viterbi 译码是二电平判决，适合于二进制对称信道(BSC)。对于二电平判决，当噪声干扰较大时判决容易丢失有用信息。而采用多电平判决，可改善 Viterbi 译码器性能(1.5～2dB)。软判决 Viterbi 译码适合于离散无记忆信道(DMC)，通常采用 4～8 电平。

软判决 Viterbi 译码与硬判决 Viterbi 译码的算法大体相同，不同之处主要表现在路径量度的求法不同，因为多进制的最大似然度不再是简单的汉明距离。因此路径量度要用对数似然函数来计算。

5.6 卷积码举例

如图 5-13 所示的卷积码编码器，$n=3$，$k=1$，$m=2$，码率等于 1/3。下面以该卷积码编码器为例对卷积码做较详细地分析和讨论。

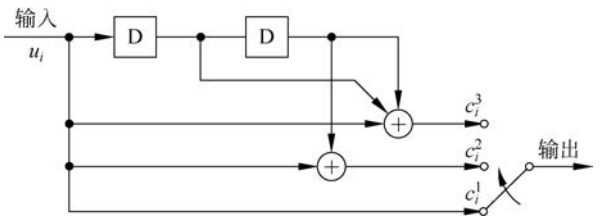


图 5-13　一种(3,1,2)卷积码编码器的方框图

由图可知

$$\begin{cases} c_i^1 = u_i \\ c_i^2 = u_i \oplus u_{i-2} \\ c_i^3 = u_i \oplus u_{i-1} \oplus u_{i-2} \end{cases}$$

输出码字为 $\boldsymbol{c}_i = (c_i^1 c_i^2 c_i^3)$。

1. 状态表

由 5.4 的内容可知，一个卷积码编码器的状态图、树图、网格图是由编码器决定的，与输入信息无关。此卷积码编码器由两级移位寄存器组成，因此只有 4 种状态，00、10、01 和 11，分别用符号 S_0、S_1、S_2 和 S_3 表示。而对于每一种状态都有“0”和“1”两种输入，于是根据图 5-13 的编码器可得到如表 5-4(a)所示的状态表。

表 5-4(a)　(3,1,2)卷积码的状态表

输入 $\boldsymbol{u}$	初态 S_i	初态 S_j	输出 $\boldsymbol{c}$
0	0 0	0 0	0 0 0
1	0 0	1 0	1 1 1
0	1 0	0 1	0 0 1
1	1 0	1 1	1 1 0
0	0 1	0 0	1 1 0
1	0 1	1 0	1 0 0
0	1 1	0 1	0 1 0
1	1 1	1 1	1 0 1

若输入信息位是 1101，则由编码器得到如表 5-4(b)的状态变换表。

表 5-4(b)　(3,1,2)卷积码的状态变换表

输入 $\boldsymbol{u}$	初态 S_i	初态 S_j	输出 $\boldsymbol{c}$
1	0 0	1 0	1 1 1
1	1 0	1 1	1 1 0
0	1 1	0 1	0 1 0
1	0 1	1 0	1 0 0
0	1 0	0 1	0 0 1
0	0 1	0 0	0 1 0

由表 5-4(b)可以得到对应的编码为 111 110 010 100 001 010。此处输入信息增加了两位清零码元“0”。

2. 状态图

由表 5-4(a)可得到如图 5-14 所示的状态图，图中实线表示输入信息位为“0”时的状态转变路线，虚线表示输入信息位为“1”时的状态转变路线。

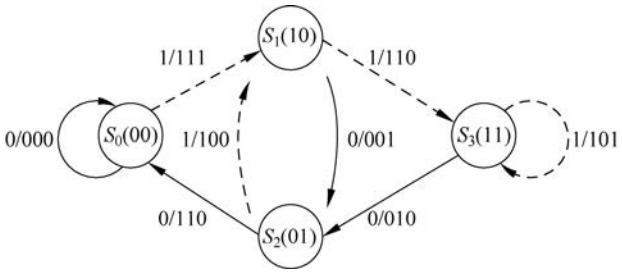


图 5-14 (3,1,2)卷积码的状态图

若输入信息位是 1101，由状态图 5-14 可得编码输出为 111 110 010 100。

3. 树图

规定输入信息为“0”，状态向上支路移动，输入信息为“1”，则状态向下支路移动，于是由表 5-4 得出如图 5-15 所示的树图。

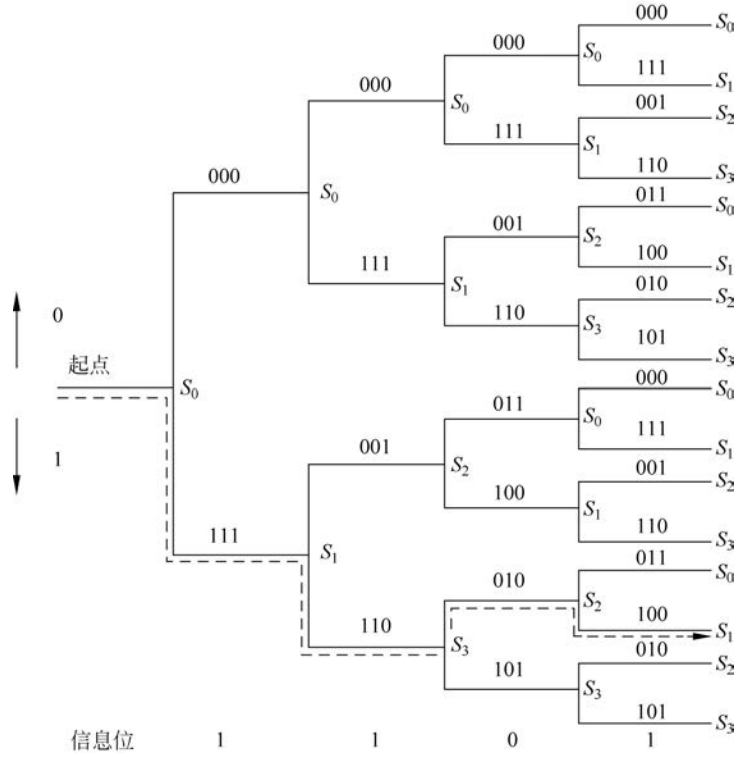


图 5-15 (3,1,2)卷积码的树图

若输入信息位是1101，由树图可得编码输出为111 110 010 100，图5-15中的虚线所示。

4. 网格图

将状态图在时间上展开，可以得到网格图，如图5-16所示。

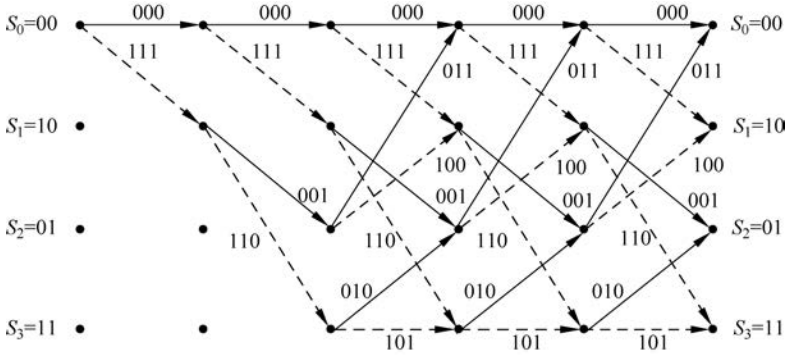


图5-16 (3,1,2)卷积码的网格图

进一步可以得到(3,1,2)卷积码路径图，如图5-17所示。

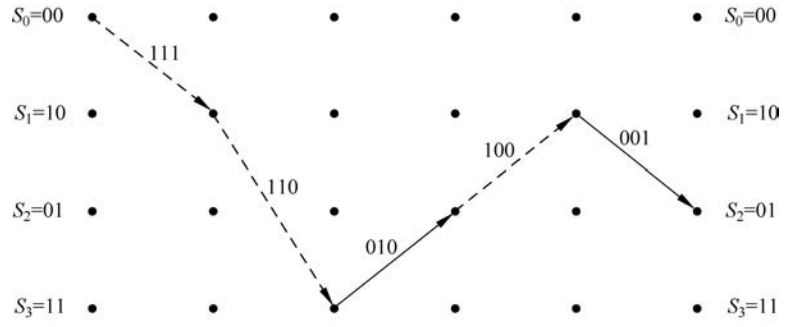


图5-17 (3,1,2)卷积码路径图

5. Viterbi译码

这种算法的基本原理是将接收到的信号序列和所有可能的信号序列作比较，选择其中汉明距离最小的序列译为发送信号序列。

前面已知发送信息位为1101，为了使移位寄存器中的信息全部移出，在信息位后面加入3个“0”，编码后的发送序列为111 110 010 100 001 011 000，若接收序列为111 010 010 110 001 011 000，则第4个码元和第11个码元为错码。

由于这是一个$(n,k,m)=(3,1,2)$卷积码，发送序列的约束长度为$N=m+1=3$，所以首先要考虑3个信息段，即考察$3n=9$位。

由图5-16可知，沿路径每一级有4种状态，S_0、S_1、S_2和S_3，每种状态只有两条路径可以到达，故4种状态共有8条路径。比较这8条路径和接收序列之间的汉明距离，如表5-5所示。

表 5-5 Viterbi 算法解码第一步计算结果

序 号	对应序列	汉明距离	幸存否
1	000 000 000	5	否
2	111 001 011	3	是
3	000 000 111	6	否
4	111 001 100	4	是
5	000 111 001	7	否
6	111 110 010	1	是
7	000 111 110	6	否
8	111 110 101	4	是

将距离较小的路径保存(若几条路径的汉明距离相同,则可以任意保存一条)为幸存路径,如图 5-18(a)所示。

第二步继续考察接收序列中后继的“110”,如图 5-18(b)所示。

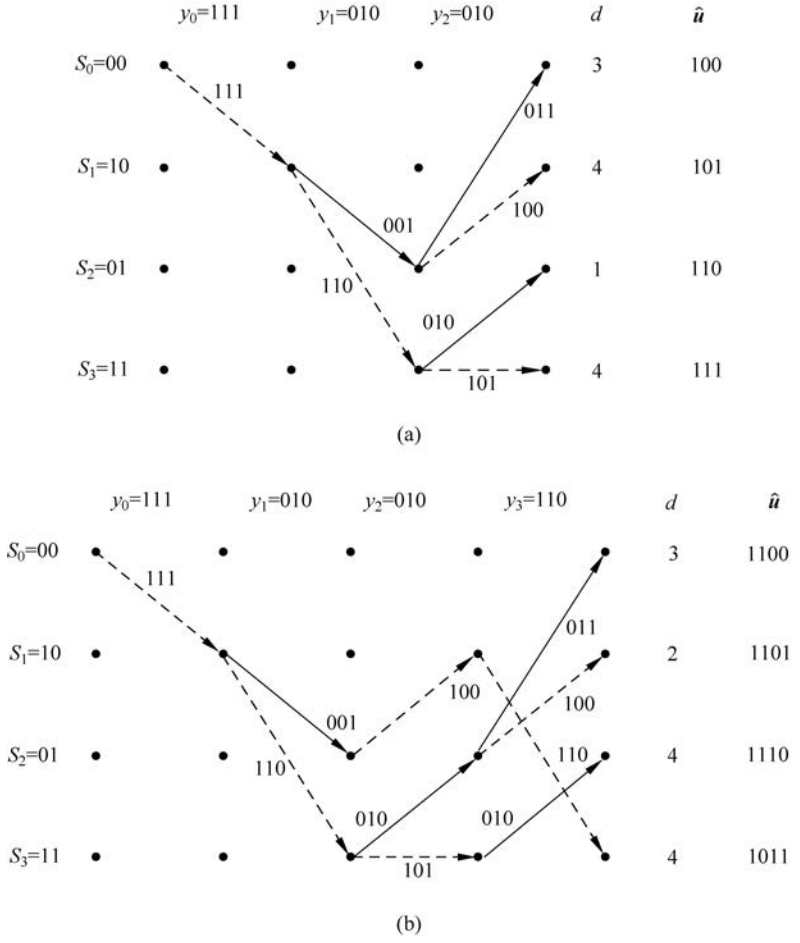


图 5-18 (3,1,2)卷积码译码幸存路径网格图

在编码时，为了使输入的信息位全部通过移位寄存器，使移位寄存器回归到初始状态，在信息位后面加入 3 个“0”。若把这 3 个“0”仍看做信息位，则可以按照上述的方法继续解码，如图 5-19 所示。

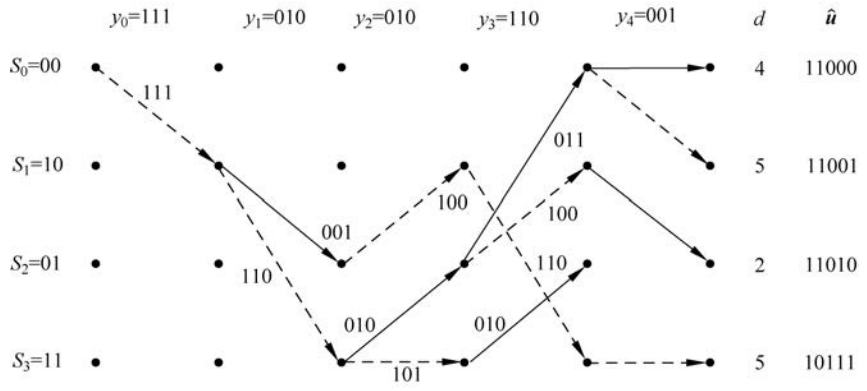


图 5-19 (3,1,2)卷积码译码幸存路径图(补“0”码)

最终得到的幸存路径网格图如图 5-20 所示。

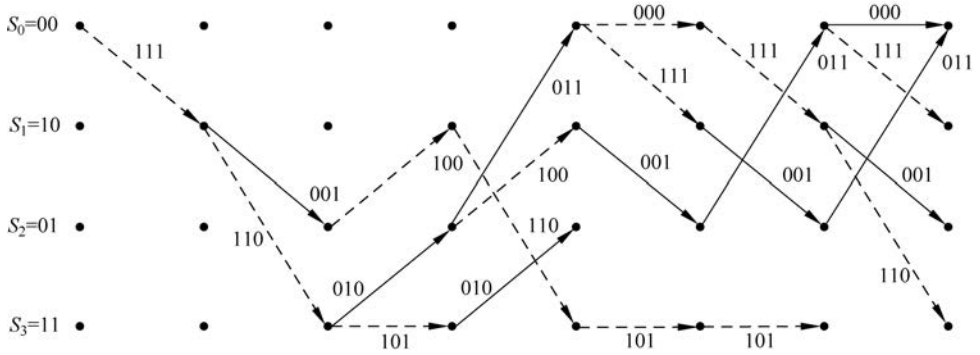


图 5-20 对应信息位 1101000 的幸存路径网格图

已知最后 3 个码元是(为结尾而补充的)“0”，则在解码计算时就预先知道在接收这 3 个“0”码元后，路径必然应该回归到状态 S_0，由图 5-20 可见，只有两条路径可以回到状态 S_0，所以，这时的图 5-20 可以简化为图 5-21。

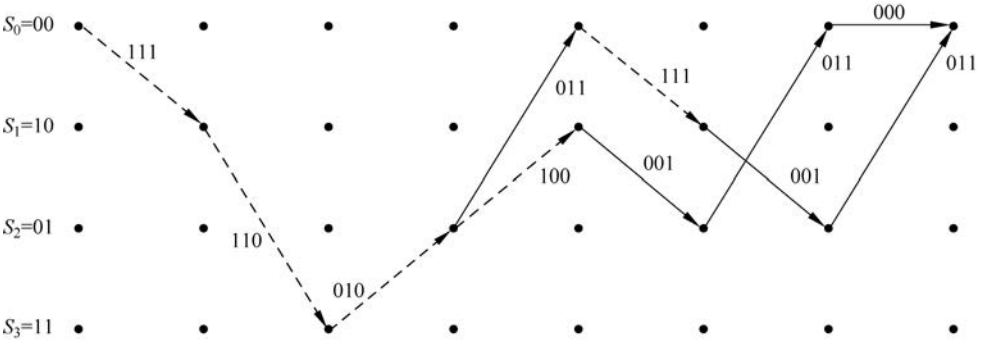


图 5-21 对应信息位 1101 及以 000 结束的幸存路径网格图

因为接收序列为 111 010 010 110 001 011 000，而幸存的两条路径的序列分别为 111 110 010 011 111 001 011 和 111 110 010 100 001 011 000，对应的码距分别为 8 和 2，因此最佳估值为 111 110 010 100 001 011 000，相应的信息序列估值为 1101000。

在该例中卷积码的约束长度为 $N=3$，需要存储和计算 8 条路径的参量。由此可见，Viterbi 算法的复杂度随约束长度 N 按指数形式 2^N 增长。故 Viterbi 算法适合约束长度较小的编码。

5.7 卷积码的编码实现与仿真

5.7.1 与卷积码有关的几个函数

1. convenc 函数

功能：卷积编码。

语法：code = convenc(msg,trellis);

code = convenc(msg,trellis,puncpat);

code = convenc(msg,trellis,…,init_state);

[code,final_state] = convenc(…);

说明：用 code=convenc (msg, trellis)可对 k 位信息进行卷积编码，trellis 是网格结构。

code = convenc(msg,trellis,puncpat)中，puncpat 是指定一个打孔模式，允许更高的编码速率。

code = convenc(msg,trellis,…,init_state)中，init_state 是寄存器的初始状态。

[code,final_state] = convenc(…)同时输出编码和寄存器的最终状态。

2. poly2trellis 函数

功能：将卷积码多项式转换为网格形式。

语法：trellis = poly2trellis(ConstraintLength,CodeGenerator);

trellis = poly2trellis(ConstraintLength,CodeGenerator, FeedbackConnection);

说明：用 trellis = poly2trellis(ConstraintLength,CodeGenerator)输入卷积编码器的多项式描述，输出相应的网格结构。ConstraintLength 是约束长度，等于 $m+1$，CodeGenerator 是生成矩阵。

trellis = poly2trellis(ConstraintLength,CodeGenerator, FeedbackConnection)可输出网格结构，参数 FeedbackConnection 是反馈连接。

3. vitdec 函数

功能：利用 Viterbi 算法的卷积译码。

语法：msg = vitdec(code,trellis,tblen,opmode,dectype);

msg =vitdec(code,trellis,tblen,opmode,'soft',nsdec);

msg ＝vitdec(code,trellis,tblen,opmode,dectype,puncpat);
msg ＝vitdec(code,trellis,tblen,opmode,dectype,puncpat,eraspat);
msg ＝vitdec(…,'cont',…,initmetric,initstates,initinputs);
[decoded,finalmetric,finalstates,finalinputs] ＝ vitdec(…,'cont',…);

说明：tblen 是一个正整数，表示反馈深度，也称回溯长度。

opmode 指在假设编码器模式的情况下解码器的模式。当 opmode＝'cont'时，编码器是全零的初始状态，这种方式有延迟；当 opmode＝'term'时，编码器是全零的初始状态和最终状态；当 opmode＝'trunc'时，这种方式没有延迟。

dectype 指判决方式。当 dectype＝'unquant'时，非量化判决，输出为实数；当 dectype＝'hard'时，硬判决，输出 0 或 1；当 dectype＝'soft'时，软判决，输出 0～(2^b-1)之间的整数，b 是软判决的位数。

5.7.2 编码

1. 利用库函数来实现编码

对于图 5-3 的(2,1,3)卷积码编码器，$k=1$，$n=2$，$m=3$，已知

$$\begin{cases} c_j^1 = u_j \oplus u_{j-2} \oplus u_{j-3} \\ c_j^2 = u_j \oplus u_{j-1} \oplus u_{j-2} \oplus u_{j-3} \end{cases}$$

则 $\boldsymbol{g}^1=(1011)$，$\boldsymbol{g}^2=(1111)$，如果信息序列为已知，如 $\boldsymbol{u}=[1\ 0\ 1\ 1\ 1]$。

利用 convenc 函数，则代码如下：

```
msg = [1 0 1 1 1];                          % 信息序列
trellis = poly2trellis([4],[13 ,17]);       % 产生上述(2,1,3)卷积码编码器的网格
code = convenc(msg,trellis);                % 卷积编码
```

运行结果如下：

```
code =
    1    1    0    1    0    0    0    1    0    1
```

注：此运行结果只包含信息序列的编码。

2. 离散卷积编码的实现方法

代码如下：

程序 5_1(program5_1.m)

```
% 离散卷积编码的实现方法
u = [1 0 1 1 1];                            % 信息序列
g1 = [1 0 1 1];
g2 = [1 1 1 1];                             % g1,g2 长度必须相同,为编码器的两个冲击响应
c1 = conv(u,g1);                            % 卷积运算
c2 = conv(u,g2);                            % 卷积运算
% ---------------------------------
% 交替读出两个卷积编码器的输出数据,得到编码结果
```

```
len = length(c1);
for i = 1:1:len
    output((2*i-1)) = rem(c1(i),2);
    output(2*i) = rem(c2(i),2);
end
%------------------------------------
%结果显示
disp('卷积编码为:')
disp(output);
```

运行结果如下：

```
卷积编码为:
    1   1   0   1   0   0   0   1   0   1   0   1   0   0   1   1
```

3. 利用自定义函数来实现编码

自定义函数 jjmencode.m，代码如下：

```
function code = jjmencode(G,k,msg)
%卷积码编码函数
% G: 决定输入序列的生成矩阵
% k: 每一时钟周期输入编码器的比特数
% msg: 输入数据
% code: 输入数据
%------------------------------------
%判断输入信息序列是否需要添零,若需要则添零
if rem(length(msg),k)>0
  msg = [msg,zeros(size(1:k-rem(length(msg),k)))];
end
%------------------------------------
% 把输入信息比特按 k 分组,m 为所得的组数
m = length(msg)/k;
%------------------------------------
% 检查生成矩阵 G 的维数是否和 k 一致
if rem(size(G,1),k)>0
  error('Error,G is not of the right size.')
end
%------------------------------------
% 从生成矩阵 G 可得到约束长度 L 和输出比特数 n
L = size(G,2)/k;
n = size(G,1);
%------------------------------------
% 在信息前后加零,使移位寄存器清零,加零个数为(L-1)k 个
u = [zeros(1,(L-1)*k),msg,zeros(1,(L-1)*k)];
%------------------------------------
%将添零后的输入序列按每组 Lk 个分组,分组是按 k 比特增加
%从 1 到 Lk 比特为第一组,从 1+k 到 Lk+k 为第二组,依次类推,并将分组按倒序排列
u1 = u(L*k:-1:1);
for i = 1:m+L-2
    u1 = [u1,u((i+L)*k:-1:i*k+1)];
```

```
end
U = reshape(u1,L * k,m + L - 1);
% ----------------------------------
% 卷积编码输出
code = reshape(rem(G * U,2),1,n * (L + m - 1));
```

对于前面的例子，利用自定义函数 jjmencode. m 实现编码。

程序 5_2（program5_2. m）

```
msg = [1 0 1 1 1];                          % 信息序列
g1 = [1 0 1 1];
g2 = [1 1 1 1];
k = 1;                                      % 每一时钟周期输入编码器的比特数
code = jjmencode([g1; g2],k,msg);           % 调用卷积编码函数
disp('卷积编码为');
disp(output);
```

运行结果如下：

```
卷积编码为:
    1   1   0   1   0   0   0   1   0   1   0   1   0   0   1   1
```

5.7.3 译码

对于图 5-3 的(2,1,3)卷积码编码器，$k=1$，$n=2$，$m=3$，已知

$$\begin{cases} c_j^1 = u_j \oplus u_{j-2} \oplus u_{j-3} \\ c_j^2 = u_j \oplus u_{j-1} \oplus u_{j-2} \oplus u_{j-3} \end{cases}$$

则 $\boldsymbol{g}^1=(1011)$，$\boldsymbol{g}^2=(1111)$。

1. 利用库函数(vitdec)来实现译码

采用 vitdec 函数进行译码，执行以下 3 行代码：

```
r = [0 0 0 0 1 1 0 1 0 0 0 1 0 1 0 1 0 0 1 1];  % 接收序列
trellis = poly2trellis([4],[13 ,17]);           % 网格参数
msg  = vitdec(r,trellis,3,'trunc','hard');      % 没有延时,硬判决
```

运行结果如下：

```
msg =
    0    0    1    0    1    1    1    0    0    0
```

执行以下 3 行代码：

```
r = [0 0 0 0 1 1 0 1 0 0 0 1 0 1 0 1 0 0 1 1];  % 接收序列
trellis = poly2trellis([4],[13 ,17]);           % 网格参数
msg  = vitdec(r,trellis,3,'term','hard');       % 编码器是全零的初始状态和最终状态,硬判决
```

运行结果如下：

```
msg =
```

```
    0    0    1    0    1    1    1    0    0    0
```

执行以下 3 行代码：

```
r = [0 0 0 0 1 1 0 1 0 0 0 1 0 1 0 1 0 0 1 1];  % 接收序列
trellis = poly2trellis([4],[13 ,17]);       % 网格参数
msg  = vitdec(r,trellis,3,'cont','hard');    % 编码器是全零的初始状态,有延迟,硬判决
```

运行结果如下：

```
msg =
    0    0    0    0    0    1    0    1    1    1
```

由以上代码可以看出，译码函数 vitdec 的参数设置不同，译码结果不同。

2. 利用自定义的函数实现译码

自定义函数 viterbidecode. m，代码如下：

```
function [msg,survivor_state,cumulated_metric] = viterbidecode(G,k,r)
% viterbi 译码函数
n = size(G,1);                                  % n: 编码输出端口数量,(2,1,3)中 n = 2
% ------------------------------------
if rem(size(G,2),k)~ = 0                        % 当 G 列数不是 k 的整数倍时
    error('G 和 k 大小不一致')                   % 发出出错信息
end
% ------------------------------------
if rem(size(r,2),n)~ = 0                        % 当输出元素个数不是输出端口的整数倍时
    error('接收序列的形式不对')
end
% ------------------------------------
% 基本参数
N = size(G,2)/k;                                % 得出移位数,即寄存器的个数
M = 2^k;                                        % 信号组成的状态数
statenumber = 2^(k * (N - 1));                  % 寄存器状态数
% ------------------------------------
% 产生状态转移矩阵、输出矩阵和输入矩阵
for j = 0: statenumber - 1 % j 表示当前寄存器组的状态
    for m = 0: M - 1 % m 为由 k 个输入端的信号组成的状态
        [next_state,memory_contents] = nextstate(j,m,N,k); % 调用 nextstate 函数,由寄存器当
% 前状态和输入确定寄存器的下一个状态和寄存器内容
        input(j + 1,next_state + 1) = m;
        branch_output = rem(memory_contents * G',2); % 输出分支
        nstate(j + 1,m + 1) = next_state;           % 下一个状态
        output(j + 1,m + 1) = bin2deci(branch_output);
    end
end
state_metric = zeros(statenumber,2); % state_metric 数组用于记录译码过程在每状态时的汉明距离
depth_of_trellis = length(r)/n;                 % 网格深度
r_matrix = reshape(r,n,depth_of_trellis);       % 信道输出的汉明距离
survivor_state = zeros(statenumber,depth_of_trellis + 1);      % 幸存状态
% ------------------------------------
% 开始无尾信道输出的解码
for i = 1: depth_of_trellis - N + 1
```

```
    flag = zeros(1,statenumber);
    if(i <= N)
        step = 2^(k * (N - i));
    else
        step = 1;
    end
    for j = 0: step: statenumber - 1
        for m = 0: M - 1
            branch_metric = 0;
            binary_output = deci2bin(output(j + 1,m + 1),n);
            for ll = 1: n
                branch_metric = branch_metric + metric(r_matrix(ll,i),binary_output(ll));
            end
% 选择码间距离较小的那条路径,当下一个状态没有被访问时就直接赋值,否则,用比它小的路径将
% 其覆盖
if(( state_metric(nstate(j + 1,m + 1) + 1,2)> state_metric(j + 1,1) + branch_metric)| …
                    flag(nstate(j + 1,m + 1) + 1) == 0 )
                state_metric(nstate(j + 1,m + 1) + 1,2) = state_metric(j + 1,1) + branch_metric;
survivor_state(nstate(j + 1,m + 1) + 1,i + 1) = j;
                flag(nstate(j + 1,m + 1) + 1) = 1;
            end
        end
    end
    state_metric = state_metric(: ,2: - 1: 1);
end
% ------------------------------------
% 开始尾部信道输出的解码
for i = depth_of_trellis - N + 2: depth_of_trellis
    flag = zeros(1,statenumber);
    % 状态数从 statenumber→statenumber/2→ … →2→1
    last_stop = statenumber/(2^(k * (i - depth_of_trellis + N - 2)));
    for j = 0: last_stop - 1
        branch_metric = 0;
        binary_output = deci2bin(output(j + 1,1),n);
        for ll = 1: n
            branch_metric = branch_metric + metric(r_matrix(ll,i),binary_output(ll));
        end
        if( (state_metric(nstate(j + 1,1) + 1,2)> state_metric(j + 1,1) + branch_metric)| …
                flag(nstate(j + 1,1) + 1) =>= 0 )
            state_metric(nstate(j + 1,1) + 1,2) = state_metric(j + 1,1) + branch_metric;
            survivor_state(nstate(j + 1,1) + 1,i + 1) = j;
            flag(nstate(j + 1,1) + 1) = 1;
        end
    end
    state_metric = state_metric(: ,2: - 1: 1);
end
% ------------------------------------
% 从最佳路径中产生解码
% 译码过程可从数组 survivor_state 的最后一个位置向前逐级译码
state_sequence = zeros(1,depth_of_trellis + 1);
state_sequence(1,depth_of_trellis) = survivor_state(1,depth_of_trellis + 1);
for i = 1: depth_of_trellis
    state_sequence(1,depth_of_trellis - i + 1) = survivor_state((state_sequence …
        (1,depth_of_trellis + 2 - i) + 1),depth_of_trellis - i + 2);
```

```
end
% ----------------------------------
% 译码输出
msg_matrix = zeros(k,depth_of_trellis - N + 1);
for i = 1: depth_of_trellis - N + 1
    % 根据数组 input 的定义得出从当前状态到下一个状态的输入信号矢量
    dec_output_deci = input(state_sequence(1,i) + 1,state_sequence(1,i + 1) + 1);
    dec_output_bin = deci2bin(dec_output_deci,k);
    % 将一次译码存入译码输出矩阵 msg_matrix 相应的位置
    msg_matrix(: ,i) = dec_output_bin(k: -1: 1)';
end
msg = reshape(msg_matrix,1,k * (depth_of_trellis - N + 1));    % 译码结果
cumulated_metric = state_metric(1,1);
disp('译码结果为')
disp(msg)
```

本函数中调用的其他自定义函数如下：

(1) nextstate.m。

```
% 由寄存器当前状态和输入确定寄存器的下一个状态和寄存器内容
% current_state 为当前状态,next_state 为下一个状态,memory_contents 为寄存器内容
% input 为输入,binary_input 为二进制输入,binary_state 为二进制状态
function [next_state,memory_contents] = nextstate(current_state,input,L,k)
binary_state = deci2bin(current_state,k * (L - 1));
binary_input = deci2bin(input,k);
next_state_binary = [binary_input,binary_state(1: (L - 2) * k)];
next_state = bin2deci(next_state_binary);
memory_contents = [binary_input,binary_state];
```

(2) deci2bin.m。

```
% 十进制转换为二进制
function y = deci2bin(x,l)
y = zeros(1,l);
i = 1;
while x >= 0 & i <= l
    y(i) = rem(x,2);
    x = (x - y(i))/2;
    i = i + 1;
end
y = y(l: -1: 1);
```

(3) bin2deci.m。

```
% 二进制转换为十进制
function y = bin2deci(x)
l = length(x);
y = (l - 1: -1: 0);
y = 2.^y;
y = x * y';
```

(4) metric.m。

```
% 求汉明距离
```

```
function distance = metric(x,y)
if x == y
    distance = 0;
else
    distance = 1;
end
```

程序 5_3（program5_3.m）

```
%已知G、k、r时利用自定义的Viterbi译码算法函数得出译码结果
G = [1 0 1 1; 1 1 1 1];              %G: 卷积编码矩阵,可以根据自己的需要输入编码矩阵
k = 1;                               %k: 信息源输入端口数
r = [1  1  0  1  0  0  0  1  0  1  0  1  0  0  1  1];          %接收到的序列
[msg,survivor_state,cumulated_metric] = viterbidecode(G,k,r);  %调用译码函数
```

运行结果如下：

```
译码结果为
    1     0     1     1     1
```

5.7.4 通信过程的编程仿真

下面利用 MATLAB 对通信过程进行仿真，以此说明卷积码对系统通信性能的改善，并对不同编码方式的性能进行比较。

1. 卷积码与未编码的性能比较

程序 5_4（program5_4.m）

```
%卷积码与未编码的性能比较
SNR = 0: 1: 12;                                  %信噪比
msg = randi([0,1],1,100000);                     % 输入信息
BER0 = zeros(1,length(SNR));
BER1 = zeros(1,length(SNR));
% ------------------------------------
% 网格结构
trellis = poly2trellis(3,[5 7]);    %由编码器得到,此处由图5-6得,m = 2,g1 = [101],g2 = [111]
% ------------------------------------
%未编码的误码率
for k = 1: length(SNR)
    modbit0 = pskmod(msg,2);                     %调制
    y0 = awgn(modbit0,SNR(k),'measured');        %在传输序列中加入AWGN噪声
    demmsg0 = pskdemod(y0,2);                    %解调
    recode0 = reshape(demmsg0',1,[]);
    [num0,rat0] = biterr(recode0,msg);           %误码计算
    BER0(k) = rat0;
end
% ------------------------------------
%编码的误码率
for k = 1: length(SNR)
    code = convenc(msg,trellis);                 %编码
    modbit1 = pskmod(code,2);                    %调制
```

```
    y1 = awgn(modbit1,SNR(k),'measured');          % 在传输序列中加入 AWGN 噪声
    demmsg1 = pskdemod(y1,2);                      % 解调
    recode1 = reshape(demmsg1',1,[]);
    tblen = 5;                                     % 回溯长度
    decoded1 = vitdec( recode1,trellis,tblen,'cont','hard');                    % 译码
    [num1,rat1] = biterr(double(decoded1(tblen + 1: end)),msg(1: end - tblen));% 误码计算
    BER1(k) = rat1;
end
% ------------------------------------
% 画图
semilogy(SNR,BER0,'b - o',SNR,BER1,'r - s');
xlabel('SNR/dB');
ylabel('BER');
legend('未编码','卷积编码(码率为 1/2)');
title('卷积编码(码率为 1/2)与未编码性能比较');
grid on
```

运行结果如图 5-22 所示。

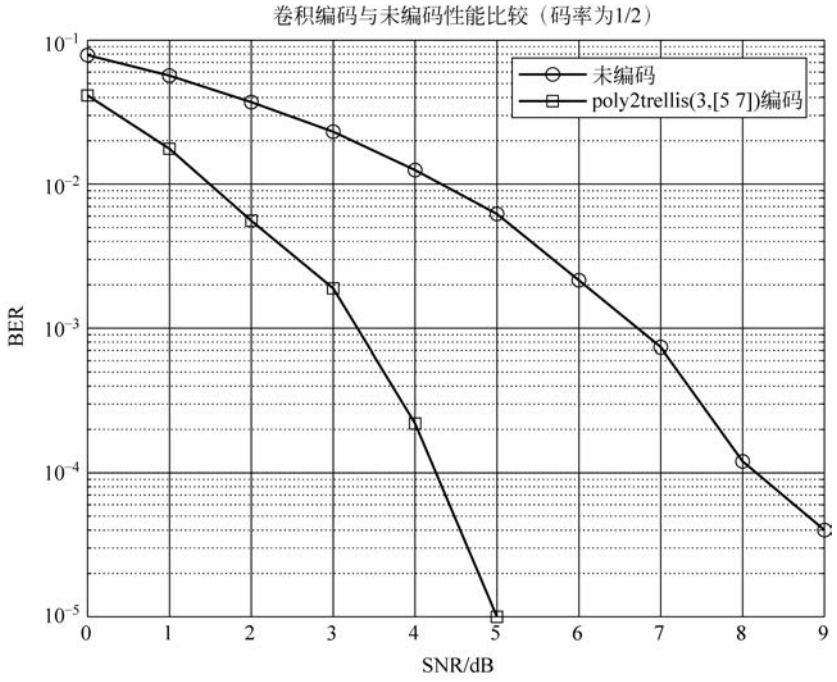


图 5-22 卷积编码与未编码的性能比较

由图 5-22 可见，卷积编码对通信系统的性能有较大的改善，当然对于不同的编码参数，通信系统的性能改善程度不同。

注：每次运行结果都会有差异，为了避免这种情况，可以增加运行次数，然后给出误码率的平均值，再利用平均值得出仿真曲线。

2. 译码回溯深度和长度对卷积码性能的影响

卷积码译码器中有个回溯判决单元，是得到译码信息的核心单元，该单元会根据加比选单元(ACS)时得到的最小状态标号和相应的译码信息，通过回溯的办法得到译码信息。因此回溯长度对卷积码译码性能有一定的影响。译码回溯深度，一般为寄存器个数的4～10倍。因此，回溯长度可以简单地理解为想要开始判决时离最开始计算度量时刻的距离，当然这个距离一般就是比特长度，z

程序 5_5(program5_5.m)

```
%卷积码译码时不同回溯长度的性能比较
clear all;clc;
cycl = 20;                                          %运行次数
SNR = 0:1:10;                                       %信噪比
msg = randi([0,1],1,100000);                        %输入信息
BER0 = zeros(1,length(SNR));
% ------------------------------------------------------------------------
% 网格结构
trellis = poly2trellis(3,[5 7]);                    %移位寄存器的个数为2,约束长度为3
% ------------------------------------------------------------------------
for i = 1:4:21                                      %回溯长度的取值
    for n = 1:cycl
        for k = 1:length(SNR)
            code = convenc(msg,trellis);            %编码
            modbit0 = pskmod(code,2);               %调制
            y0 = awgn(modbit0,SNR(k),'measured');   %在传输序列中加入AWGN噪声
            demmsg0 = pskdemod(y0,2);               %解调
            recode0 = reshape(demmsg0',1,[]);
            tblen = i;                              %回溯长度
            decoded0 = vitdec( recode0,trellis,tblen,'cont','hard'); % 译码
            [num0,rat0] = biterr(double(decoded0(tblen+1:end)),msg(1:end-tblen));
                                                    %误码计算
            BER0(n,k) = rat0;
        end
    end
    BER0 = mean(BER0);
    BER(i,:) = BER0;
end
% ------------------------------------------------------------------------
%画图
figure(1)
semilogy(SNR,BER(1,:),'b-o',SNR,BER(5,:),'r-s',SNR,BER(9,:),'k-+ ',SNR,BER(13,:),'m-p',
SNR,BER(17,:),'b-d',SNR,BER(21,:),'r-*');
xlabel('SNR (dB)');
ylabel('BER');
legend('回溯长度为1','回溯长度为5','回溯长度为9','回溯长度为13','回溯长度为17','回溯长度
为21');
title('译码时不同回溯长度的性能比较');
grid on
```

运行结果如图5-23所示。

由图5-23可见，当卷积码的回溯长度增加时，系统的误码率则越小，卷积码的性能也越

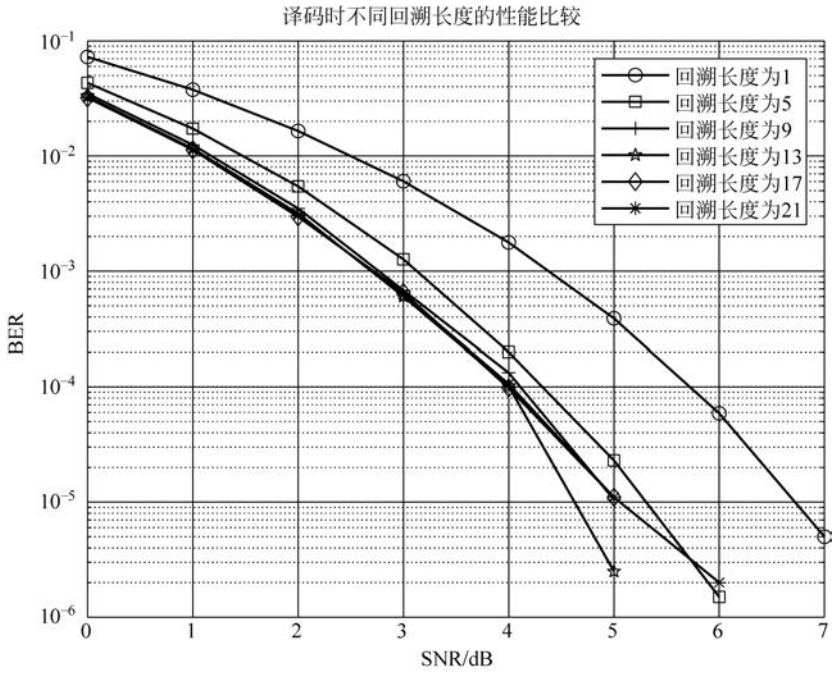


图 5-23 不同回溯长度的卷积编码性能比较

来越好。但是当回溯长度大于 5 倍的编码约束长度时(本仿真采用的编码器约束长度为 3),误码率变化越来越小,逐渐趋于一个稳定值。

3. 未编码、卷积码、汉明码的性能比较

下面就卷积码和汉明码对通信系统性能的改善进行比较。

程序 5_6(program5_6.m)

```
% 未编码、卷积码、汉明码的性能比较
cycl = 50;                                          % 运行次数
SNR = 0: 1: 12;                                     % 信噪比
msg = randi([0,1],1,100000);                        % 输入信息
BER0 = zeros(1,length(SNR));
BER1 = zeros(1,length(SNR));
BER2 = zeros(1,length(SNR));
% ------------------------------------
% 网格结构
trellis = poly2trellis(3,[5 7]);                    % 从电路求出参数,或已知参数
% ------------------------------------
% 未编码的误码率
for n = 1: cycl
for k = 1: length(SNR)
    modbit0 = pskmod(msg,2);                        % 调制
    y0 = awgn(modbit0,SNR(k),'measured');           % 在传输序列中加入 AWGN 噪声
    demmsg0 = pskdemod(y0,2);                       % 解调
    recode0 = reshape(demmsg0',1,[]);
```

```
    [num0,rat0] = biterr(recode0,msg);                    % 误码计算
    BER0(n,k) = rat0;
end
end
BER0 = mean(BER0);
% ------------------------------------
% 卷积编码的误码率
    code = convenc(msg,trellis);                          % 编码
    modbit1 = pskmod(code,2);                             % 调制
for n = 1: cycl
for k = 1: length(SNR)
    y1 = awgn(modbit1,SNR(k),'measured');                 % 在传输序列中加入 AWGN 噪声
    demmsg1 = pskdemod(y1,2);                             % 解调
    recode1 = reshape(demmsg1',1,[]);
    tblen = 5;                                            % 回溯长度
    decoded1 = vitdec(recode1,trellis,tblen,'cont','hard');                    % 译码
    [num1,rat1] = biterr(double(decoded1(tblen + 1: end)),msg(1: end - tblen)); % 误码计算
    BER1(n,k) = rat1;
end
end
BER1 = mean(BER1);
% ------------------------------------
% 汉明编码的误码率
code2 = encode(msg,7,4,'hamming');                        % (7,4)汉明编码
modbit2 = pskmod(code2,2);                                % 调制
for n = 1: cycl
for k = 1: length(SNR) % 编码的序列,调制后经过高斯白噪声信道,再解调制,再纠错后求误码
    y2 = awgn(modbit2,SNR(k),'measured');                 % 在传输序列中加入 AWGN 噪声
    demmsg2 = pskdemod(y2,2);                             % 解调
    recode = reshape(demmsg2',1,[]);
    bitdecoded = decode(recode,7,4,'hamming');            % 译码
    % ------------------------------------
    % 计算误码率
    error2 = (bitdecoded~ = msg);
    errorbits = sum(error2);
    BER2(n,k) = errorbits/length(msg);
end
end
BER2 = mean(BER2);
% ------------------------------------
% 画图
semilogy(SNR,BER0,'b - o',SNR,BER1,'r - s',SNR,BER2,'k - + ');
xlabel('SNR/dB');
ylabel('BER');
legend('未编码','卷积编码(码率为 1/2)','汉明编码');
title('未编码、卷积编码(码率为 1/2)与汉明码性能比较');
grid on
```

运行结果如图 5-24 所示。

由图 5-24 可知，此处比较结果是卷积码的性能较好。也可以加入循环码进行比较。值

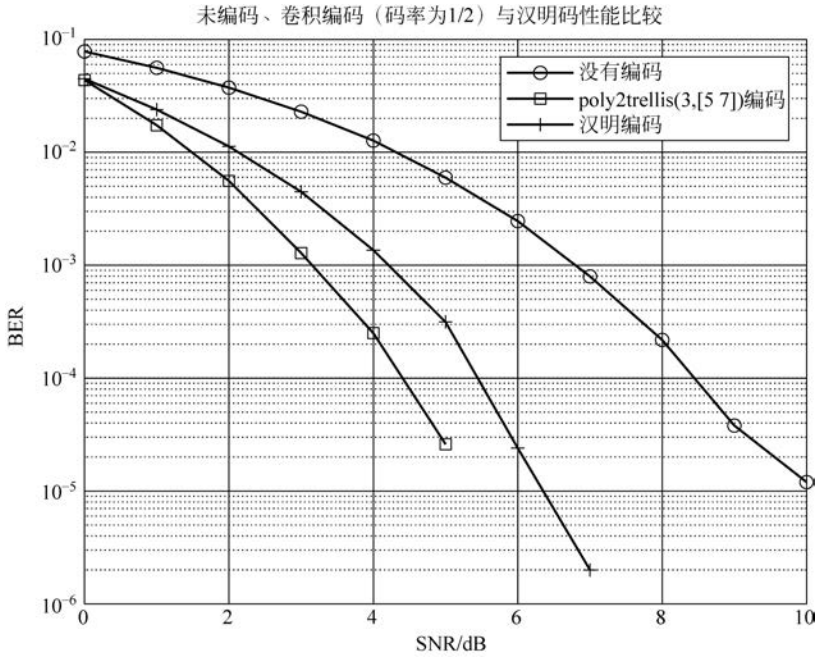
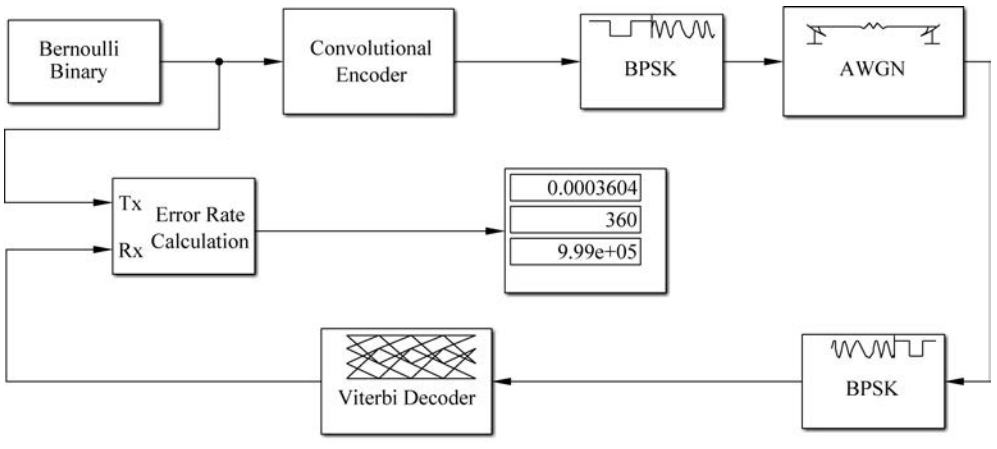


图 5-24 未编码、卷积编码(码率为 1/2)与汉明码性能比较

得注意的是,比较结果与具体的编码方式以及编码采用的参数有关,对于编码性能的好坏不能轻易下结论。

5.7.5 Simulink 仿真

(1) 利用 Display 模块显示仿真结果,网格结构为 poly2trellis([5,4],[23,35,0; 0,05,13]) 采用 AWGN 信道时,卷积码系统仿真模型(jjm_awgn_1.slx)如图 5-25 所示。



jjm_awgn_1_V

图 5-25 AWGN 信道卷积码系统仿真模型 1

图 5-25 中各模块参数设置如图 5-26～图 5-29 所示。

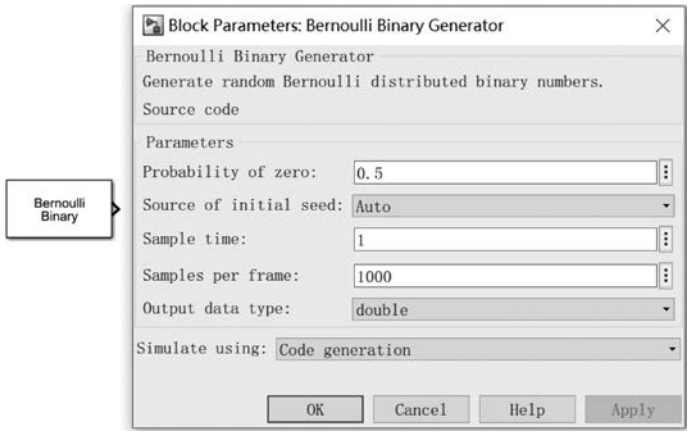


图 5-26　Bernoulli Binary Generator 模块参数

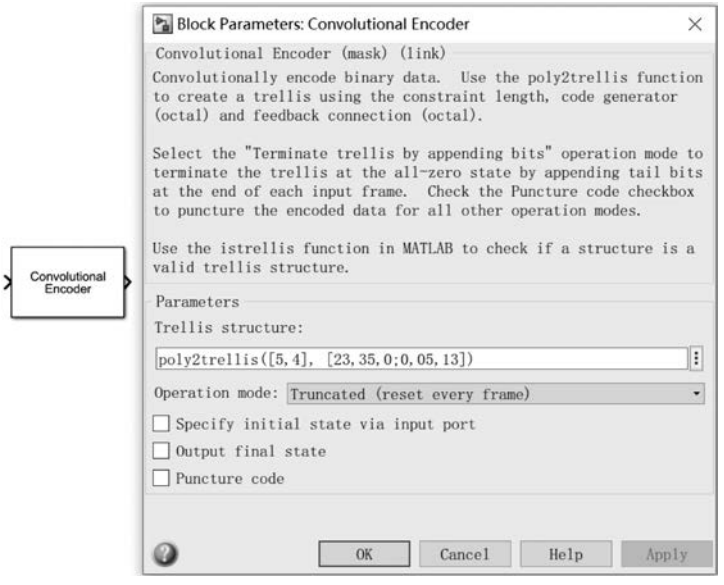


图 5-27　Convolutional Encoder 模块参数

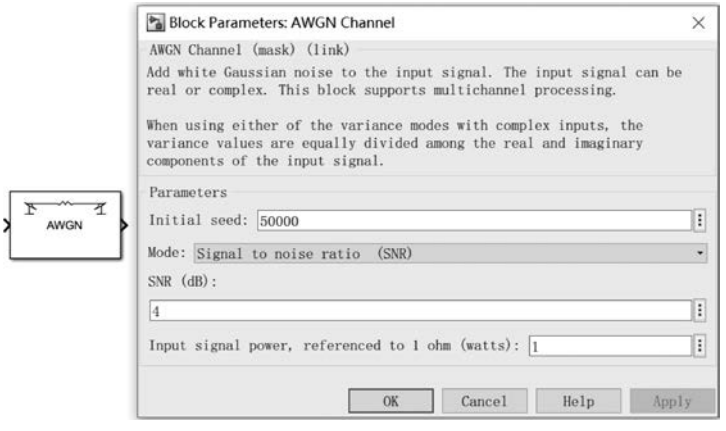


图 5-28　AWGN Channel 模块参数

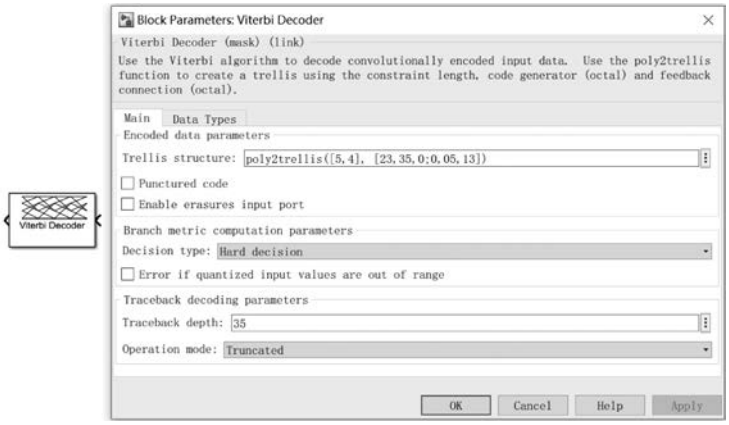
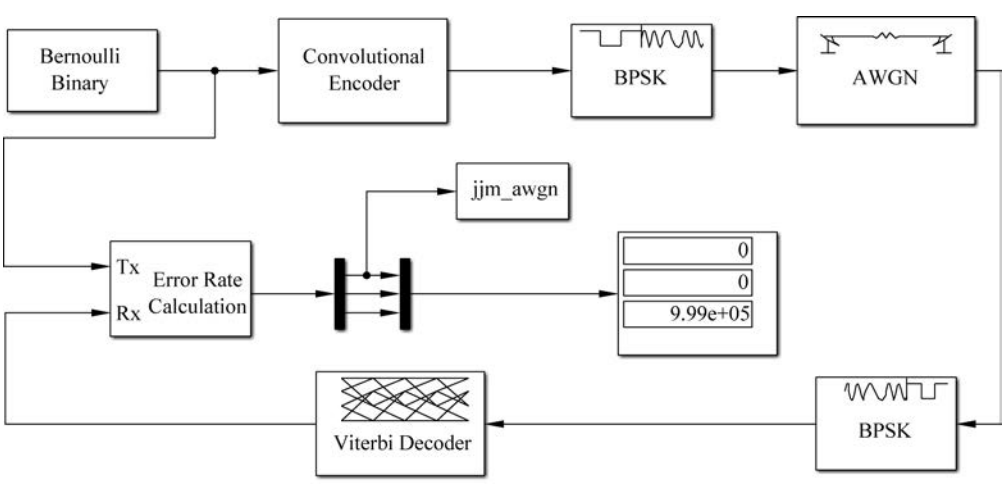


图 5-29 Viterbi Decoder 模块参数

运行图 5-25,显示器的第一行显示的信号误码率为 0.0003604。

(2) 利用曲线显示仿真结果

如果要得到高斯信道中卷积编码误码率与信道信噪比之间的关系曲线,卷积码系统仿真模型(jjm_awgn_2.slx)如图 5-30 所示。



jjm_awgn_2_V

图 5-30 AWGN 信道卷积码系统仿真模型 2

图 5-30 中主要模块的参数设置如图 5-31 和图 5-32 所示。

其余模块的参数设置同前。

程序 5_7(program5_7.m)

```
%高斯白噪声信道卷积码性能
SNR = -1:1:8;                                          %信噪比
% ------------------------------------------------------------------------
for n = 1:length(SNR)                                  %误码率计算
    snr = SNR(n);
    sim('jjm_awgn_2')
```

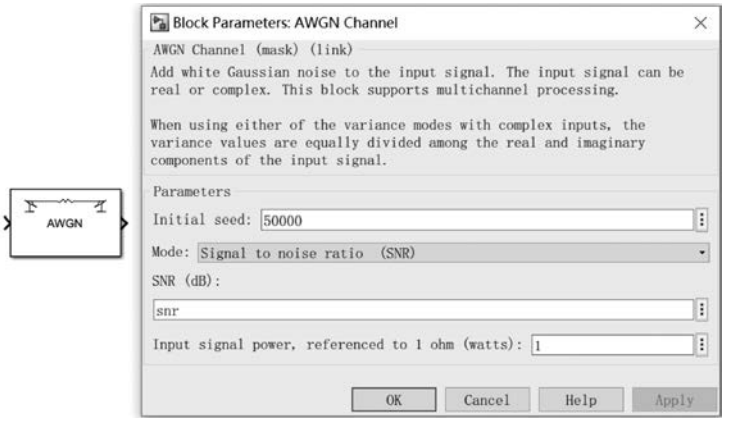


图 5-31 AWGN Channel 模块参数

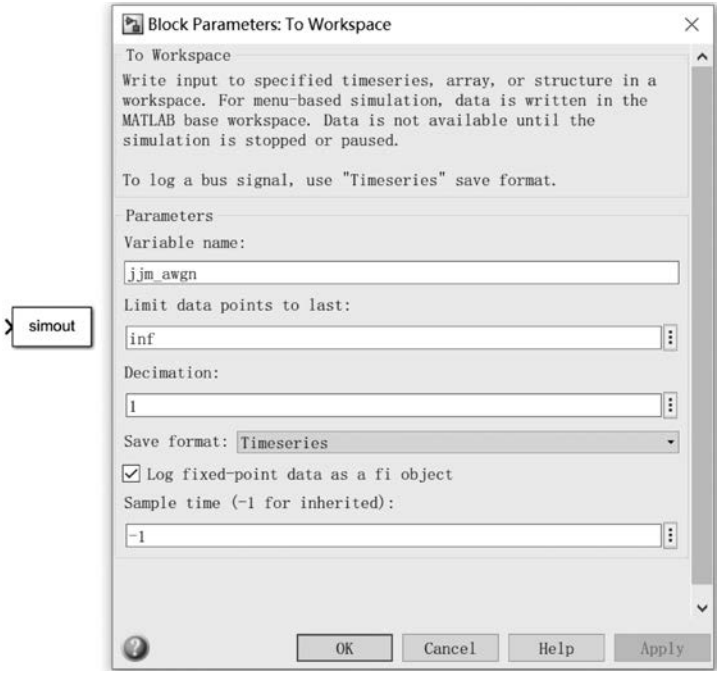


图 5-32 To Workspace 模块参数

```
    S2(n) = [mean(jjm_awgn)]';
    S3(n) = S2(n) + eps;
    EN(n) = [SNR(n)]';
end
 % ------------------------------------------------------------------------
semilogy(EN,S3,'b-*')                                    % 画图
axis([-1,8,1e-7,1]);grid
xlabel('高斯白噪声信道信噪比 SNR(dB)');
ylabel('误码率');
```

```
title('高斯白噪声信道卷积码性能');
```

运行结果如图 5-33 所示。

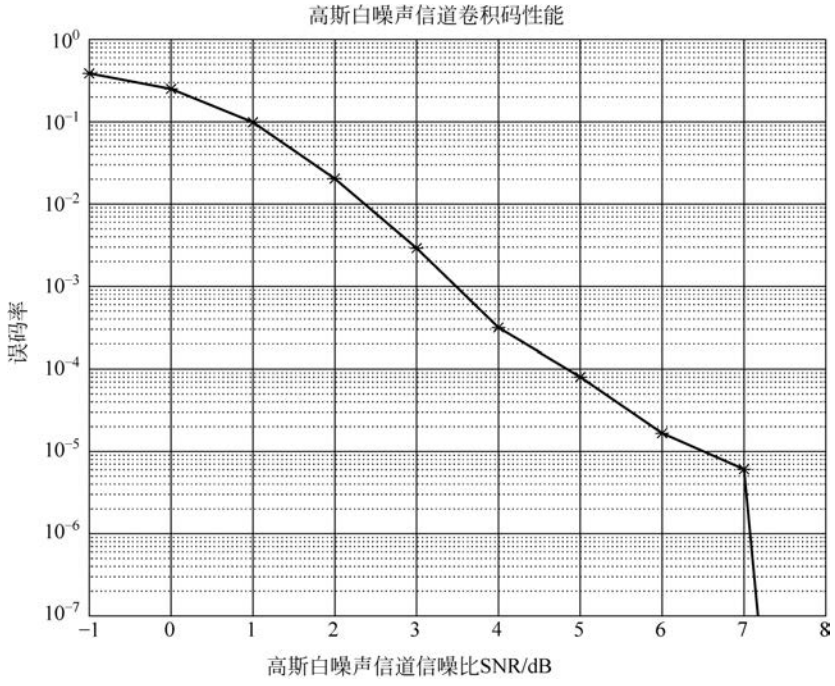


图 5-33 高斯白噪声信道卷积码性能

(3) 几种卷积编码的性能比较

仿真模型如图 5-34 所示(jjm_awgn_bj. slx),卷积码的网格结构分别为 poly2trellis([5,4], [23,35,0; 0,05,13])、poly2trellis([7], [171,133])和 poly2trellis([4], [13, 17]),其中第一种卷积码的码率为 2/3,后两者的码率为 1/2。其余模块的参数设置略。

程序 5-8(program5_8. m)

```
% 三种卷积码编码的性能比较
SNR = -1:1:8;                                          % 信噪比
% ------------------------------------------------------------------------
for n = 1:length(SNR)                                  % 误码率计算
    snr = SNR(n);
    sim('jjm_awgn_bj')
    S1(n) = [mean(jjm_awgn1)]';
    S1(n) = S1(n) + eps;
    EN(n) = [SNR(n)]';
    S2(n) = [mean(jjm_awgn2)]';
    S2(n) = S2(n) + eps;
    S3(n) = [mean(jjm_awgn3)]';
    S3(n) = S3(n) + eps;
end
% ------------------------------------------------------------------------
semilogy(EN,S1,'b-*')                                  % 画图
```

jjm_awgn_bj_V

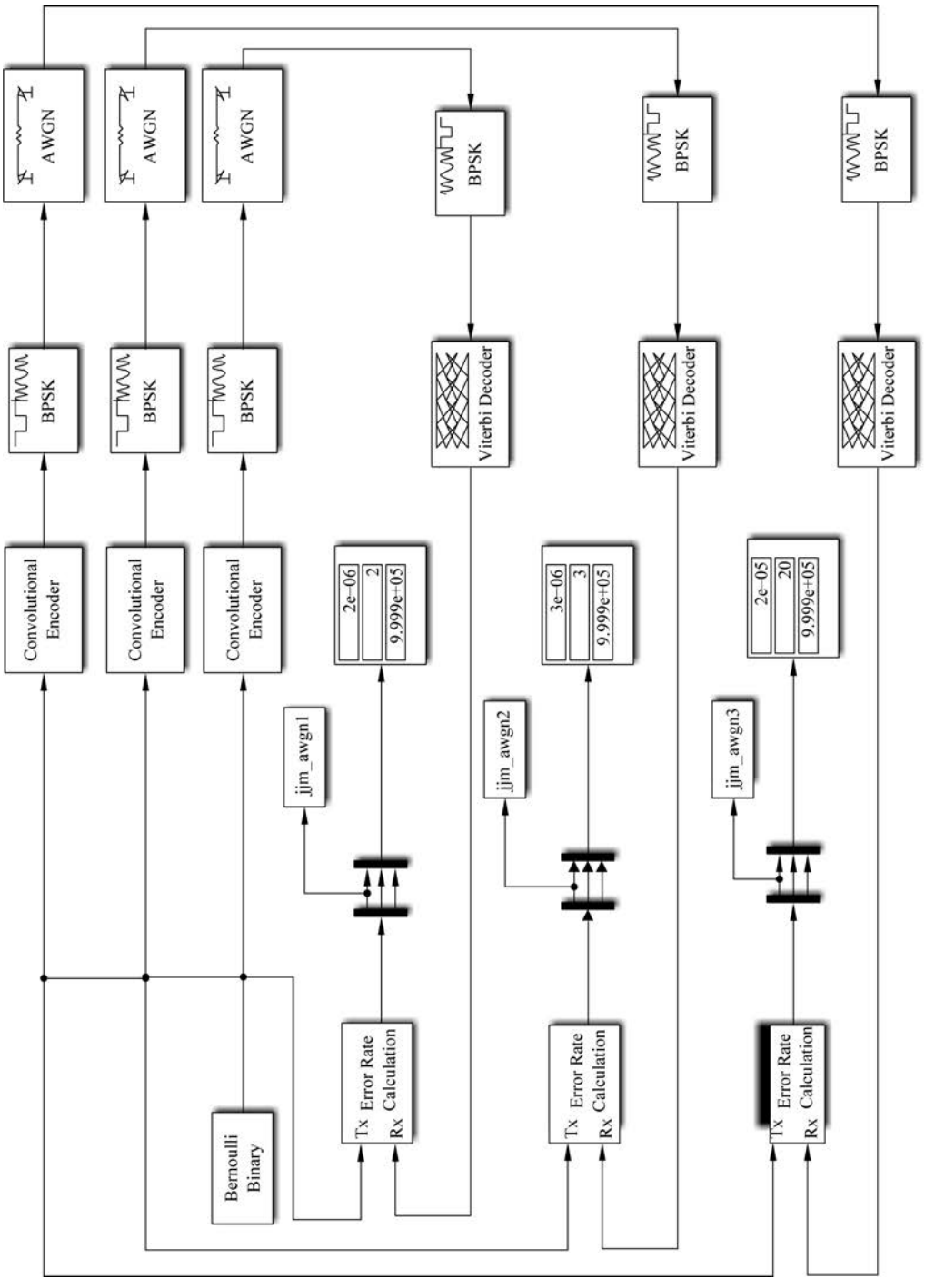


图 5-34 AWGN 信道卷积编码前后比较

```
hold on
semilogy(EN,S2,'r-o')                                    %画图
hold on
semilogy(EN,S3,'k-s')                                    %画图
legend('poly2trellis([4], [13,17])','poly2trellis([7], [171,133])','poly2trellis([5,4], [23,
35,0;0,05,13])');
axis([-1,8,1e-7,1]);grid
xlabel('高斯白噪声信道信噪比 SNR(dB)');
ylabel('误码率');
title('三种卷积编码的性能比较');
```

运行程序结果如图 5-35 所示。

由图 5-35 可见,码率小的码的纠错能力好,而两个码率相同的码的纠错能力也基本一致。

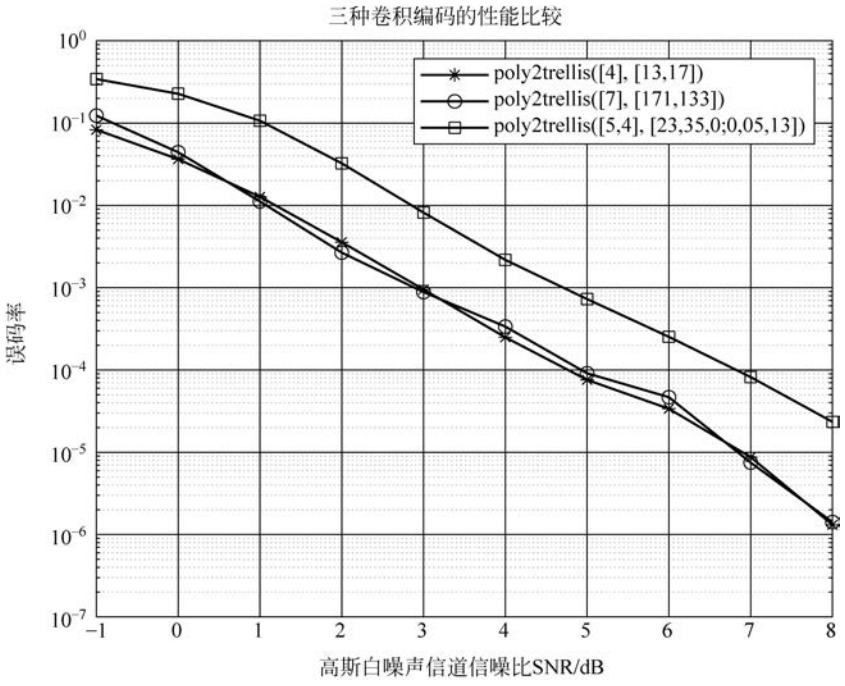


图 5-35 卷积编码性能比较

(4) 卷积码的纠错能力与回溯长度的关系

仿真模型如图 5-36 所示,图 5-36 中 Viterbi Decoder 的参数设置如图 5-37 所示,AWGN 模块的信噪比设置为 4,其余模块参数设置略。

程序 5-9(program5_9.m)

```
%卷积码译码硬判决中的回溯长度对纠错性能的影响
tblen = 1:2:40;                                          %硬判决中的回溯长度
% ------------------------------------------------------------------------
for n = 1:length(tblen)                                  %误码率计算
    tbl = tblen(n);
    sim('jjm_awgn_tbl')
    S2(n) = [mean(err_awgn)]';
```

jjm_awgn_tbl_V

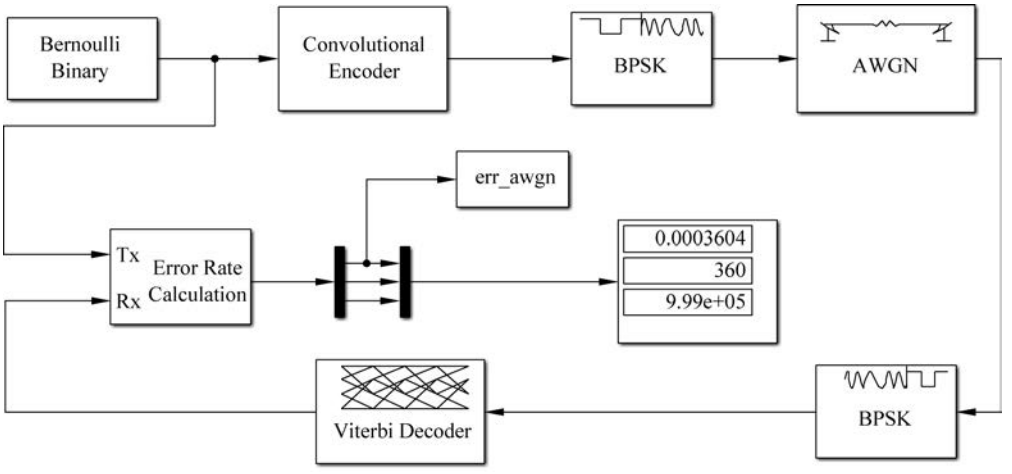


图 5-36　卷积码译码硬判决中的回溯长度对纠错性能的影响

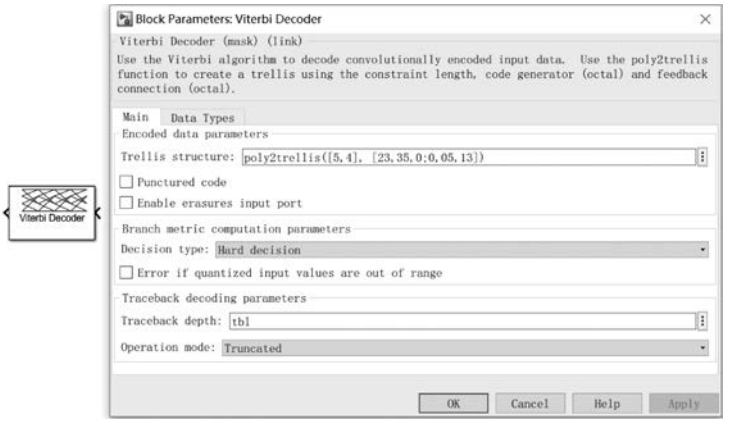


图 5-37　Viterbi Decoder 模块参数

```
    S3(n) = S2(n) + eps;
    TBN(n) = [tblen(n)]';
end
% ------------------------------------------------------------------------
semilogy(TBN,S3,'b-*')                                    %画图
axis([0,40,1e-4,1]);grid
xlabel('回溯长度');
ylabel('误码率');
title('回溯长度与纠错能力的关系');
```

运行程序，结果如图 5-38 所示。由图可见，硬判决的回溯长度会影响卷积码的纠错能力，但是当回溯长度的取值大于某个值以后误码率就不会再降低了。

5.7.6　卷积码编码电路的 Simulink 实现

图 5-3 是一个(2,1,3)卷积码编码器结构，其中 $n=2$，$k=1$，$m=3$。假设输入信息序列为 $u=(10111)$，经过计算得编码器输出为 1101000101，由于是 1/2 码率，所以共有 10 位输出。以此为例来介绍卷积码编码电路的 Simulink 实现。

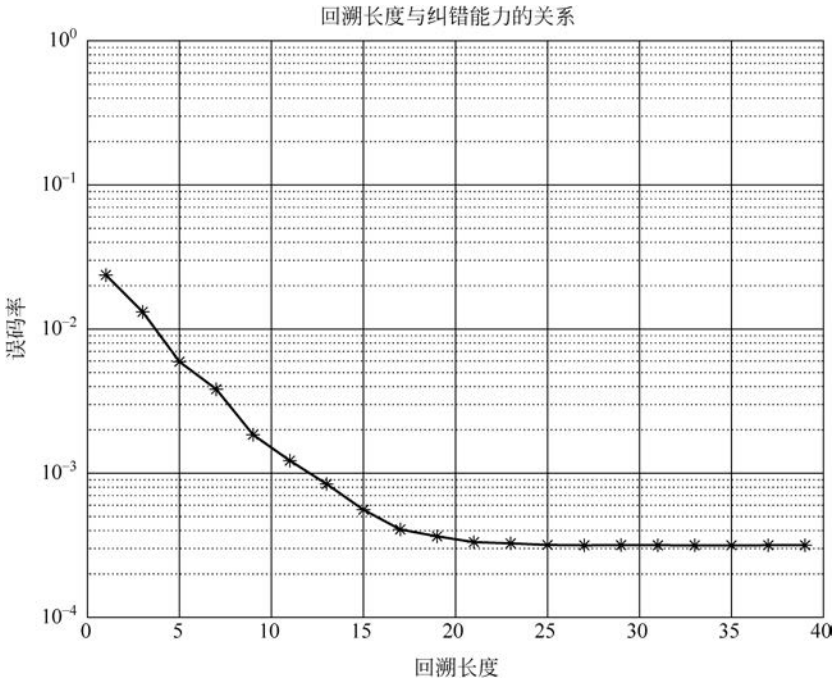
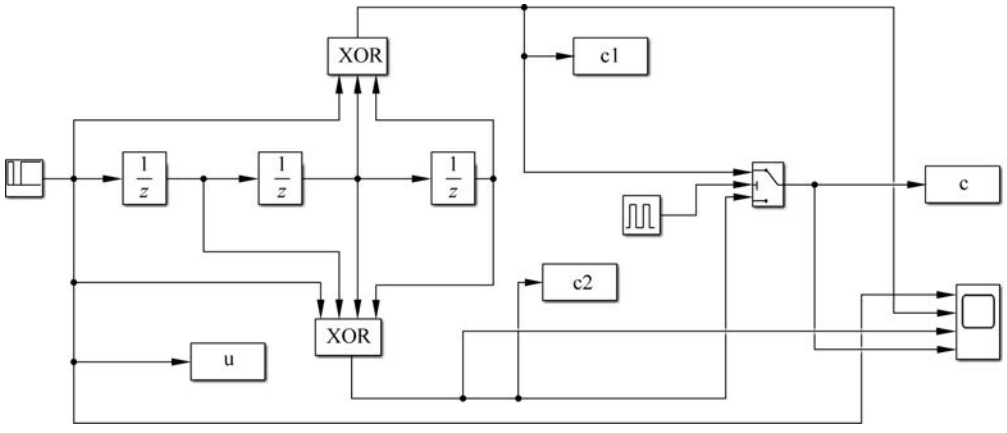


图 5-38 卷积码译码硬判决中的回溯长度对纠错性能的影响

1.（2,1,3）卷积码编码电路的 Simulink 仿真模型

对于图 5-3 给出的（2,1,3）卷积码编码电路，其 Simulink 仿真模型如图 5-39 所示（jjmbmdl_1.slx），这里输入信息序列采用 Repeating Sequence Stair 模块，目的是为了得到确定的输入信息，如这里输入信息序列为 10111，则参数设置如图 5-40 所示，其他模块的参数设置略。



jjmbmdl_1_V

图 5-39 Repeating Sequence Stair 模块参数 1

移位寄存器采用 Unit Delay 模块，模 2 加法器采用 Logical Operator 模块，切换开关采用 Switch 模块。四个 To Workspace 模块用来查看输出序列，一个 Scope 模块用来查看序

列的波形，并进行比较。Switch 模块采用 Pulse Generator 来控制。Repeating Sequence Stair 模块和 Unit Delay 模块的采样时间均设置为 1，参数 u、c1 和 c2 的采样时间也设置为 1。

由于输出序列为上下两个模 2 加法器的交替取值，因此 Switch 模块和参数 c 的采样时间均设置为 0.5，Scope 模块的采样时间也设置为 0.5。

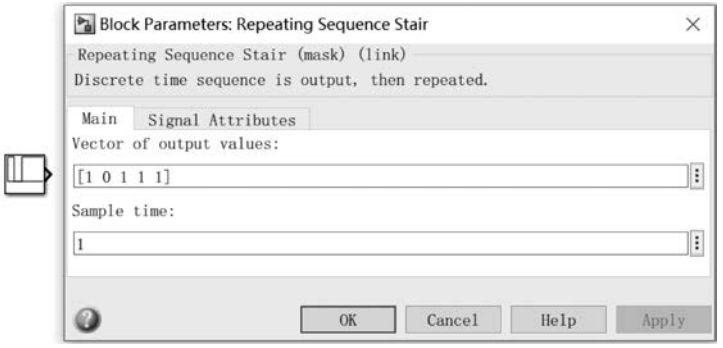


图 5-40　Repeating Sequence Stair 模块参数 2

2. 运行结果

对于图 5-39 所示的仿真模型，仿真模型的运行时间设为 4.99（因为 Matlab 是两边采样，若采样时间设为 1，为了使信息长度为 5，运行时间必须大于等于 4.5，小于 5）。运行仿真模型，仿真结果如图 5-41 和图 5-42 所示。

图 5-42 示波器的显示波形从上到下分别对应于信息序列 u、编码序列 c1、编码序列 c2 以及编码输出序列 c。由图 5-41 和图 5-42 可以看出，该输出结果与前面计算得到的结果完全一致。

```
>> u',c1',c2',c'
ans =
     1     0     1     1     1
ans =
     1     0     0     0     0
ans =
     1     1     0     1     1
ans =
     1     1     0     1     0     0     0     1     0     1
```

图 5-41　命令窗口看到的运行结果

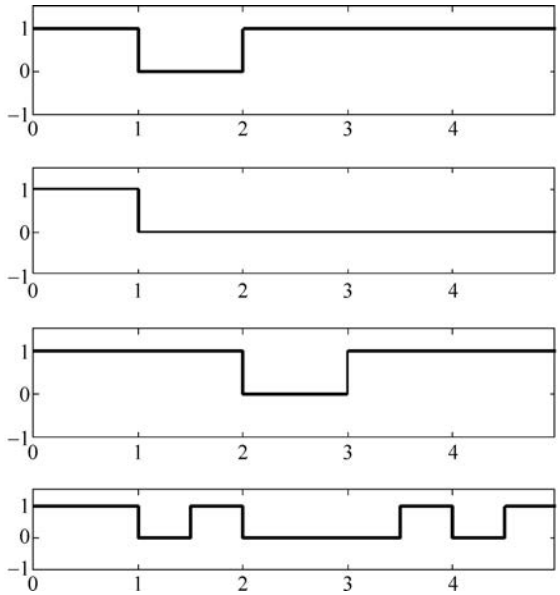
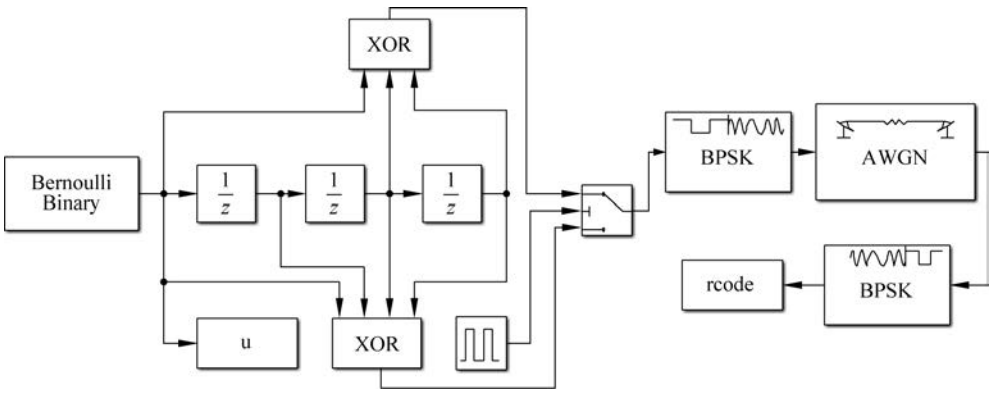


图 5-42 示波器显示的波形

3. 卷积码的通信过程仿真

卷积码通信系统仿真模型如图 5-43 所示(jjmbmdl_2.slx),各模块的参数设置略。这里信源采用 Bernoulli Binary Generator 模块。信源为随机产生的 10000 个二进制码元,通过编码电路后成为 20000 个二进制码元,这些码元经过 BPSK 调制后,送入加性高斯白噪声信道传输,接收端收到信息后先进行解调,然后送入 To Workspace 模块,模块变量名设为 rcode。再利用 MATLAB 编程实现译码,译码调用 vitdec 函数,采用硬判决译码。



jjmbmdl_2_V

图 5-43 卷积码通信系统仿真模型

程序 5_10(program5_10.m)

```
%卷积编码的误码率
SNR = -1:1:5; %信噪比
trellis = poly2trellis(4,[13 17]); %从电路求出参数,或已知参数
```

```
% ------------------------------------------------------------------------
for n = 1:length(SNR) % 误码率计算
    errB = SNR(n);
    sim('jjmbmdl_2')
    recode = permute(rcode,[3,1,2]);
    recode1 = reshape(recode',1,[]);
    tblen = 15; % 回溯长度
    decoded1 = vitdec(recode1,trellis,tblen,'cont','hard'); % 译码
    decoded1 = decoded1';
    xxu = permute(u,[3,1,2]);
    [num1,rat1] = biterr(double(decoded1(tblen + 1:end)),xxu(1:end - tblen)); % 误码计算
    S2(n) = [mean(rat1)]';
    S3(n) = S2(n) + eps;
    EN(n) = [SNR(n)]';
end
% ------------------------------------------------------------------------
semilogy(EN,S3,'b - * ') % 画图
axis([ - 1,5,1e - 5,1]);grid
xlabel('高斯白噪声信道信噪比 SNR(dB)');
ylabel('误码率');
title('高斯白噪声信道卷积码性能');
```

仿真结果如图 5-44 所示。

由图 5-44 可以看出，卷积码的误码率随信道信噪比的增大而减小。当然误码率还与编码方式、译码方式等因素有关。

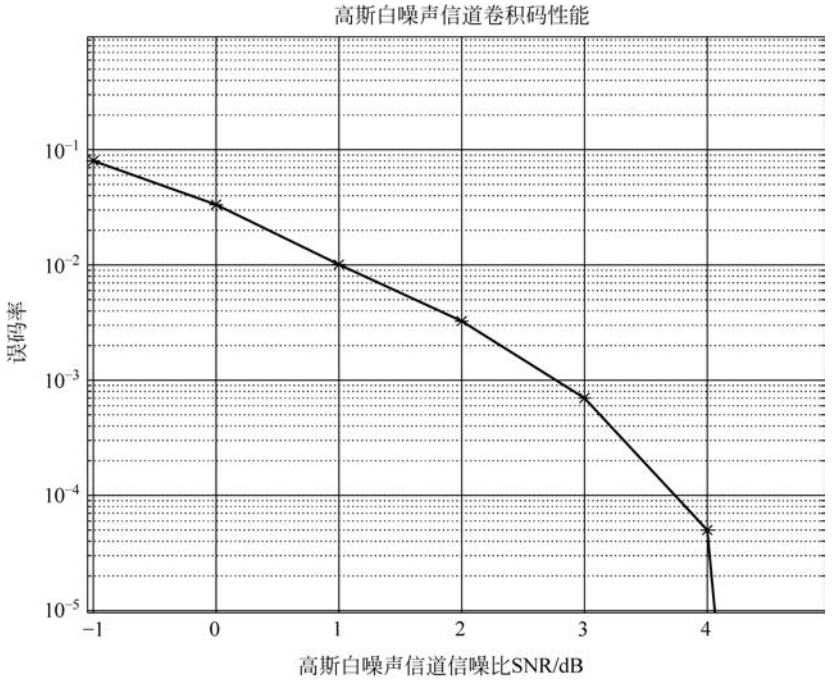


图 5-44 卷积码通信性能

习题

1. Viterbi 译码就是极大似然译码，这种说法对吗？

2. 已知(2,1,3)码，生成序列 $\boldsymbol{g}^1=(1101)$，$\boldsymbol{g}^2=(1111)$。

(1) 求出该码的生成矩阵 $\boldsymbol{G}_\infty$ 和生成多项式 $\boldsymbol{G}(x)$。

(2) 求对应于信息序列 $\boldsymbol{u}=(11101)$的码序列。

(3) 此码是否是系统码？

3. 某(3,1,2)卷积码，生成序列有 $\boldsymbol{g}^1=(100)$，$\boldsymbol{g}^2=(101)$，$\boldsymbol{g}^3=(111)$。

(1) 画出该码编码器。

(2) 写出该码生成矩阵。

(3) 若输入信息序列为 110101，求输出的码序列是多少？

4. 已知(3,1,2)卷积码，生成多项式有 $g^1(x)=1+x+x^2$，$g^2(x)=1+x$，$g^3(x)=1+x^2$。

(1) 画出该码的状态图。

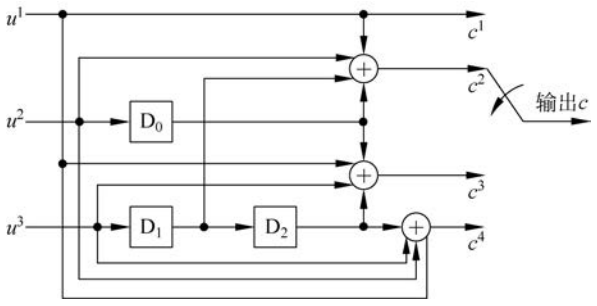
(2) 画出 $l=4$ 的树图和网格图。

(3) 用 Viterbi 译码算法对接收序列 101100001011111101 译码。

5. 已知(3,2,1)卷积码，$g_1^1(x)=1$，$g_1^2(x)=x$，$g_1^3(x)=1+x$，$g_2^1(x)=x$，$g_2^2(x)=1$，$g_2^3(x)=1$，求 $\boldsymbol{u}(x)=[1+x+x^3,1+x^2+x^3]$时的码多项式 $c(x)$。

6. 某卷积码编码器如题图 5-1 所示。

(1) $(n,k,m)=$？ (2) $\boldsymbol{G}=$？ $\boldsymbol{G}(x)=$？ (3) $\boldsymbol{u}=[110\quad 011\quad 101]$时 $c=$？



题图 5-1 卷积码编码器

7. 已知(2,1,2)卷积码编码器的输出与信息位的关系为 $c^1=u_1+u_2$，$c^2=u_1+u_2+u_3$，当接收序列为 1000100000 时，试用 Viterbi 译码算法求解发送信息序列。

8. 设(3,1,2)卷积码的生成多项式为：

$$g_1^1=x^2+x+1,g_1^2=x^2+x+1,g_1^3=x^2+1$$

(1) 试画出该编码的树图、网格图和状态转移图。

(2) 画出编码电路。

(3) 如果输入序列为 1011，求出编码后的序列。

(4) 假设接收序列为 110011001001，根据 Viterbi 译码算法进行译码。

9. 利用 MATLAB 实现题 8(3)。

10. 对于(2,1,3)卷积码编码器，已知

$$\begin{cases} c_j^1 = u_j \oplus u_{j-2} \oplus u_{j-3} \\ c_j^2 = u_j \oplus u_{j-1} \oplus u_{j-2} \oplus u_{j-3} \end{cases}$$

利用 Simulink 搭建仿真模型，要求得出(2,1,3)卷积码在 AWGN 信道时信噪比与误码率的关系曲线。