第5章 差错控制系统的MATLAB/Simulink仿真

差错控制技术是指在发送端加入差错控制码元，即监督码元，利用监督码元和信息码元之间的某种确定关系来进行自动纠错和检错。一般来说，增加的监督码元越多，传输效率就越低，纠错和检错能力反而就越强。所以，差错控制技术是通过降低传输效率来提高信息在通信系统中传输的可靠性。本章介绍几种常用差错控制编码的基本原理，并通过 MATLAB/Simulink 对其进行仿真分析。

5.1 基于线性分组码的差错控制系统仿真

对信源编码器输出的序列进行分组，并对每一组独立变换，称为分组码，记为(n,k)码，其中k表示每分组输入符号数，n为编码输出符号数。编码后的码组具有抗信道干扰的能力。若这种变换是线性变换，则称变换后的码组为线性分组码；若变换是非线性变换，则称变换后的码组为非线性分组码。常用的是线性分组码。

线性分组码具有如下两个性质。

(1) 线性(包含全零码字，封闭性)。

(2) 最小码距等于除零码外的码字的最小码重。

为了具体说明线性分组码的基本原理，以(7,3)线性分组码为例。设(7,3)线性分组码为$\boldsymbol{C}=(c_6,c_5,c_4,c_3,c_2,c_1,c_0)$，其中$c_6,c_5,c_4$为信息位，$c_3,c_2,c_1,c_0$为监督位。将信息流分成每3位为一组，构成原码，即$\boldsymbol{A}=(a_2,a_1,a_0)$，按下列线性方程进行编码：

$$\begin{cases} c_6=a_2 \\ c_5=a_1 \\ c_4=a_0 \\ c_3=a_1\oplus a_0 \\ c_2=a_2\oplus a_1 \\ c_1=a_2\oplus a_1\oplus a_0 \\ c_0=a_2\oplus a_0 \end{cases} \tag{5-1}$$

写成矩阵形式，则可表示为

$$\boldsymbol{C}=[c_6,c_5,c_4,c_3,c_2,c_1,c_0]=[a_2,a_1,a_0]\begin{bmatrix}1&0&0&0&1&1&1\\0&1&0&1&1&1&0\\0&0&1&1&0&1&1\end{bmatrix}=\boldsymbol{A}\cdot\boldsymbol{G} \tag{5-2}$$

式中，$\boldsymbol{A}$为原码；$\boldsymbol{C}$为生成的线性分组码；$\boldsymbol{G}$为生成线性分组码的生成矩阵。一般情况下，生成(n,k)线性分组码的生成矩阵大小为$k\times n$。

由式(5-1)进一步变换可以得到监督位与信息位的关系：

$$\boldsymbol{H}\cdot\boldsymbol{G}^{\mathrm{T}}=\boldsymbol{0}^{\mathrm{T}} \tag{5-3}$$

式中，$\boldsymbol{H}$为监督矩阵。监督矩阵中的每个行向量与线性分组码中的任一码元内积为0，并且其中每个行向量都线性无关。

1. 线性分组码的编码译码仿真

采用 MATLAB 仿真程序完成对(7,4)线性分组码的编码、解码，其中信息位和检验

位的约束关系为 c1＝a1；c2＝a2；c3＝a3；c4＝a4；c5＝a1＋a2＋a3；c6＝a2＋a3＋a4；c7＝a1＋a2＋a4；生成矩阵为 G，校验矩阵为 H，原码为 A，生成码字为 C，纠错后的码字为 Cr。

```
clear all;
G1 = eye(4);                                    % 生成 4×4 单位阵
G2 = [1,0,1;1,1,1;1,1,0;0,1,1];                 % 约束关系
G = [G1,G2];                                    % 生成矩阵 G
fprintf('生成矩阵为:G = ')
disp(G);
A = [0,0,0,1;0,0,1,0;0,0,1,1;0,1,0,0;0,1,0,1;0,1,1,0;0,1,1,1;1,0,0,0;1,0,0,1;1,0,1,0;
1,0,1,1;1,1,0,0;1,1,0,1;1,1,1,0;1,1,1,1;]; % A = [a1,a2,a3,a4]编码的原码
fprintf('原码为:A = ')
disp(A);
C1 = A * G;
C = mod(C1,2);                                  % 模 2 运算
fprintf('输出的编码为:C = ')
disp(C);
H = gen2par(G)                                  % 生成校验矩阵
fprintf('校验矩阵为:H = ')

 %% 以下输入接收到的码字,译出原码
Rev = input('请输入 7 位接收码字,用空格隔开:','s');
Rev = str2num(Rev)                              % 接收到的码字
S1 = Rev * (H');                                % S 为校阵子
S = mod(S1,2);
E = [1,1,1,1,1,1,1];
for i = 1:7;                                    % 取出 H 中的每一列,与 S 相加
  Hi = H(:,[i]);
  Sum = S + Hi';
  Sum = mod(Sum,2);
  if (all(Sum(:) == 0));                        % 如果 S 与 H 的第 i 列之和 Sum 为 0 矩阵,则表示
                                                % Rev 中第 i 个码字有误
      fprintf('接收码字中错误码位是第:');
      disp(i)
  else
      E(1,i) = 0;
  end;
end;
Cr = mod((Rev + E),2);
fprintf('正确接收码字:Cr = ');
disp(Cr);
```

运行结果如下：

```
生成矩阵为:
G =
    1     0     0     0     1     0     1
    0     1     0     0     1     1     1
    0     0     1     0     1     1     0
    0     0     0     1     0     1     1
原码为:
```

```
A =
     0     0     0     1
     0     0     1     0
     0     0     1     1
     0     1     0     0
     0     1     0     1
     0     1     1     0
     0     1     1     1
     1     0     0     0
     1     0     0     1
     1     0     1     0
     1     0     1     1
     1     1     0     0
     1     1     0     1
     1     1     1     0
     1     1     1     1
输出的编码为:
C =
     0     0     0     1     0     1     1
     0     0     1     0     1     1     0
     0     0     1     1     1     0     1
     0     1     0     0     1     1     1
     0     1     0     1     1     0     0
     0     1     1     0     0     0     1
     0     1     1     1     0     1     0
     1     0     0     0     1     0     1
     1     0     0     1     1     1     0
     1     0     1     0     0     1     1
     1     0     1     1     0     0     0
     1     1     0     0     0     1     0
     1     1     0     1     0     0     1
     1     1     1     0     1     0     0
     1     1     1     1     1     1     1
校验矩阵为:
H =
     1     1     1     0     1     0     0
     0     1     1     1     0     1     0
     1     1     0     1     0     0     1
请输入7位接收码字,用空格隔开:0 1 1 1 1 0 1
接收码字中错误码位是第:     2
正确码字为:Cr =     0     0     1     1     1     0     1
```

2. 线性分组码信号的传输仿真

采用 MATLAB 和 Simulink 交互的方式仿真实现线性分组码编码后的信号在通信系统中的传输。Simulink 仿真模型如图 5-1 所示。模型中 Bernoulli Binary Generator(伯努利二进制序列产生器)模块产生的信源序列经过 Binary Linear Encoder(二进制线性编码器)进行线性分组码编码。编码后的序列经过 Binary Symmetric Channel(二元对称信道)传输,该信道具有误码概率。在接收端进行译码。译码后的序列和信源序列输入 Error Rate Calculation(误码率统计)模块,统计接收端误码率。

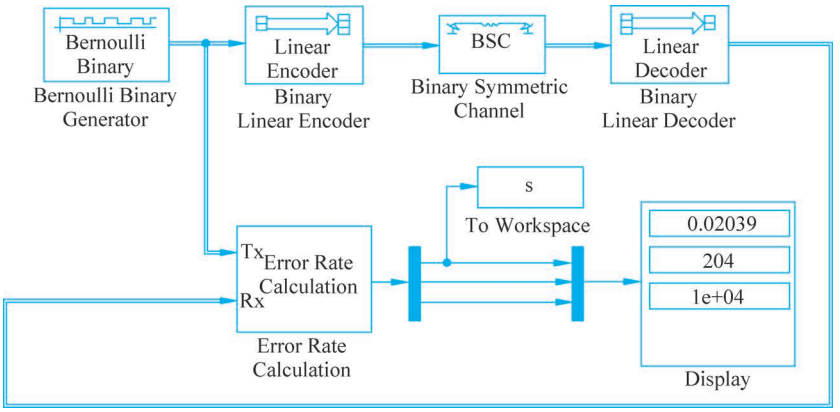


图 5-1　线性分组码的 Simulink 仿真模型

模型中各模块的主要参数设置见表 5-1。

表 5-1　线性分组码的 Simulink 仿真参数

模 块 名 称	参 数 名 称	参　数　值
Bernoulli Binary Generator	Probability of zero	0.5
	Initial seed	10000
	Sample time	1
	Frame-based output	Checked
	Samples per frame	4
Binary Linear Encoder	Generator matrix	[[1 1 0; 0 1 1; 1 1 1; 1 0 1]eye(4)]
Binary Symmetric Channel	Error probability	errB
	Initial seed	2137
Error Rate Calculation	Receive delay	0
	Computation delay	0
	Computation mode	Entire frame
	Output data	port
To Workspace	Variable name	s
	Limit data point to last	inf
	Decimation	1
	Sample time	−1
	Save format	Array

本例仿真采用 MATLAB 和 Simulink 交互的方式，分析经过线性分组码编码的信号在不同误码率的信道中传输后，接收端收到信号的误码率情况，代码如下：

```
Clear;
x = 0:0.01:0.05;
y = x;
hold off;
fori = 1:length(x)
```

```
    errB = x(i);
    sim('m');                                  % 实现 MATLAB 和 Simulink 模块的交互
    y(i) = mean(s);
end
plot(x,y);
hold on
grid on;
xlabel('信道误码率');
ylabel('接收端误码率 Pe ');
```

图 5-2 为线性分组码的 MATLAB 和 Simulink 交互仿真结果。通过对图中信道误码率和接收端信号误码率的分析，可以看出使用线性分组码的差错控制仿真编码以后，差错率明显下降，比如信道误码率在 0.04 时，接收端信号误码率不足 0.015。

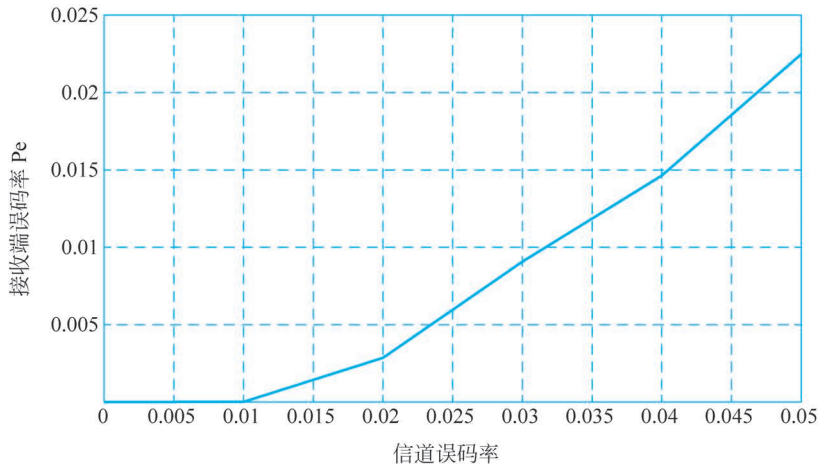


图 5-2　线性分组码的传输仿真结果

5.2　基于循环码的差错控制系统仿真

设 $\boldsymbol{C}$ 是某(n,k)线性分组码的码字集合，如果将任一码字 $c=(c_{n-1},c_{n-2},\cdots,c_0)$向左移一位，记为 $c^{(1)}=(c_{n-2},c_{n-3},\cdots,c_0,c_{n-1})$，也属于码集 $\boldsymbol{C}$，则该线性分组码为循环码。循环码的特点是具有循环性，即任何许用码字的循环移位仍然是一个许用码字。

对于一个任意长为 n 的码字 $c=(c_{n-1},c_{n-2},\cdots,c_1,c_0)$，用多项式的形式表示，即 $c(x)=c_{n-1}x^{n-1}+c_{n-2}x^{n-2}+\cdots+c_1x+c_0$，此多项式即为码多项式，系数不为 0 的 x 的最高次数称为多项式 $c(x)$的次数或阶数。在进行码多项式简单运算时，所得系数需进行模 2 运算。这就是循环码的编码原理。

构造循环码主要是要找出一组线性分组码的生成矩阵，将信息位与生成矩阵 $\boldsymbol{G}$ 相乘得到码字。由于循环码的码字多项式是生成多项式 $g(x)$的倍式，且根据线性码生成矩阵的特性，(n,k)码的生成矩阵可以由(n,k)码 k 个不相关的码组构成。根据以上两点，可以挑选出 k 个线性不相关的循环码的码多项式。

接收端检测时,用码多项式 $y(x)$除以生成多项式 $g(x)$。若不能除尽,则说明接收码不属于循环码,在传输过程中发生错误,即$\frac{y(x)}{g(x)}=Q(x)+\frac{R(x)}{g(x)}$。因此,可用其余式 $R(x)$是否为零来判断接收码组中是否有错。但当码字中错误位数超过循环码的检错能力时,接收码多项式仍有可能被 $g(x)$整除,利用以上方法则不能检测出误码。

1. 循环码的编码仿真

下面的程序完成(7,4)循环码的编码。代码中 cyclpoly(n,k,'all')返回(n,k)循环码的所有生成多项式(1 个生成多项式为返回矩阵的 1 行);cyclgen(n,g)返回循环码的监督矩阵和生成矩阵,其中 g 为生成多项式向量;rem(msg * G,2)返回循环码的所有需用码组,其中 G 为生成矩阵,msg 为信息矩阵。

```
clear all;
close all;
n = 7;
k = 4;
p = cyclpoly(n,k,'all');                    % 产生循环码的生成多项式
[H,G] = cyclgen(n,p(1,:));                  % 产生循环码的生成矩阵和校验矩阵
Msg = [0 0 0 0;0 0 0 1;0 0 1 0;0 0 1 1;0 1 0 0;0 1 0 1;0 1 1 0;0 1 1 1;
    1 0 0 0;1 0 0 1;1 0 1 0;1 0 1 1;1 1 0 0;1 1 0 1;1 1 1 0;1 1 1 1 ];
C = rem(Msg * G,2)
```

运行结果如下:

```
C =
     0     0     0     0     0     0     0
     0     1     1     0     0     0     1
     1     1     0     0     0     1     0
     1     0     1     0     0     1     1
     1     1     1     0     1     0     0
     1     0     0     0     1     0     1
     0     0     1     0     1     1     0
     0     1     0     0     1     1     1
     1     0     1     1     0     0     0
     1     1     0     1     0     0     1
     1     1     0     1     0     0     1
     0     1     1     1     0     1     0
     0     0     0     1     0     1     1
     0     1     0     1     1     0     0
     0     0     1     1     1     0     1
     1     0     0     1     1     1     0
     1     1     1     1     1     1     1
```

2. 循环码信号的传输仿真

循环码的 Simulink 仿真模型如图 5-3 所示。Bernoulli Binary Generator 模块产生的信源序列经过 Binary Cyclic Encode 编码后在 Binary Symmetric Channel 中传输。Error Rate Calculation 模块将接收端 Binary Cyclic Decode 译码后的序列和信源序列输入进行比较,统计误码率。

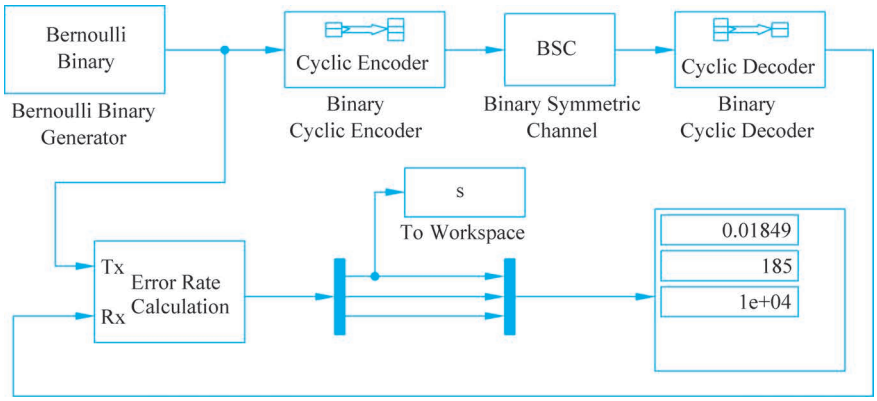


图 5-3 循环码的 Simulink 仿真模型

表 5-2 是模型中各模块的主要参数设置。

表 5-2 循环码的 Simulink 仿真参数

模块名称	参数名称	参数值
Bernoulli Binary Generator	Probability of zero	0.5
	Source of initial seed	Auto
	Sample time	1
	Samples per frame	4
	Output data type	double
	Simulate using	解释执行
Binary Cyclic Encoder	Codeword length N	7
	Message length K	4
Binary Symmetric channel	Error probability	errB
	Initial seed	2137
	Simulate using	代码生成
Binary Cyclic Decoder	Codeword length N	7
	Message length K	4
Error Rate Calculation	Receive delay	0
	Computation delay	0
	Computation mode	Entire frame
	Output data	port
To Workspace	Variable name	s
	Limit data point to last	inf
	Decimation	1
	Sample time	−1
	Save format	数组

本例仿真同样采用 MATLAB 和 Simulink 交互的方式，完成不同信道误码率下接收端的误码率统计，代码如下：

```
clear all;
xval = 0:0.01:0.05;
yval = xval;
hold off;
for i = 1:length(xval)
    errB = xval(i);
    sim('book_cyc');
    yval(i) = mean(s);
end
plot(xval,yval);
hold on;
grid on;
title('循环码在 BSC 信道中的传输性能');
 xlabel('信道误码率');
 ylabel('接收端误码率 Pe');
```

图 5-4 是仿真结果图。从图中可以看出,循环码同样降低了接收端的误码率,提高了通信质量。

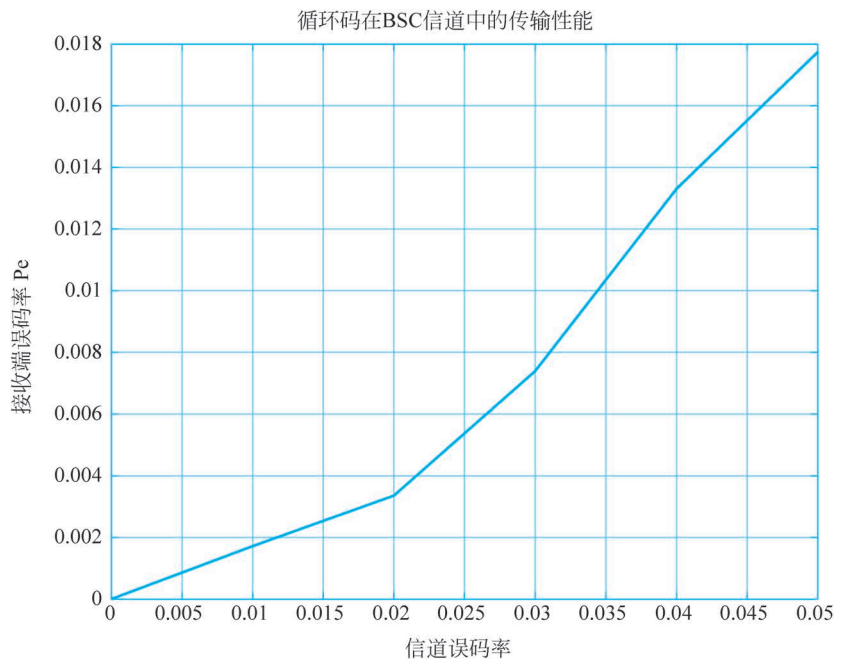


图 5-4 循环码的传输仿真结果

5.3 基于卷积码的差错控制系统仿真

5.3.1 卷积码简介

卷积码是结合当前信息组和之前信息组之间的关系进行编码的。卷积码通常记作(n,k,L)的形式,其中 n 是输出码字的长度,k 为一组信息的长度,L 是与当前输出码字

相关的前信息组的数量。卷积码当前的输出码字不仅与当前输入的信息组有关，还与之前的 L 个信息组有关，因此，卷积码的约束长度为 $L+1$。约束长度越长，纠错能力就越强，但码率会因此降低。卷积码的码率是 $R=k/n$。图 5-5 是(3,2,1)卷积码的一种结构，i 表示当前时刻，$i-1$ 表示上一时刻。

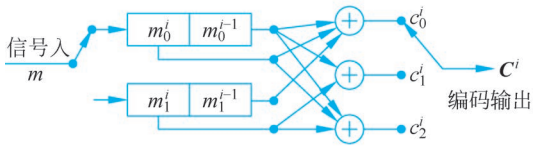


图 5-5　卷积码编码器结构

卷积码中 k 和 n 的值通常比较小，延时也相应小，适合串行传输。

下面利用 MATLAB 和 Simulink 两种方式进行仿真，对输入信号进行卷积编码，经过 AWGN 信道传输，接收端解码后统计传输误码率。

5.3.2　卷积码的编译码和传输仿真

1. 卷积码的 MATLAB 仿真

下面采用(2,1,9)卷积码。代码利用 randi 函数产生信源序列，经过 convenc 卷积编码后，进行 BPSK 调制并在 AWGN 信道中传输。卷积码生成多项式为 G0=561(八进制)，G1=753(八进制)。接收端 BPSK 解调后利用软判决滑动窗维特比译码，译码深度为 40。

```
clear all;
close all
SNRdB = 0:0.5:3;                          % 设置信噪比范围,0～3dB
declen = 40;                              % 译码深度
SNRnum = length(SNRdB);                   % 信噪比数目
iter = 10;                                % 每个信噪比下的迭代次数
for i = 1:SNRnum                          % 循环内计算每个信噪比下的误码率
  for j = 1:iter                          % 每个信噪比下迭代计算误码率 10 次,再求
                                          % 平均误码率
    trel = poly2trellis(9,[561 753]);     % 卷积码(2,1,9)网格图,约束长度为 9
    siglen = 1000000;                     % 设置信号长度
    msg = randi([0,1],siglen,1);          % 生成 0、1 序列,长度同信号长度
    encode = convenc(msg,trel,0);         % 从 0 状态开始做卷积编码
    I = 0.5 * ones(siglen * 2,1);
    y = encode - I;
    bpsk = sign(y);                       % BPSK 调制
    channelout = awgn(bpsk,SNRdB(i));     % 添加高斯白噪声,AWGN 信道
    debpsk = channelout * 0.5 + 0.5;      % 解调
    parti = 0:.15:.9;                     % 设置量化等级划分
    codebk = 0:7;                         % 设输出等级
    [x,qcode] = quantiz(debpsk,parti,codebk); % 量化,准备维特比软判决
      decode = vitdec(qcode',trel,declen,'cont','soft',3); % 维特比译码,量化级数 = 2^3
      % 计算本次 BER
      [errorbit,errorrate(j)] = biterr(decode(declen + 1:end),msg(1:end - declen));
  end
```

```
    BER(i) = sum(errorrate)/iter;                    % 求平均 BER
end
semilogy(SNRdB,BER);                                 % 绘制不同信噪比下的误码率
xlabel('信噪比 SNR(dB)');
ylabel('误码率');
grid on;
```

图 5-6 为卷积码的 MATLAB 仿真结果。从图中可以看出,接收端误码率随着信道信噪比的提高迅速降低。

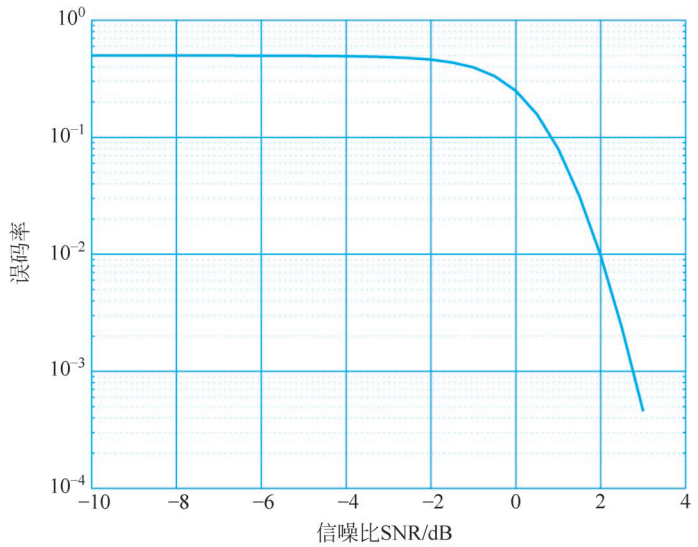


图 5-6　卷积码的 MATLAB 仿真结果

2. 卷积码的 Simulink 仿真

图 5-7 是卷积码的 Simulink 仿真模型。信源序列由 Bernoulli Binary Generator 模块产生,经过 Convolution Encoder 编码后进行 BPSK 调制,输入 AWGN 信道。接收端

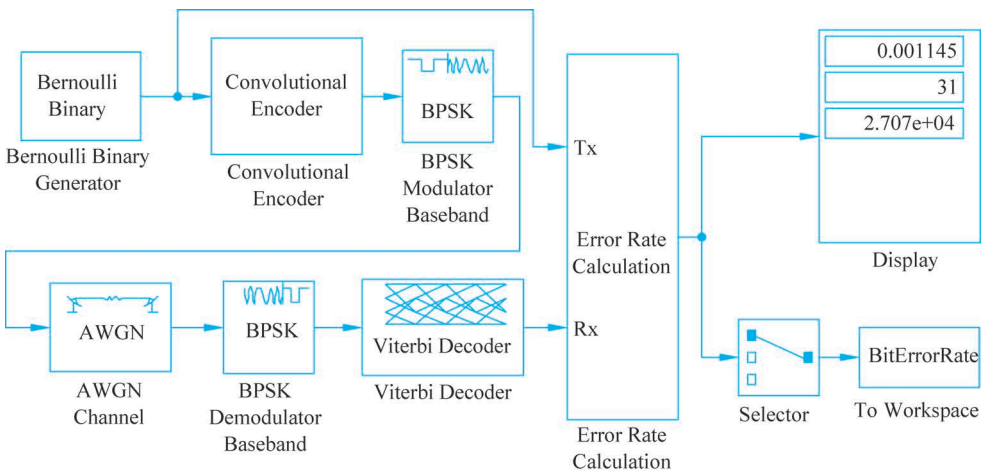


图 5-7　卷积码的 Simulink 仿真模型

将信号 BPSK 解调后，采用 Viterbi Decoder 译码。译码后的序列和信源序列一起输入 Error Rate Calculation 模块统计误码率。

表 5-3 是卷积码的 Simulink 仿真参数设置。

表 5-3 卷积码的 Simulink 仿真参数

模块名称	参数名称	参数值
Bernoulli Binary Generator	Probability of zero	0.5
	Source of initial seed	Parameter
	Initial seed	67
	Sample time	0.02/268
	Samples per frame	268
	Output data type	double
	Simulate using	解释执行
BPSK Modulator Baseband（BPSK 调制）	Phase offset(rad)	0
	Data Types	double
AWGN Channel（加性高斯白噪声信道）	Initial seed	67
	Mode	Signal to noise ratio(SNR)
	SNR(dB)	SNR
	Input signal power(watts)	1
Convolution Encoder	Trellis structure	STRUCTURE
	Operation mode	Truncated(reset every frame)
Selector(数据选通器)	输入维数	1
	索引模式	从 1 开始
	端口大小	3
	Index	[1]
Viterbi Decoder	Trellis structure	STRUCTURE
	Decision type	Hard Decision
	Operation mode	Truncated
	Traceback depth	34

本例通过 MATLAB 和 Simulink 交互的方式，完成不同码率、不同信噪比下接收端误码率的计算，代码如下：

```
x = -10:2;                                  % x 表示信噪比
% 卷积方式分别取 1/3 卷积和 1/2 卷积
A = [poly2trellis(9, [557 663 711]),poly2trellis(7, [171 133])];
% 不同卷积方式、信噪比下重复调用图 5-7 的 Simulink 模型
for j = 1: length(A)
    STRUCTURE = A(j);
        for i = 1:length(x)
        SNR = x(i);                         % 信道的信噪比依次取 x 中的元素
        sim('book_sim_juanjima');           % 实现 MATLAB 和 Simulink 模块的交互
        y(j,i) = mean(BitErrorRate);        % 计算 BitErrorRate 的均值
    end
end
semilogy(x,y(1,:),'r',x,y(2,:),'b')         % 绘图，采用对数坐标
```

```
title('卷积码在 AWGN 信道的传输性能');
xlabel('信道信噪比'),ylabel('接收端误码率');
legend('1/3 卷积码','1/2 卷积码');
grid on
```

图 5-8 是卷积码的 MATLAB 和 Simulink 交互仿真结果。图中两条线分别表示不同码率的卷积码在不同信噪比下的误码率性能。

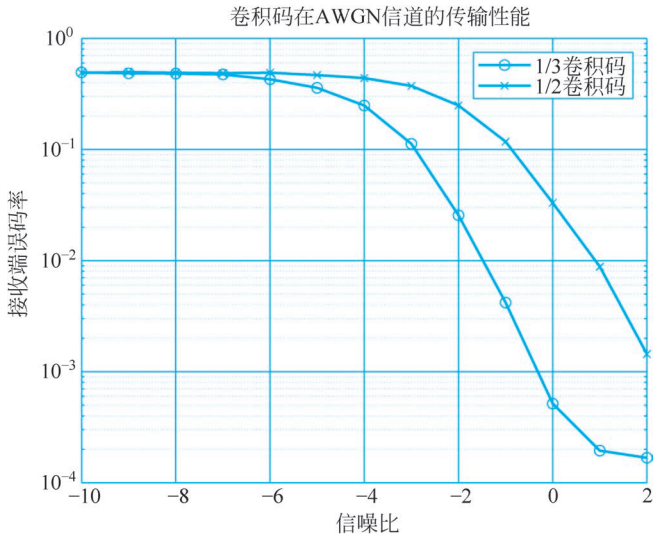


图 5-8　卷积码的仿真结果

5.4　基于循环冗余码的差错控制系统仿真

5.4.1　循环冗余码简介

舍去(n,k)线性分组码中所有首位为 1 的码字，再将所有剩余码字的首位 0 舍去，组成新的$(n-1,k-1)$码，称为缩短码。由于保留的均是首位为 0 的码，舍去它们的第一位不会改变码的最小重量，因此缩短码与原码具有相同的最小距离 $d_{\min}$。

循环冗余校验码就是一种系统的缩短码。这种码的信息位 k 和码长 n 可变，但校验位长度 $n-k$ 固定，符合$(n-i,k-i)$缩短循环码的特点。因此，以一个选定的(n,k)循环码为基础，改变 i 值，可得出任意信息长度的码字，满足实际中对 n、k 取值的多样性要求，而纠检错能力保持不变。

循环冗余码是最常见的校验码，由信息位和校验位两部分组成。其编码方法如下。

(1) 移位：将 k 比特原码左移 r 位，形成 $k+r=n$ 位。

(2) 相除：用生成多项式 $g(x)$，以模 2 除的方式去除移位后的式子，得到的余数就是校验码。

接收端收到数据后对其进行 CRC 校验，方法是将整个数据串当作一个整体去除以生成多项式。若余数为零，则说明接收正确，否则接收错误，并不纠错。

5.4.2 循环冗余码的编译码和传输仿真

1. 循环冗余码的 MATLAB 仿真

循环冗余码不具有纠错能力，只能检错。下面的程序对 CRC-8 循环冗余的检错性能进行了仿真，生成多项式为 $g(x)=x^8+x^2+x+1$，为了看出检错性能，设置的发送帧数量比较大，程序运行较慢。

```
clear all;
N = 1000000;                              % 发送的帧数
L_info = 16;                              % 信息位长度
poly = [1 0 0 0 0 0 1 1 1];               % CRC 生成多项式系数
N1 = length(poly) - 1;                    % 校验位长度
EbN0 = 0:10;
ber = berawgn(EbN0,'qam',16);             % 16-OAM 理论误比特率
for i = 1:length(ber)
    Pe = ber(i);                          % BSC 信道错误概率
    for j = 1:N
        msg = randi([0 1],1,L_info);      % 生成一组信息序列
        msg1 = [msg,zeros(1,N1)];         % 信息位左移，准备拼接校验位
        [Q_x,R_x] = deconv(msg1,poly);    % 求 CRC 校验码，R_x 为余数系数
        R_x = mod(abs(R_x),2);            % 模 2 运算
        crc_bits = R_x(L_info + 1:end);   % 校验位
        frame = [msg,crc_bits];           % 组成一帧数据
        y = bsc(frame,Pe);                % 通过 BSC 信道
        [Q_y,R_y] = deconv(y,poly);       % 接收序列除以多项式
        R_y = mod(abs(R_y),2);            % 模 2 处理
        err(j) = biterr(frame,y);         % 统计本帧是否产生误码
        errl(j) = sum(R_y);               % 通过 CRC 统计本帧是否产生误码
    end
    ferl(i) = sum(err~= 0);               % 误帧率
    fer2(i) = sum(errl~= 0);              % 通过 CRC 计算误帧率
end
Pf = (ferl - fer2)/N;                     % CRC 漏检的概率
semilogy(EbN0,Pf);
title('CRC - 8 检错性能');
xlabel('Eb/N0');
ylabel('漏检概率');
grid on;
```

运行结果如图 5-9 所示，可看出循环冗余码的漏检率非常低。CRC-8 由于校验位少，计算开销小，常用于数据量小且对数据传输可靠性要求不是特别高的场合，如某些传感器数据传输、嵌入式系统内部通信等。

2. 循环冗余码的 Simulink 仿真

循环冗余码的 Simulink 仿真模型如图 5-10 所示。Bernoulli Binary Generator 模块产生的信源序列经过 General CRC Generator 编码后在 Binary Symmetric Channel 中传输。Error Rate Calculation 模块将输入信道前后信号进行比较，统计错误比特数量。统计结果经过关系运算模块与 CRC 的解码模块结果比较，若不相等，则视为漏检，由 Cumulative Sum 模块进行统计。模型中的循环冗余码的生成多项式是 $g(x)=x^{16}+x^{12}+x^5+1$。

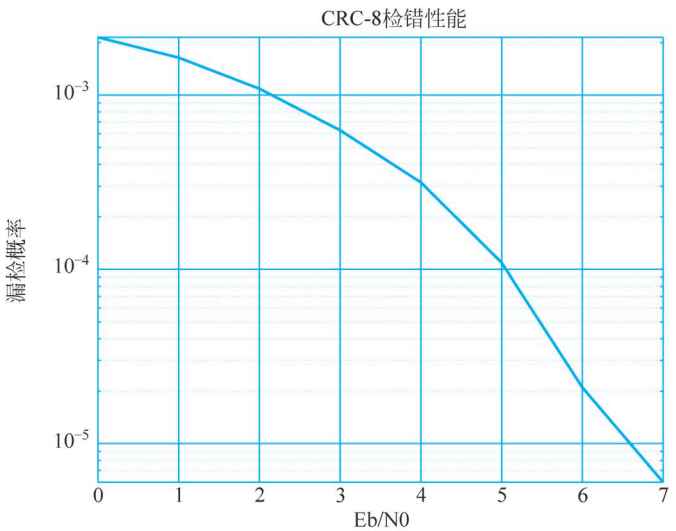


图 5-9　CRC-8 的检错性能

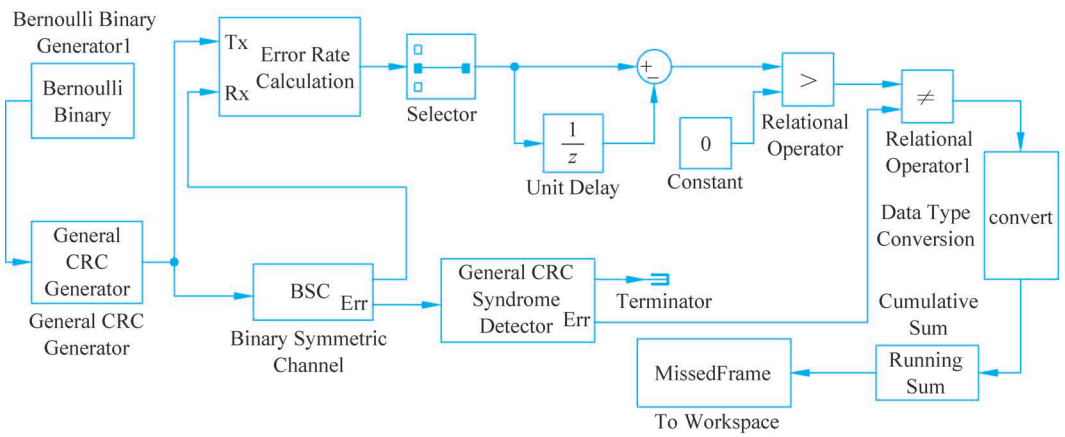


图 5-10　循环冗余码的 Simulink 仿真模型

表 5-4 是模型中各模块的主要参数设置。

表 5-4　循环冗余码的 Simulink 仿真参数

模 块 名 称	参 数 名 称	参　数　值
Bernoulli Binary Generator	Probability of zero	0.5
	Source of initial seed	Auto
	Sample time	1/64000000
	Samples per frame	64
	Output data type	double
	Simulate using	解释执行

续表

模块名称	参数名称	参数值
General CRC Generator	Codeword length N	'z^16+z^12+z^5+1'
	Final XOR	0
	Checksums per frame	1
	Initial states	0
Binary Symmetric Channel	Error probability	BER
	Initial seed	71
	Output data type	double
Selector	索引	[2]
	输入端口大小	3
	输入维数	1
Unit Delay(延时器)	采样时间	-1
	初始条件	0
	输入处理	元素作为通道
Error Rate Calculation	Receive delay	0
	Computation delay	0
	Computation mode	Entire frame
	Output data	port
To Workspace	Variable name	s
	Limit data point to last	inf
	Decimation	1
	Sample time	-1
	Save format	数组

本例仿真同样采用 MATLAB 和 Simulink 交互的方式，完成信噪比下的漏检该类统计，代码如下：

```
clear all;
x = 0:10; % x 表示 EbN0
ber = berawgn(x,'qam',16);
for j = 1:length(x)
    BER = ber(j);
    % 运行仿真程序，得到的误比特率保存在工作区变量 MissedFrame 中
    sim('book_sim_crc');
    % 计算 BitErrorRate 的均值，作为本次仿真的误比特率
    y(j) = ErrChecked(end)/length(ErrChecked);
end
% 绘制 x 和 y 的关系曲线图，纵坐标采用对数坐标
semilogy(x,y,'r-o')
% title('CRC-16 的检错能力');
xlabel('Eb/N0'),ylabel('漏检概率');
% axis([0 10 10.^(-6) 10.^(-3)])
grid on
```

运行结果如图 5-11 所示。从实验结果可以看出,CRC-16 的漏检率比 CRC-8 低很多。因为循环冗余码校验的准确率与码长有关,码长越长,漏检率越低,可靠性越高。

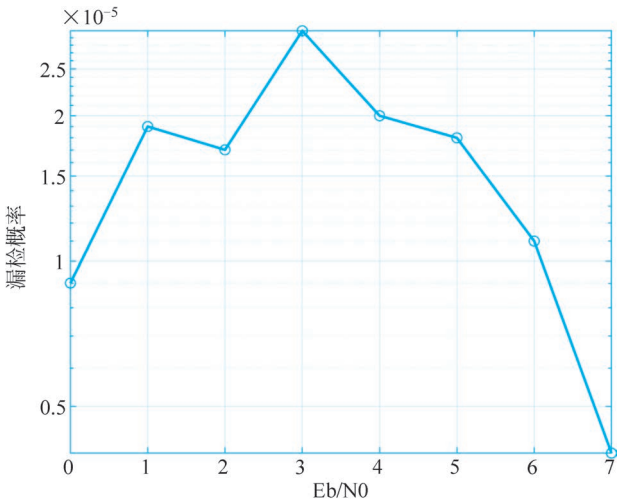


图 5-11　循环冗余码的传输仿真结果

5.5　基于 LDPC 码的差错控制系统仿真

5.5.1　LDPC 码简介

低密度奇偶检验(Low-Density Parity-Check,LDPC)码是一类具有稀疏校验矩阵(矩阵中"1"的数量很少)的线性分组码,由麻省理工学院的 Robert Gallager 在 1962 年提出,Mackay 和 Spielman 等在 1996 年证明其性能优越,有线性复杂度的译码算法。在长码长和高码率的情况下,LDPC 码性能非常接近香农极限,适用于卫星通信、深空通信等对数据通信质量要求较高的场景。在 5G 中,LDPC 码已被用于数据信道的编码。

LDPC 码具有以下特点。

(1) 译码复杂度很低,不会因为码长的增加而使运算量急剧增加,译码算法不仅是理论上存在,在物理上也是可以实现的,并且实际译码的仿真性接近于理论分析。

(2) 采用迭代译码算法,在物理上可以实现并行操作,并且译码速度高于其他编码。

(3) 拥有大的吞吐量,可提高传输效率,尤其在硬件模块更能体现这个优点。

(4) 奇偶校验矩阵具有稀疏性,这种稀疏性保证了译码复杂度和最小码距都只随码长呈现线性增加,所以不会出现码长过大而导致计算复杂度指数型增长的情况。

(5) 编码结构特性自带抗突发差错特性,应用于通信系统时不需要交织器,因此没有因交织器的存在而带来延时。

LDPC 校验矩阵 $\boldsymbol{H}$ 的每一行对应一个校验方程,每一列对应码字中的一比特。因此,对于一个二进制码,如果它有 m 个奇偶校验约束关系,码字的长度为 n,则校验矩阵是一个尺寸为 $m \times n$ 的二进制矩阵。对于 $m \times n$ 维校验矩阵 $\boldsymbol{H}$,当且仅当向量 $\boldsymbol{C}=$

$(c_{m-1},c_{m-2},\cdots,c_0)$满足

$$\boldsymbol{H}\boldsymbol{C}^{\mathrm{T}}=\boldsymbol{0}^{\mathrm{T}} \tag{5-4}$$

它才是该码的一个有效码字。

LDPC 码可通过伪随机的方法构造，需要在给定一些设计规范后得到所有 LDPC 码的集合，而不仅限于如何选择一些特殊的校验矩阵使之满足设计规范。LDPC 码常通过 Tanner 图表示，而 Tanner 图所表示的其实是 LDPC 码的校验矩阵。Tanner 图包含两类顶点：n 个码字比特顶点，也称为比特节点，分别与校验矩阵的各列相对应；m 个校验方程顶点，也称为校验节点，分别与校验矩阵的各行对应。校验矩阵的每一行代表一个校验方程，每一列代表一个码字比特。所以，如果一个码字比特包含在相应的校验方程中，那么就用一条连线将所涉及的比特节点和校验节点连起来，所以 Tanner 图中的连线数与校验矩阵中 1 的个数相同。图 5-12 是式(5-4)校验矩阵的 Tanner 图，其中上面的圆形节点表示校验节点，下面的圆形节点表示比特节点，黑线表示的是一个循环。

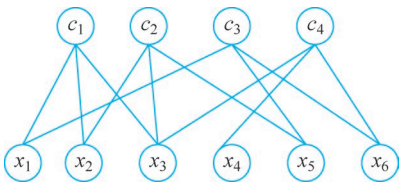


图 5-12　LDPC 码的 Tanner 图

$$\boldsymbol{H}=\begin{bmatrix}1&1&0&1&0&0\\0&1&1&0&1&0\\1&0&0&0&1&1\\0&0&1&1&0&1\end{bmatrix} \tag{5-5}$$

除了校验矩阵 $\boldsymbol{H}$ 是稀疏矩阵外，LDPC 码本身与任何其他的分组码并无二致。如果现有的分组码可以被稀疏矩阵所表达，那么用于 LDPC 码的迭代译码算法也可以成功地移植到它身上。不同的是，LDPC 码的设计是以构造一个校验矩阵开始的，然后才通过它确定一个生成矩阵进行后续编码。

除了采用为伪随机序列构造 LDPC 码之外，还采用高斯消元算法，算法流程图如图 5-13 所示。

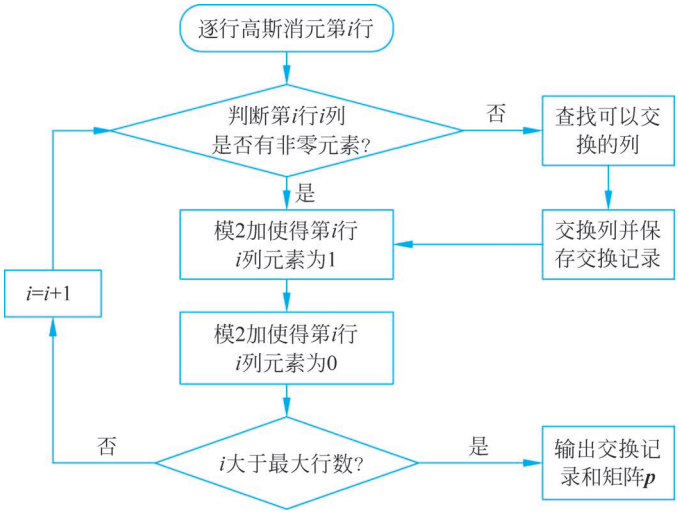


图 5-13　高斯消元算法流程图

LDPC 码的译码方法采用概率 BP 译码算法,流程图如图 5-14 所示。

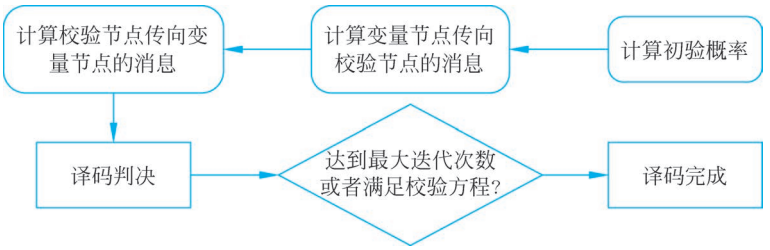


图 5-14　概率 BP 译码算法过程

5.5.2　LDPC 码的编译码和传输仿真

1. LDPC 码的 MATLAB 仿真

下面的程序是借助通信工具箱中的 LDPC 编译码函数实现,采用 BPSK 调制,AWGN 信道传输,计算误码率。

```
H = dvbs2ldpc(1/2);                                    % DVB-S2 标准的 LDPC 码
hEnc = comm.LDPCEncoder(H);                            % 声明 LDPC 码编码函数
hDec = comm.LDPCDecoder(H,'DecisionMethod','Soft decision');    % 声明 LDPC 码解码函数
hMod = comm.BPSKModulator;                             % 声明 BPSK 调制函数
hDemod = comm.BPSKDemodulator('DecisionMethod','Hard decision');    % 声明解调函数
% 设置信道参数
hAWGN = comm.AWGNChannel('NoiseMethod', 'Signal to noise ratio (Eb/No)','EbNo',10);
hError = comm.ErrorRate;                               % 声明误码率计算函数
for counter = 1:10
    data = randi([0 1], 32400, 1);                     % 产生随机序列
    encodedData = step(hEnc, data);                    % LDPC 编码
    modSignal = step(hMod, encodedData);               % BPSK 调制
    receivedSignal = step(hAWGN,modSignal);            % 通过加性高斯白噪声信道
    demodSignal = step(hDemod, receivedSignal);        % BPSK 解调
    receivedBits = step(hDec, demodSignal);            % LDPC 解码
    errorStats = step(hError, data, receivedBits);     % BPSK 调制解调后计算误码率
end
% 输出误码率和错误比特数
fprintf('Error rate = %1.2f\nNumber of errors = %d\n',  errorStats(1), errorStats(2));
```

运行结果如下:

```
Error rate = 0.15
Number of errors = 49897
```

2. LDPC 码的 Simulink 仿真

LDPC 码的 Simulink 仿真模型如图 5-15 所示。Bernoulli Binary Generator 模块,采样频率是 32400。信道模块和程序设计选用的信道模型一样,都是选用的加性高斯白噪声信道,调制解调模块选用的是二进制相移键控(BPSK),误码率的计算就用 Error Rate Calculation 模块,这个模块的作用的原理就是把信源的原始数据接入 Tx 端口,经过信道

传输后的输出信号接入 Rx 端口，让输出信号与原始信号进行对比来计算误码率。对比后的数据就用输出端口接入 Display 模块来显示误码率的大小。

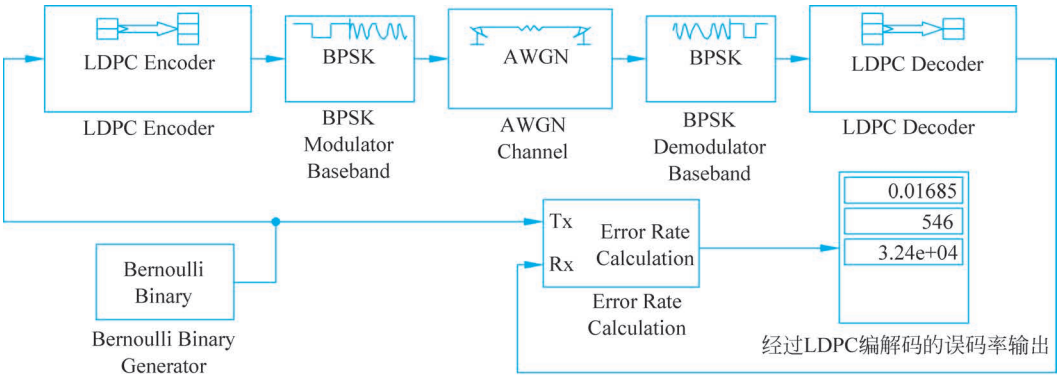


图 5-15　LDPC 码的 Simulink 仿真模型与仿真结果

表 5-5 是模型中各模块的主要参数设置。

表 5-5　LDPC 码的 Simulink 仿真参数

模块名称	参数名称	参数值
Bernoulli Binary Generator	Probability of zero	0.7
	Source of initial seed	61
	Sample time	1
	Samples per frame	32400
LDPC Encoder	Parity-check matrix	dvbs2ldpc(1/2)
BPSK Modulator	Phase offset(rad)	0
	Output data type	double
AWGN Channel	Initial seed	67
	Mode	Signal to noise ratio(Eb/No)
	Eb/No(dB)	10
	Number of bits per symbol	1
	Input signal power(watts)	1
	Symbol period(s)	1
LDPC Decoder	Output format	Information part
	Decision type	Soft decision
	Number of interation(迭代次数)	50
BPSK Demodulator	Decision type	Hard decision
	Phase offset(rad)	0
Error Rate Calculation	Receive delay	0
	Computation delay	0
	Computation mode	Entire frame
	Output data	port

图 5-16 是信噪比为 10dB 时 LDPC 码 Simulink 仿真系统的输出结果图，从图中可以非常清楚明了地看出 LDPC 码经过 BPSK 调制解调后输出消息的误码率较低。

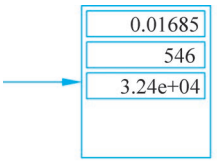


图 5-16　LDPC 码信号传输的仿真结果

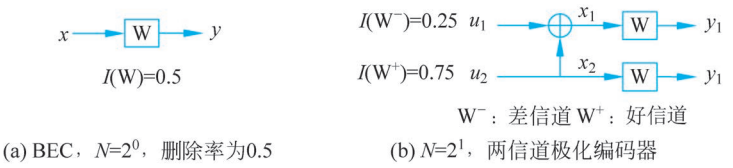
针对 5G 标准相关的建模仿真，MATLAB 的 5G 工具箱中提供了函数 nrLDPCEncode 和 nrLDPCDecode，这两个函数根据符合 5G NR 标准协议，可以直接调用。

5.6　基于极化码的差错控制系统仿真

5.6.1　极化码简介

极化码是 5G 的信道编码方案，是 Erdal Arikan 于 2009 年的论文“Channel Polarization: A Method for Constructing Capacity-Achieving Codes for Symmetric Binary-Input Memoryless Channels”中提出的信道编码方法。该方法基于阵列极化(Array Polarization，AP)的思想，对原始信道进行特定的线性变换，完成信道复合和分裂，使信道极化，达到一部分子信道的可靠性增强，而另一部分子信道的可靠性降低，最终使信道渐进性能达到香农限。

以码长 $N=2$ 为例，在离散无记忆信道的二进制删除信道(Binary Erasure Channel，BEC)中，要传输的数据为 u_1 和 u_2，则最简单的极化编码器结构如图 5-17(b)所示。



(a) BEC，$N=2^0$，删除率为0.5　　(b) $N=2^1$，两信道极化编码器

图 5-17　BEC 中 N＝2 时的信道极化编码器

图中的 W 表示信道，原数据组 $\boldsymbol{U}=[u_1, u_2]$ 经信道极化编码后获得新数据组 $\boldsymbol{X}=[x_1, x_2]$，其中 $x_1=u_1 \oplus u_2$，$x_2=u_2$。“⊕”表示模 2 加，也就是异或操作。$\boldsymbol{X}$ 和 $\boldsymbol{U}$ 之间的关系采用生成矩阵的方式可描述为

$$\boldsymbol{X}=\boldsymbol{U}\boldsymbol{G}_2 \tag{5-6}$$

式中，$\boldsymbol{G}_2=\begin{bmatrix}1 & 0\\ 1 & 1\end{bmatrix}$ 是生成矩阵。经过矩阵 $\boldsymbol{G}_2$ 的极化运算，原先的独立信道对(W，W)变换为相关子信道对(W^-，W^+)，对应的互信息分别为 $I(W^-)=0.25$，$I(W^+)=0.75$。对比图 5-17(a)中未极化的信道 W，图 5-17(b)中的两个子信道产生了分化，W^- 信道较差，W^+ 较好，矩阵 $\boldsymbol{G}_2$ 也称为极化核。

图 5-17(b)描述的是一个基本的极化编码单元。多个基本极化编码器单元可以扩展构成更加复杂的编码器。图 5-18 是码长 $N=4$ 时的信道极化编码器。

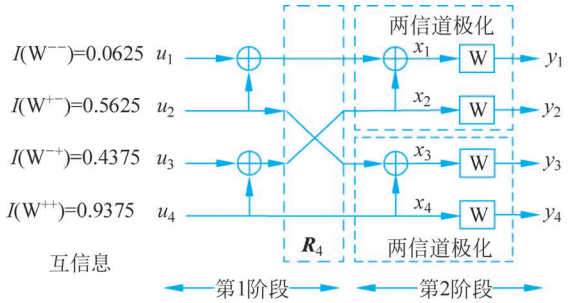


图 5-18 BEC 中 $N=4$ 时的信道极化编码器

对于 $N=4$，信道的极化为两个阶段，第 2 阶段分化出两个 W^- 信道和两个 W^+ 信道，第 1 阶段是这 4 个信道的复合，两个 W^- 信道复合出 W^{--} 和 W^{-+} 信道，两个 W^+ 信道复合出 W^{+-} 和 W^{++} 信道，各信道的互信息不同。为了提高可靠性，选择好信道传送信息比特，差信道传输固定比特（即冻结比特）。

分析图 5-18 中极化前后数据组的关系，可以得到原数据组 $\boldsymbol{U}=[u_1, u_2, u_3, u_4]$ 经极化编码后获得新数据组 $\boldsymbol{X}=[x_1, x_2, x_3, x_4]$，其中 $x_1=u_1\oplus u_2\oplus u_3\oplus u_4$，$x_2=u_3\oplus u_4$，$x_3=u_2\oplus u_4$，$x_4=u_4$。因此，$N=4$ 时的极化核为

$$\boldsymbol{G}_4=\begin{bmatrix}1&0&0&0\\1&0&1&0\\1&1&0&0\\1&1&1&1\end{bmatrix} \tag{5-7}$$

按照同样的信道极化方法，扩展图 5-18 中的信道数量，将两个码长为 $N/2$ 的相互独立的信道递归迭代，合并成长度为 N 的组合信道。以此得到不同信息组长度的信道极化编码器和相应的极化核。但 N 信道极化更严谨和通用的数学描述方法为

$$u_1^N\boldsymbol{B}_N\boldsymbol{G}_2^{\otimes n}=x_1^N \tag{5-8}$$

式中，输入长度 $N=2^n$；$\boldsymbol{B}_N$ 是比特逆序重排矩阵（图 5-18 中 $\boldsymbol{R}_4$ 部分的运算），作用是将输入序列比特序号重新排列，便于后续运算，如将(u_1, u_2, u_3, u_4)重排为(u_1, u_3, u_2, u_4)，具体计算方法是 $\boldsymbol{B}_N=\boldsymbol{R}_N(\boldsymbol{I}_2\otimes\boldsymbol{B}_{N/2})$。$\boldsymbol{R}_N$ 排列运算，作用是将输入的奇数序号和偶数序号分开排列，这里是奇数序号在前，偶数序号在后，$\boldsymbol{I}_2$ 是 2 阶单位矩阵，“$\otimes$”表示 Kronecker 积，对于 $m\geqslant 1$ 有 $\boldsymbol{A}^{\otimes n}=\boldsymbol{A}\otimes\boldsymbol{A}^{\otimes n-1}$。$\boldsymbol{G}_2=\begin{bmatrix}1&0\\1&1\end{bmatrix}$。

图 5-18 所示的四信道极化变换过程可以表示为

$$u_1^4\boldsymbol{B}_4\boldsymbol{G}_2^{\otimes 2}=(u_1,u_2,u_3,u_4)(\boldsymbol{R}_4(\boldsymbol{I}_2\otimes\boldsymbol{B}_{4/2}))(\boldsymbol{G}_2\otimes\boldsymbol{G}_2)$$

$$=(u_1,u_2,u_3,u_4)\left(\begin{bmatrix}1&0&0&0\\0&0&1&0\\0&1&0&0\\0&0&0&1\end{bmatrix}\left(\begin{bmatrix}1&0\\0&1\end{bmatrix}\otimes\begin{bmatrix}1&0\\0&1\end{bmatrix}\right)\right)\left(\begin{bmatrix}1&0\\1&1\end{bmatrix}\otimes\begin{bmatrix}1&0\\1&1\end{bmatrix}\right)$$

$$
\begin{aligned}
&=(u_1,u_2,u_3,u_4)\left(\begin{bmatrix}1&0&0&0\\0&0&1&0\\0&1&0&0\\0&0&0&1\end{bmatrix}\begin{bmatrix}1&0&0&0\\0&1&0&0\\0&0&1&0\\0&0&0&1\end{bmatrix}\right)\begin{bmatrix}1&0&0&0\\1&1&0&0\\1&0&1&0\\1&1&1&1\end{bmatrix}\\
&=(u_1,u_2,u_3,u_4)\begin{bmatrix}1&0&0&0\\0&0&1&0\\0&1&0&0\\0&0&0&1\end{bmatrix}\begin{bmatrix}1&0&0&0\\1&1&0&0\\1&0&1&0\\1&1&1&1\end{bmatrix}\\
&=(u_1,u_2,u_3,u_4)\begin{bmatrix}1&0&0&0\\1&0&1&0\\1&1&0&0\\1&1&1&1\end{bmatrix}\\
&=u_1^4\boldsymbol{G}_4\\
&=x_1^4
\end{aligned}
\tag{5-9}
$$

这里的 $\boldsymbol{G}_4$ 结果与式(5-7)一样。

极化码得到众多学者的研究,提出了多种信道的可靠性评估方法。目前比较主流的方法有计算巴氏参数、高斯近似(Gaussian Approximation,GA)和极化重量(Polarization Weight,PW)这 3 种。巴氏参数常出现在早期极化码文献,代表 BEC 信道使用最大后验概率判定时错误概率的上限,巴氏参数数值越小,相对应的子信道对称容量越大,说明信道越可靠。针对 AWGN 信道,可利用 GA 算法计算,该把多维信道参数简化成一维,计算结果越小的子信道具有越高的可靠性,缺点是计算较为复杂。2016 年,华为公司提出 PW 算法,该方法已被用于 5G 标准协议中的,适用性和构造性更好。

串行抵消译码(Successive Cancellation,SC)算法是极化码最经典的译码算法之一。SC 算法根据信道输出和之前已译码的比特信息,计算当前比特的对数似然比(Logarithmic Likelihood Ratio,LLR),然后根据 LLR 的正负来判决当前比特的值。SC 译码算法简单,易于硬件实现,但前面的译码错误会对后序比特的译码产生影响,尤其是在码长较短时,误码率较高。串行抵消列表译码(Successive Cancellation List,SCL)法建立所有可能的译码结果列表,然后基于 LLR 的路径度量(Path Metric,PM)迭代挑选几个值高的进行扩展,继续利用信道估计和编码规则进行似然值的更新,最后选取 PM 值最小的路径译码输出,在一定程度上能减少译码结果错误的情况。循环冗余校验辅助的串行抵消列表译码(CRC－Adied SCL,CRC－SCL)首先要求在信源 CRC 编码后再进行极化编码发送,信宿进行 SCL 译码,保留 L 条候选路径,并对每条候选路径进行 CRC 校验,最终选定最优路径作为译码结果。该方法通过合理选择列表大小 L,可以在译码性能和复杂度之间取得较好的平衡,使其在实际硬件实现中具有可行性,是 5G 标准中的译码方法。

5.6.2 极化码的编译码和传输仿真

1. 信道极化后的对称容量

经过信道极化操作，一组原本具有相同信道特性的二进制输入离散无记忆信道变成具有不同可靠性的等效子信道，这些子信道的对称信道容量近乎两极分化，一部分子信道的容量趋于香农限，而一部分子信道的对称容量趋于零。下面程序计算了一组数量为1024、删除率为0.5的BEC信道进行10级信道极化之后各子信道的对称容量。

```
clear;
index = 10;
n = 2.^(1:index);
W = zeros(n(10));
W(1,1) = 0.5;
for i = n
    for j = 1:i/2
        W(i,2*j-1) = W(i/2,j)^2;
        W(i,2*j) = 2*W(i/2,j)-W(i/2,j)^2;
    end
end
scatter(1:1024,W(1024,1:1024),'b.');
axis([0 1024 0 1]);
xlabel('子信道序号');ylabel('对称容量');
title('信道极化现象');
```

图5-19是极化后各子信道对称容量的图形化显示。1024个子信道的对称容量在[0,1]区间出现两极化分布。

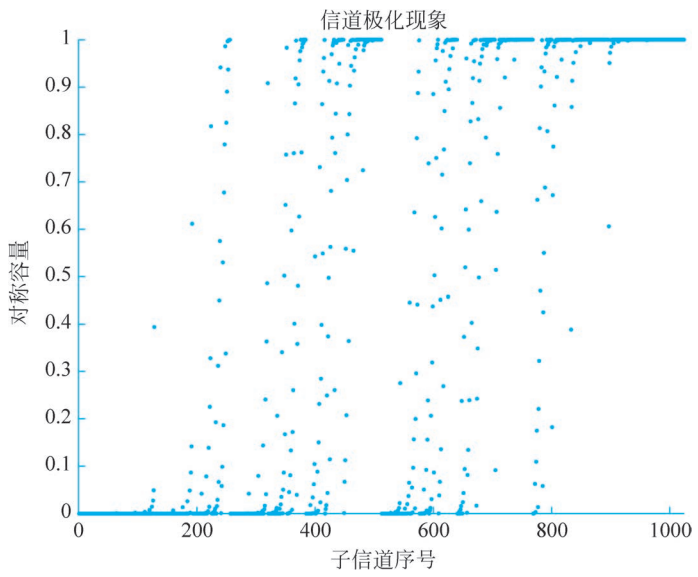


图5-19 BEC极化后各等效信道的对称容量，删除率为0.5

2. 极化码的编译码和传输仿真

下面的程序实现(64,32)极化码的编译码、信道传输及误码率计算。码长为64,信息位长为32,码率为1/2。利用PW公式计算信道重量,在63个子信道中选出32个好信道用于传输信息位。子信道的索引号在getInfoIndex函数中得到。位长为32的信息序列在getPerfDot函数中由randi随机产生。polarEncode函数负责产生码长为64的极化码码字。编码完成后,在getPerfDot函数中进行BPSK调制,并模拟在AWGN信道中传输。polarDecode函数负责对传输后信号采用SC方法译码。

```
clear; clc; close all;
L_word = 64; % 码长
L_info = 32; % 信息位长度
info_index = getInfoIndex(L_word, L_info);      % 获得信息位索引
EbN0_dBs = linspace(0, 5, 11);                  % 生成0～5的11个'Eb/N0'值
sigma2s = 1 ./ (10 .^(EbN0_dBs / 10) * L_info / L_word * 2);
                                                % 信道噪声方差的理论值
blers = zeros(size(EbN0_dBs));
Nloop = 1000;                                   % 设置循环仿真次数
for iter = 1:Nloop
    blers_temp = zeros(size(EbN0_dBs));
    for iter_i = 1:numel(EbN0_dBs)
        blers_temp(iter_i) = getPerfDot(info_index, sigma2s(iter_i), "soft");
    end

% ******** 函数,计算信息位索引 ********
function bler = getPerfDot(info_index, sigma2, judge_type)
info_bits = randi([0, 1], 1, numel(find(info_index)));      % 生成信息位
code_bits = polarEncode(info_index, info_bits);             % 获得码字
tx = 1 - 2 * code_bits; % BPSK调制{0 -> +1, 1 -> -1}
rx_llrs = (tx + sqrt(sigma2) * randn(size(tx)) ) * 2 / sigma2;
% 获得块误码率
dcd_bits = polarDecode(info_index, rx_llrs, judge_type);    % 解码
bler = double(mean(dcd_bits(info_index) ~= info_bits));     % 误码率
end

% ******** 函数,计算指定噪声方差下的BLER ********
function bler = getPerfDot(info_index, sigma2, judge_type)
info_bits = randi([0, 1], 1, numel(find(info_index)));      % 生成信息位
code_bits = polarEncode(info_index, info_bits);             % 获得码字
tx = 1 - 2 * code_bits; % BPSK调制{0 -> +1, 1 -> -1}
rx_llrs = (tx + sqrt(sigma2) * randn(size(tx)) ) * 2 / sigma2;
% 获得块误码率
dcd_bits = polarDecode(info_index, rx_llrs, judge_type);    % 解码
bler = double(mean(dcd_bits(info_index) ~= info_bits));     % 误码率
end

% ******** 函数,Polar编码 ********
function code_word = polarEncode(info_index, info_bits)
L = numel(info_index);                                      % 码长
N = log2(L);                                                % 层级
```

```
code = zeros(1, L);
code(info_index) = info_bits; % 在信息索引处放入信息位
% 编码
for i = N:-1:1
    polar_groups = reshape(1:L, 2^(N-i), 2, 2^(i-1));
    code = reshape(pagemtimes(code(polar_groups), [1, 0; 1, 1]), 1, L);
end
code_word = mod(code, 2);                                    % 码字
end

% ******** 函数,Polar 解码 ********
function dcd_word = polarDecode(info_index, llrs, judge_type)
L = numel(info_index);                                       % 码长
N = log2(L);                                                 % 码树层数
pin = [llrs.', nan(L, N-1), inf(L, 1)];                      % 引脚数据
pin(info_index, end) = nan;
arikan_F = @(X1, X2) 2 * atanh(tanh(X1 / 2) * tanh(X2 / 2));
minsum_func = @(X1, X2) sign(X1 * X2) * min(abs(X1), abs(X2));
if judge_type == "soft"
    judge_func = arikan_F;
else
    judge_func = minsum_func;
end
% SC 译码
i_level = 1;
i_node = 1;
while true
    % 若当前节点有子节点
    if i_level <= N
        polar_group = reshape(1:L, 2^(N-i_level), 2, 2^(i_level-1));
        node_group = polar_group(:, :, i_node);
        % 若当前节点左孩子节点未计算
        if isnan(pin(node_group(1), i_level + 1))
            for i_group = 1:size(node_group, 1)              % 计算当前节点
                Y1 = pin(node_group(i_group, 1), i_level);
                Y2 = pin(node_group(i_group, 2), i_level);
                U1 = judge_func(Y1, Y2);
                pin(node_group(i_group, 1), i_level + 1) = U1;
            end
            i_node = i_node * 2 - 1;                         % 设置跳到左孩子
            i_level = i_level + 1;                           % 设置到跳左孩子层
        % 若当前右孩子未计算
        elseif isnan(pin(node_group(end), i_level + 1))
            for i_group = 1:size(node_group, 1)              % 计算当前节点
                Y1 = pin(node_group(i_group, 1), i_level);
                Y2 = pin(node_group(i_group, 2), i_level);
                U1 = pin(node_group(i_group, 1), i_level + 1);
                U2 = judge_func(U1, Y1) + Y2;
                pin(node_group(i_group, 2), i_level + 1) = U2;
            end
```

```
            i_node = i_node * 2;                                    % 设置跳到右孩子
            i_level = i_level + 1;                                  % 设置跳到右孩子层
        % 计算本节点并跳转到父节点
        else
            for i_group = 1:size(node_group, 1)
                U1 = pin(node_group(i_group, 1), i_level + 1);
                U2 = pin(node_group(i_group, 2), i_level + 1);
                Y1 = judge_func(U1, U2);
                Y2 = U2;
                pin(node_group(i_group, 1), i_level) = Y1;
                pin(node_group(i_group, 2), i_level) = Y2;
            end
            i_node = ceil(i_node / 2);
            i_level = i_level - 1;
        end
    % 没有未计算子节点
    elseif i_node == L
        break;
    else
        if judge_type == "hard"
            pin(i_node, i_level) = inf * pin(i_node, i_level);
        end
        i_node = ceil(i_node / 2);
        i_level = i_level - 1;
    end
end
dcd_word = (1 - sign(pin(:, end).')) / 2;
end
```

运行结果如图 5-20 所示。

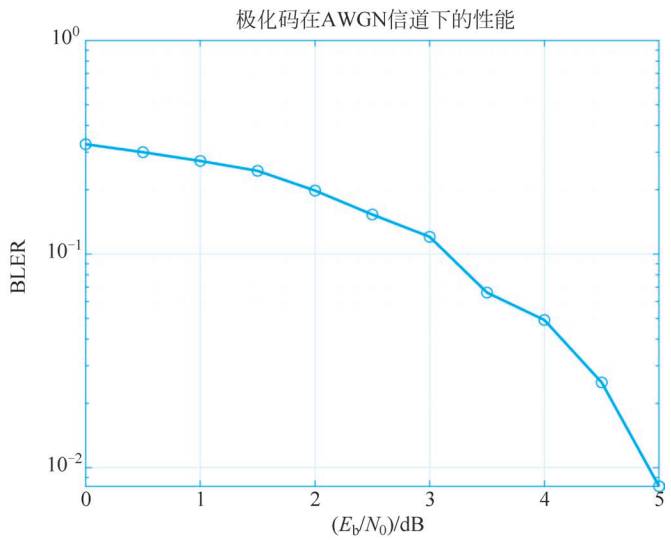


图 5-20　AWGN 信道下极化码的性能

针对极化码在5G新空口(NR)的系统级仿真,可以利用MATLAB提供的5G Toolbox™,其中组件函数多以“nr”开头,如nrPolarEncode、nrPolarDecode、nrCRCEncode、nrRateMatchPolar、nrSymbolModulate等,实现5G中的极化码编解码、速率匹配、附加CRC等,这些函数符合5G NR标准协议,可以直接调用,极化码主要用于DCI、UCI上下行控制信息与BCH承载的广播信息的信道编码。