第3章 通信系统接收机设计

3.1 利用直接序列扩频技术设计发射机

直接序列扩频通信系统的接收机如图 3-1 所示。

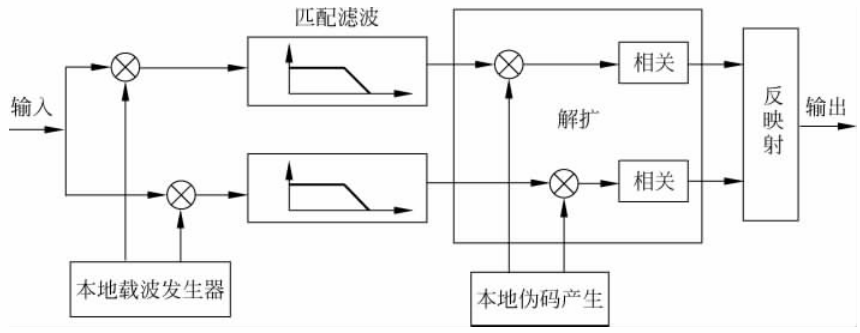

图 3-1 扩频通信系统接收机框图

(1) 相干解调

数字信号经过两路正交的载波进行下变频之后重新得到基带信号。

(2) Nyquist 滤波

该滤波器的作用有两点：①经过 A/D 变换和下变频的信号含有许多寄生频谱，因而必须用一个低通滤波器予以消除；②对接收到的信号进行匹配滤波。

(3) 解扩

将接收机的信号与本地伪码进行相关运算，以恢复出原始传输数据。

(4) 反映射

将解扩后的数据通过符号判决重新对应为星座图上的点，再对应为 0 和 1 表示的二进制数据。

3.2 利用 IS-95 前向链路技术设计接收机

接收部分从信道接收信号。经基带滤波、短码解扩、沃尔什解调、解扰、去交织和维特比译码，输出解调后的信号。各通信模块和发射机的相关模块设计类似。

3.3 利用 OFDM 技术设计接收机

OFDM 系统设计的接收机的框图如图 3-2 所示。

接收机很多通信处理的模块都与发射机的相关模块功能相似，这里不再一一介绍，接收机主要增加了同频模块。

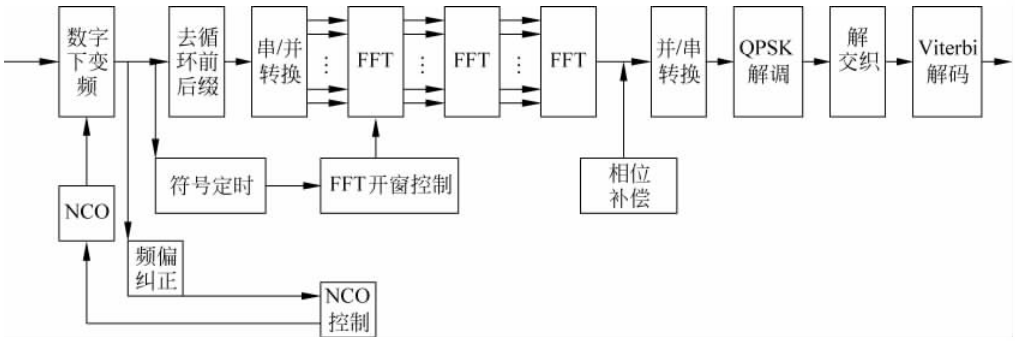

图 3-2 OFDM 系统接收机框图

3.4 通信系统的 MATLAB 实现

下面列出了 IS-95 前向链路系统的 MATLAB 仿真程序。

```
>> %数据速率 = 9600kbps
clear all
global Zi Zq Zs show R Gi Gq
show = 0; SD = 0;                                    %选择软/硬判决接收
%主要的仿真参数设置
BitRate = 9600; ChipRate = 1228800;
N = 184; MFType = 1;                                 %匹配滤波器类型升余弦
R = 5;
%Viterbi 生成多项式
G_Vit = [1 1 1 1 0 1 0 1 1; 1 0 1 1 1 1 0 0 0 1];
K = size(G_Vit, 2);L = size(G_Vit, 1);
%Walsh 矩阵代码
WLen = 64;
Walsh = reshape([1;0] * ones(1, WLen/2), WLen , 1);
%Walsh = zeros(WLen ,1);
%扩频调制 PN 码的生成多项式
Gi_ind = [15, 13, 9, 8, 7, 5, 0]';
Gq_ind = [15, 12, 11, 10, 6, 5, 4, 3, 0]';
Gi = zeros(16, 1);
Gi(16 - Gi_ind) = ones(size(Gi_ind));
Zi = [zeros(length(Gi) - 1, 1); 1];
%I 路信道 PN 码生成器的初始状态
Gq = zeros(16, 1);
Gq(16 - Gq_ind) = ones(size(Gq_ind));
```

```
Zq = [zeros(length(Gq) - 1, 1); 1];
%Q路信道PN码生成器的初始状态
%扰码生成多项式
Gs_ind = [42, 35, 33, 31, 27, 26, 25, 22, 21, 19, 18, 17, 16, 10, 7, 6, 5, 3, 2, 1, 0]';
Gs = zeros(43, 1);
Gs(43 - Gs_ind) = ones(size(Gs_ind));
Zs = [zeros(length(Gs) - 1, 1); 1];
%长序列生成器的初始状态
%AWGN信道
EbEc = 10 * log10(ChipRate/BitRate);
EbEcVit = 10 * log10(L);
EbNo = [ - 2 : 0.5 : 6.5];                          %仿真信噪比范围(dB)
%实现主程序
ErrorsB = []; ErrorsC = []; NN = [];
if (SD == 1)
    fprintf('\n SOFT Decision Viterbi Decoder\n\n');
else
    fprintf('\n HARD Decision Viterbi Decoder\n\n');
end
for i = 1:length(EbNo)
    fprintf('\nProcessing %1.1f (dB)', EbNo(i));
    iter = 0;ErrB = 0; ErrC = 0;
    while (ErrB < 300) & (iter < 150)
        drawnow;
        %发射机实现
        TxData = (randn(N, 1)> 0);
        %速率为19.2kcps
        [TxChips, Scrambler] = PacketBuilder(TxData, G_Vit, Gs);
        %速率为1.2288Mcps
        [x PN MF] = Modulator(TxChips, MFType, Walsh);
        %实现信道代码
        noise = 1/sqrt(2) * sqrt(R/2) * ( randn(size(x)) + j * randn(size(x))) * ...
10 ^ ( - (EbNo(i) - EbEc)/20);
        r = x + noise;
        %实现接收机代码
        RxSD = Demodulator(r, PN, MF, Walsh);  %软判决,速率为19.2kcps
        RxHD = (RxSD > 0);                     %定义接收码片的硬判决
        if (SD)
            [RxData Metric] = ReceiverSD(RxSD, G_Vit, Scrambler);     %软判决
        else
            [RxData Metric] = ReceiverHD(RxHD, G_Vit, Scrambler);     %硬判决
        end
        if(show)
            subplot(311); plot(RxSD, '-o'); title('Soft Decisions');
            subplot(312); plot(xor(TxChips, RxHD), '-o'); title('Chip Errors');
            subplot(313); plot(xor(TxData, RxData), '-o');
            title(['Data Bit Errors. Metric = ', num2str(Metric)]);
        end
        if(mod(iter, 50) == 0)
            fprintf('.');
            save TempResults ErrB ErrC N iter
```

```
        end
        ErrB = ErrB + sum(xor(RxData, TxData));
        ErrC = ErrC + sum(xor(RxHD, TxChips));
        iter = iter + 1;
    end
    ErrorsB = [ErrorsB; ErrB];
    ErrorsC = [ErrorsC; ErrC];
    NN = [NN; N * iter];
    save SimData *
end
% 实现误码率计算
PerrB = ErrorsB./NN; PerrC = ErrorsC./NN;
Pbpsk = 1/2 * erfc(sqrt(10.^(EbNo/10)));
PcVit = 1/2 * erfc(sqrt(10.^((EbNo - EbEcVit)/10)));
Pc = 1/2 * erfc(sqrt(10.^((EbNo - EbEc)/10)));
% 实现性能仿真显示代码
figure;
semilogy(EbNo(1:length(PerrB)), PerrB, 'b - * '); hold on;
xlabel('信噪比/dB');
ylabel('误码率');
grid on;
```

运行程序，得到前向链路系统仿真效果图，如图 3-3 所示。

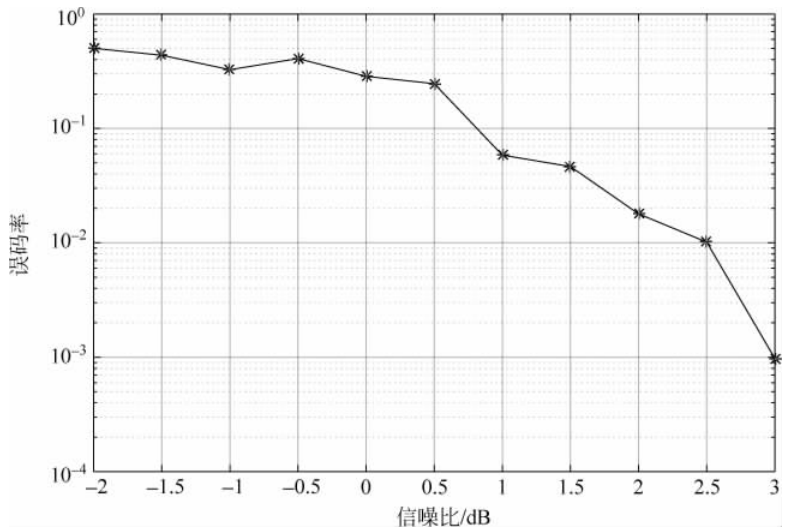

图 3-3　前向链路系统仿真效果图

在运行程序过程中，调用到以下用户自定义编写的函数，它们的源代码分别如下。

```
function [ChipsOut, Scrambler] = PacketBuilder(DataBits, G, Gs);
% 此函数用于产生 IS - 95 前向链路系统的发送数据包
% DataBits 为发送数据(二进制形式)
% G 为 Viterbi 编码生成多项式
% Gs 为长序列生成多项式(扰码生成多项式)
% ChipsOut 为输入到调制器的码序列(二进制形式)
% Scrambler 为扰码
```

```
global Zs
K = size(G, 2); L = size(G, 1);
N = 64 * L * (length(DataBits) + K - 1);          % 码片数 (9.6kbps -> 1.288 Mbps)
chips = VitEnc(G, [DataBits; zeros(K - 1,1)]);    % Viterbi 编码
% 实现交织编码
INTERL = reshape(chips, 24, 16);                  % IN: 列, OUT: 行
chips = reshape(INTERL', length(chips), 1);       % 速率 = 19.2kbps
% 产生扰码
[LongSeq Zs] = PNGen(Gs, Zs, N);
Scrambler = LongSeq(1:64:end);
ChipsOut = xor(chips, Scrambler);
function y = VitEnc(G, x);
% 此函数根据生成多项式进行 Viterbi 编码
% G 为生成多项式的矩阵
% x 为输入数据(二进制形式)
% y 为 Viterbi 编码输出序列
K = size(G, 1); L = length(x);
yy = conv2(G, x'); yy = yy(:, 1:L);
y = reshape(yy,K * L, 1); y = mod(y, 2);
function [y, Z] = PNGen(G, Zin, N);
% 此函数是根据生成多项式和输入状态产生长度为 N 的伪随机序列
% G 为生成多项式
% Zin 为移位寄存器初始化
% N 为 PN 序列长度
% y 为生成的 PN 码序列
% Z 为移位寄存器的输出状态
L = length(G); Z = Zin;                           % 移位寄存器的初始化
y = zeros(N, 1);
for i = 1:N
    y(i) = Z(L);
    Z = xor(G * Z(L), Z);
    Z = [Z(L); Z(1:L - 1)];
end
function [TxOut, PN, MF] = Modulator(chips, MFType, Walsh);
% 此函数用于实现 IS - 95 前向链路系统的数据调制
% chips 为发送的初始数据
% MFType 为成型滤波器的类型选择
% Walsh 为 walsh 码
% TxOut 为调制输出信号序列
% PN 为用于扩频调制的 PN 码序列
% MF 为匹配滤波器参数
global Zi Zq show R Gi Gq
N = length(chips) * length(Walsh);
% 输入速率 = 19.2kbps, 输出速率 = 1.2288mcps
tmp = sign(Walsh - 1/2) * sign(chips' - 1/2);
chips = reshape(tmp, prod(size(tmp)), 1);
[PNi Zi] = PNGen(Gi, Zi, N);
[PNq Zq] = PNGen(Gq, Zq, N);
PN = sign(PNi - 1/2) + j * sign(PNq - 1/2);
chips_out = chips. * PN;
chips = [chips_out, zeros(N, R - 1)];
```

```
chips = reshape(chips.', N*R, 1);
%成型滤波器
switch (MFType)
case 1
    %升余弦滤波器
    L = 25; L_2 = floor(L/2);
    n = [-L_2:L_2]; B = 0.7;
    MF = sinc(n/R).*(cos(pi*B*n/R)./(1-(2*B*n/R).^2));
    MF = MF/sqrt(sum(MF.^2));
case 2
    %矩形滤波器
    L = R; L_2 = floor(L/2);
    MF = ones(L, 1);
    MF = MF/sqrt(sum(MF.^2));
case 3
    %汉明滤波器
    L = R; L_2 = floor(L/2);
    MF = hamming(L);
    MF = MF/sqrt(sum(MF.^2));
end
MF = MF(:);
TxOut = sqrt(R)*conv(MF, chips)/sqrt(2);
TxOut = TxOut(L_2+1: end - L_2);
if (show)
    figure;
    subplot(211); plot(MF, '-o'); title('Matched Filter'); grid on;
    subplot(212); psd(TxOut, 1024, 1e3, 113); title('Spectrum');
end
function [SD] = Demodulator(RxIn, PN, MF, Walsh);
%此函数是实现基于RAKE接收机的IS-95前向信链路系统的数据包的解调
%RxIn为输入信号
%PN为PN码序列(用于解扩)
%MF为匹配滤波器参数
%Walsh为用于解调的walsh码
%SD为RAKE接收机的软判决输出
global R
N = length(RxIn)/R; L = length(MF);
L_2 = floor(L/2); rr = conv(flipud(conj(MF)), RxIn);
rr = rr(L_2+1: end - L_2);
Rx = sign(real(rr(1:R:end))) + j*sign(imag(rr(1:R:end)));
Rx = reshape(Rx, 64, N/64);
Walsh = ones(N/64, 1)*sign(Walsh'-1/2);
PN = reshape(PN, 64, N/64)'; PN = PN.*Walsh;
%输入速率 = 1.2288 Mpbs, 输出速率=19.2kbps
SD= PN*Rx; SD= real(diag(SD));
function [DataOut, Metric] = ReceiverSD(SDchips, G, Scrambler);
%此函数用于实现基于Viterbi译码的发送数据的恢复
%SDchips为软判决RAKE接收机输入符号
%G为Viterbi编码生成多项式矩阵
%Scrambler为扰码序列
%DataOut为接收数据(二进制形式)
```

```
%Metric 为 Viterbi 译码最佳度量
if (nargin == 1)
    G = [1 1 1 1 0 1 0 1 1; 1 0 1 1 1 0 0 0 1];
end
%速率 = 19.2kbps
SDchips = SDchips. * sign(1/2 - Scrambler);
INTERL = reshape(SDchips, 16, 24);
SDchips = reshape(INTERL', length(SDchips), 1);%速率 = 19.2kbps
[DataOut Metric] = SoftVitDec(G, SDchips, 1);
function [xx, BestMetric] = SoftVitDec(G, y, ZeroTail);
%此函数是实现软判决输入的 Viterbi 译码
%G 为生成多项式的矩阵
%y 为输入的待译码序列
%ZeroT 为判断是否包含 0 尾
%xx 为 Viterbi 译码输出序列
%BestMetric 为最后的最佳度量
L = size(G, 1);                                    %输出码片数
K= size(G, 2);                                     %生成多项式的长度
N = 2^(K-1);                                       %状态数
T = length(y)/L;                                   %最大栅格深度
OutMtrx = zeros(N, 2 * L);
for s = 1:N
    in0 = ones(L, 1) * [0, (dec2bin((s-1), (K-1))-'0')];
    in1 = ones(L, 1) * [1, (dec2bin((s-1), (K-1))-'0')];
    out0 = mod(sum((G. * in0)'), 2);
    out1 = mod(sum((G. * in1)'), 2);
    OutMtrx(s, :) = [out0, out1];
end
OutMtrx = sign(OutMtrx-1/2);
PathMet = [100; zeros((N-1), 1)];                  %初始状态 = 100
PathMetTemp = PathMet(:,1);
Trellis = zeros(N, T); Trellis(:,1) = [0 : (N-1)]';
y = reshape(y, L, length(y)/L);
for t = 1:T
    yy = y(:, t);
    for s = 0:N/2-1
        [B0 ind0] = max( PathMet(1+[2 * s, 2 * s+1]) + [OutMtrx(1+2 * s, 0+[1:L])...
 * yy; OutMtrx(1+(2 * s+1), 0+[1:L]) * yy] );
        [B1 ind1] = max( PathMet(1+[2 * s, 2 * s+1]) + [OutMtrx(1+2 * s, L+[1:L])...
 * yy; OutMtrx(1+(2 * s+1), L+[1:L]) * yy] );
        PathMetTemp(1+[s, s+N/2]) = [B0; B1];
        Trellis(1+[s, s+N/2], t+1) = [2 * s+(ind0-1); 2 * s + (ind1-1)];
    end
    PathMet = PathMetTemp;
end
xx = zeros(T, 1);
if (ZeroTail)
    BestInd = 1;
else
    [Mycop, BestInd] = max(PathMet);
end
```

```
BestMetric = PathMet(BestInd);
xx(T) = floor((BestInd-1)/(N/2));
NextState = Trellis(BestInd, (T+1));
for t=T:-1:2
    xx(t-1) = floor(NextState/(N/2));
    NextState = Trellis( (NextState+1), t);
end
if (ZeroTail)
    xx = xx(1:end-K+1);
end
function [DataOut, Metric] = ReceiverHD(HDchips, G, Scrambler);
%此函数用于实现基于Viterbi译码的硬判决接收机
%SDchips为硬判决RAKE接收机输入符号
%G为Viterbi编码生成多项式矩阵
%Scrambler为扰码序列
%DataOut为接收数据(二进制形式)
%Metric为Viterbi译码最佳度量
if (nargin == 1)
    G = [1 1 1 1 0 1 0 1 1; 1 0 1 1 1 0 0 0 1];
end
%速率=19.2kbps
HDchips = xor(HDchips, Scrambler);
INTERL = reshape(HDchips, 16, 24);
HDchips = reshape(INTERL', length(HDchips), 1);
[DataOut Metric] = VitDec(G, HDchips, 1);
function [xx, BestMetric] = VitDec(G, y, ZeroTail);
%此函数是实现硬判决输入的Viterbi译码
%G为生成多项式的矩阵
%y为输入的待译码序列
%Zer为判断是否包含0尾
%xx为Viterbi译码输出序列
%BestMetric为最后的最佳度量
L = size(G, 1);                                    %输出码片数
K= size(G, 2);                                     %生成多项式长度
N = 2^(K-1);                                       %状态数
T = length(y)/L;                                   %最大栅格深度
OutMtrx = zeros(N, 2*L);
for s = 1:N
    in0 = ones(L, 1)*[0, (dec2bin((s-1), (K-1))-'0')];
    in1 = ones(L, 1)*[1, (dec2bin((s-1), (K-1))-'0')];
    out0 = mod(sum((G.*in0)'), 2);
    out1 = mod(sum((G.*in1)'), 2);
    OutMtrx(s, :) = [out0, out1];
end
PathMet = [0; 100*ones((N-1), 1)];
PathMetTemp = PathMet(:,1);
Trellis = zeros(N, T);
Trellis(:,1) = [0 : (N-1)]';
y = reshape(y, L, length(y)/L);
for t = 1:T
    yy = y(:, t)';
```

```
    for s = 0:N/2-1
        [B0 ind0] = min( PathMet(1+[2*s, 2*s+1]) + [sum(abs(OutMtrx(1+2*s, 0+[1:L])...
 - yy).^2); sum(abs(OutMtrx(1+(2*s+1), 0+[1:L]) - yy).^2)] );
        [B1 ind1] = min( PathMet(1+[2*s, 2*s+1]) + [sum(abs(OutMtrx(1+2*s,...
 L+[1:L]) - yy).^2); sum(abs(OutMtrx(1+(2*s+1), L+[1:L]) - yy).^2)] );
        PathMetTemp(1+[s, s+N/2]) = [B0; B1];
        Trellis(1+[s, s+N/2], t+1) = [2*s+(ind0-1); 2*s + (ind1-1)];
    end
    PathMet = PathMetTemp;
end
xx = zeros(T, 1);
if (ZeroTail)
    BestInd = 1;
else
    [Mycop, BestInd] = min(PathMet);
end
BestMetric = PathMet(BestInd);
xx(T) = floor((BestInd-1)/(N/2));
NextState = Trellis(BestInd, (T+1));
for t=T:-1:2
    xx(t-1) = floor(NextState/(N/2));
    NextState = Trellis( (NextState+1), t);
end
if (ZeroTail)
    xx = xx(1:end-K+1);
end
```