实验 1 调制解调

信息 005 王靳朝 2206113602

一. 实验内容

1.2ASK 幅移键控

2.2FSK 频移键控

3.2PSK 相移键控

4.QAM 调制

二. 实验原理

2.1 IQ 调制基本原理

IQ 调制是经常使用的调制技术。它使用两路正交的载波信号（即正弦和余弦信号），将原始基带信号分为 I 路和 Q 路，然后将它们与正弦和余弦信号进行乘法运算，得到两个调制信号，之后将两个已调制信号求和得到。在解调时，分别将正弦和余弦载波与 I 路和 Q 路信号相乘实现频谱反搬移，滤波之后将低频两路信号相加即可恢复原始基带信号。

此外 IQ 调制在信道中传输的仅是单边带信号，只存在正频率分量。统一根据三角公式，只要控制 IQ 两路信号的幅度，合成之后即可控制已调载波的幅度和相位，从而传输更多信息。

2.2 2ASK、2FSK、3PSK 调制解调原理

2ASK(幅移键控)是指用数字信号 01 码流来调制载波的幅度。一般情况下，数字基带信号可以表示为：

$$s(t) = \sum a_n \, g(t - nT_s)$$

其中：

$$a_n = \begin{cases} 1，概率为p \\ 0，概率为\ 1-p \end{cases}$$

$g(t)$是宽度为T_s的门函数，如此可以用数字基带信号即为高低电平，可以表示 0-1 码流。将数字基带信号和余弦载波相乘即可实现 2ASK 数字调制。

2ASK 解调时可以使用同步解调，和正弦载波相乘之后低通滤波实现频谱恢复基带信号。

2FSK 频移键控，指用数字信号控制载波频率的调制方式，即用载波的频率来传送数字信息，2FSK 便是指符号“0”对应于载频f_1 ，而符号“1”对应于载频 f_2 的调制方式。 一般情况下，载频f_1和载频 f_2相差不会太大从而增大系统带宽。

2FSK 可以使用相干或者非相干的方式解调，此外 2FSK 信号还可以使用过零比较进行解调。

2PSK 相移键控是用数字信号 0、1 来改变载波信号的相位值的调制方法，2PSK 中，当发送数字信号 0 时，使载波发送初相为 0 的波形，当发送数字信号 1 时，使载波发送初相为π的波形。解调时，为准确恢复相位，需使用相干解调，用与载波同频同相的信号与接收信号相乘以得到发送信号。

2PSK 必须使用同步解调进行解调，和本振相乘实现极性比较，从而恢复绝对码型。

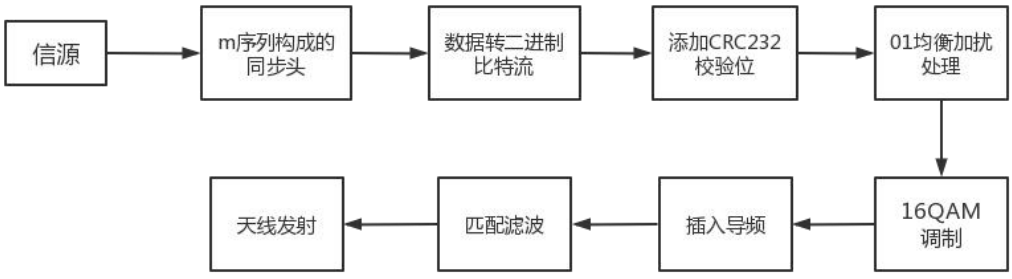
2.3 QAM 调制原理

正交幅度调制是一种在两个正交载波上进行幅度调制的调制方式。这两个载波通常是相位差为 90 度($\frac{\pi}{2}$)的正弦波，因此被称作正交载波。这种调制方式因此而得名。

QAM 发射信号集可以用星座图方便地表示。星座图上每一个星座点对应发射信号集中的一个信号。设正交幅度调制的发射信号集大小为 N，称之为 N-QAM。星座点经常采用水平和垂直方向等间距的正方网格配置。数字通信中数据常采用二进制表示，这种情况下星座点的个数一般是 2 的幂。常见的 QAM 形式有 16-QAM、64-QAM、256-QAM 等，星座点数越多，每个符号能传输的信息量就越大。但是，如果在星座图的平均能量保持不变的情况下增加星座点，会使星座点之间的距离变小，进而导致误码率上升。

三. 实验过程

3.1 发送端整体框图



3.2 发送端数据帧结构

发送数据包含：m 序列同步头、导频、信息位、CRC32 校验序列，其中，信息位和校验序列还需经过扰码器加扰。

信息位由一段长为 380 的字符串产生，将字符串转换为 ASCII 码后，信息位长 380*8=3040 bits。

由这些信息位产生 CRC32 校验序列，其产生原理为在二元域上用这些信息位除以 CRC32 指定的多项式，取其 32 位余数作为校验序列，将 32 位余数放置到信息位后，其长为 32 bits。

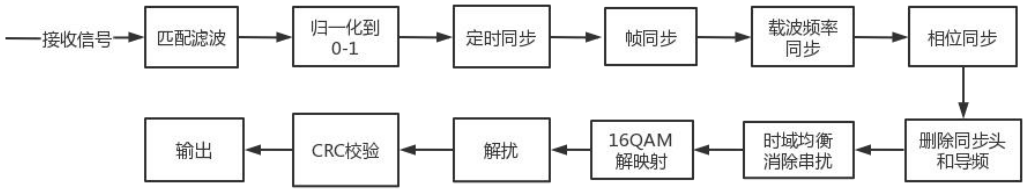
然后对信息位和校验序列构成的序列进行加扰。加扰过程采用乘法（自同步）扰码器。加扰原理在报告最后叙述。

信息位、校验序列按照 64QAM 调制，得到序列长度为(3040+32)/6=512，然后插入导频。

导频序列为[1+j -1-j 1+j -1-j -1-j 1+j -1-j 1+j]，共 8 symbols，将上段中的序列每 64 位插入一段导频，即总共插入 8 段导频，序列长度变为 8*8+512=576 symbols。最后在序列前插入训练序列。

m 序列同步头长 127 bits，是由 7 位移位寄存器产生的 m 序列,具体为：
1,1,1,1,1,1,1,0,1,0,1,0,1,0,0,1,1,0,0,1,1,1,1,0,1,1,1,1,0,1,0,0,1,0,1,1,0,0,1,0,1,1,0,1,0,1,1,0,1,1,0,0,1,0,1,0,0,1,0,0,1,0,0,0,1,1,1,1,0,0,0,0,0,1,0,1,1,1,1,1,1,0,0,1,0,1,0,1,0,1,1,1,1,0,0,
1,1,0,1,0,0,0,1,0,0,1,1,1,1,1,0,0,0,1,0,1,0,0,0,0,1,1,0,0,0,0,0,0,1,0,0,0,0,0,0，将训练序列按 BPSK 调制，并放入序列最前方，得到长为 576+127=703 symbols 的发送号集。

3.3 接收端整体框图



在接收端，经定时同步和帧同步后，收到的信号可以表示为$x(n)e^{j(\Delta\omega nT+\theta)}$， 其中$x(n)$为发送信号，$\Delta\omega nT$为由于频偏造成的误差，$\theta$为由于初相不同造成的偏差，因此需要经过三次载波同步和一次相位同步来消除该误差。

为了消除码间串扰，可再经过时域均衡后进行解码。因为发送端进行了加扰，因此在接收端进行解扰，解扰与加扰为逆过程，加扰采用了乘法（自同步）加扰，因此接收端采用加法（同步）加扰法进行解扰。 最后进行 CRC 校验，此例中将所得数据的最后 32 位去掉后，在二元域除以 CRC32 的生成多项式，若得到的余数与收到数据的最后 32 位相同，则说明接收数据判决正确，否则说明产生误码。

3.4 接收时对帧的拆解

接收时，在同步之后首先对 m 序列同步头进行拆解，然后按每 64+8=72 位得到数据帧，其中 64 位数据位，8 位导频序列，然后将导频序列删除。

3.5 QAM 调制解调的具体实现

调制解调过程如下：

```
function txdata = qam16_tx_func
%% train sequence  （同步序列 m序列）
seq_sync=tx_gen_m_seq([1 0 0 0 0 0 1]);
sync_symbols=tx_modulate(seq_sync, 'BPSK');
%% message 128-4 byte
msgStr=[
    'a--------------a',...
    '-b------------b-',...
    '--c----------c--',...
    '---d--------d---',...
    '----e------e----',...
    '-----f----f-----',...
    '------g--g------',...
    '-------h--------',...
    'a--------------a',...
    '-b------------b-',...
    '--c----------c--',...
    '---d--------d---',...
    '----e------e----',...
    '-----f----f-----',...
    '------g--g------',...
    '-------h----',...
    ];
%% string to bits       二进制序列
mst_bits=str_to_bits(msgStr);
%% crc32               crc校验
ret=crc32(mst_bits);
inf_bits=[mst_bits ret.'];
%% scramble            加扰（让0 1 均匀分布）
scramble_int=[1,1,0,1,1,0,0];
sym_bits=scramble(scramble_int, inf_bits);
%% modulate       调制 星座映射
mod_symbols=tx_modulate(sym_bits, '16QAM');
%% insert pilot      信息位前加同步序列 导频
data_symbols=insert_pilot(mod_symbols);
trans_symbols=[sync_symbols data_symbols];
%% srrc      成型滤波
fir=rcosdesign(1,128,4);
tx_frame=upfirdn(trans_symbols,fir,4);
tx_frame=[tx_frame, zeros(1, ceil(length(tx_frame)/2))];
txdata = tx_frame.';
%% display
plot(real(tx_frame));
hold on
plot(imag(tx_frame));
end
```

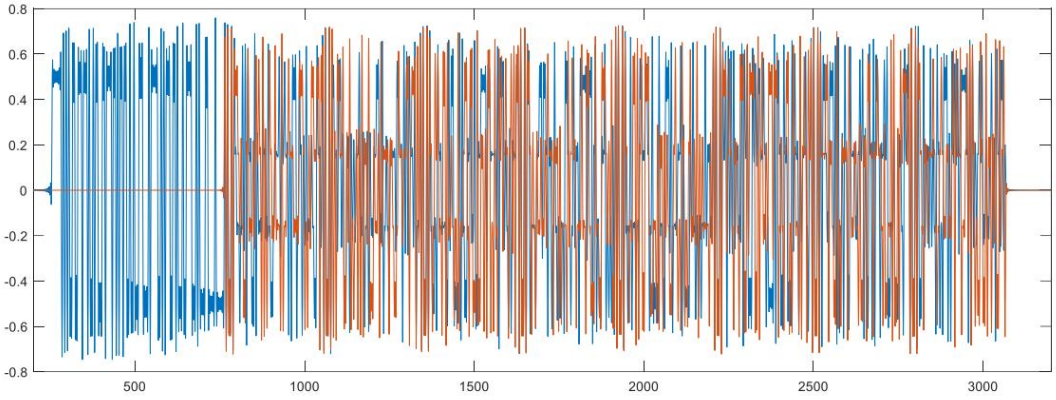
在接收端：

```
function qam16_rx_func(rxdata)
global cyc;
seq_sync = tx_gen_m_seq([1 0 0 0 0 0 1]);
local_sync = tx_modulate(seq_sync, 'BPSK');
rx_signal=rxdata;
%% matched filtering       匹配滤波
fir = rcosdesign(1,128,4);
rx_sig_filter = upfirdn(rx_signal,fir,1);
%% normalization           归一化
c1=max([abs(real(rx_sig_filter.')),abs(imag(rx_sig_filter.'))]);
rx_sig_norm=rx_sig_filter ./c1;
%% sampling synchronization    时间样点同步
[time_error,rx_sig_down]=rx_timing_recovery(rx_sig_norm.');
% rx_sig_down=rx_sig_norm(1:4:end).';
%% package search
[rx_frame,cor_abs,th_max,index_s]=rx_package_search(rx_sig_down,local_sync,703);
%% coarse freq synchronization
coarse_sync_seq=rx_frame(1:8);
[deltaf1,out_signal1] = rx_freq_sync(coarse_sync_seq,4,rx_frame);
%% first fine freq synchronization
fine_sync_seq_1=out_signal1(1:120);
[deltaf2,out_signal2] = rx_freq_sync(fine_sync_seq_1,2,out_signal1);
%% second fine freq synchronization
fine_sync_seq_2=out_signal2(1:120);
[deltaf3,out_signal3]=rx_freq_sync(fine_sync_seq_2,2,out_signal2);
deltaf=deltaf1+deltaf2+deltaf3;
%% initial phase estimate
[out_signal4,ang]=rx_phase_sync(out_signal3,local_sync);
%% phase track
rx_no_syn_seq=out_signal4(127+1:end);
[out_signal6,phase_curve]=rx_phase_track(rx_no_syn_seq);

%% delete pilot
out_signal7=rx_delete_pilot(out_signal6);
%% time domain equalize
out_signal8=rx_time_equalize(out_signal7);
%% signal demod       解扰
[soft_bits_out,evm] = rx_qam16_demod(out_signal8);
Si=[1 1 0 1 1 0 0];
m=0;
for i=1:length(soft_bits_out)
    [c,Si]=descramble(soft_bits_out(i),Si);
    m=m+1;
    y(m)=c;
end
soft_bits_out=y;
```

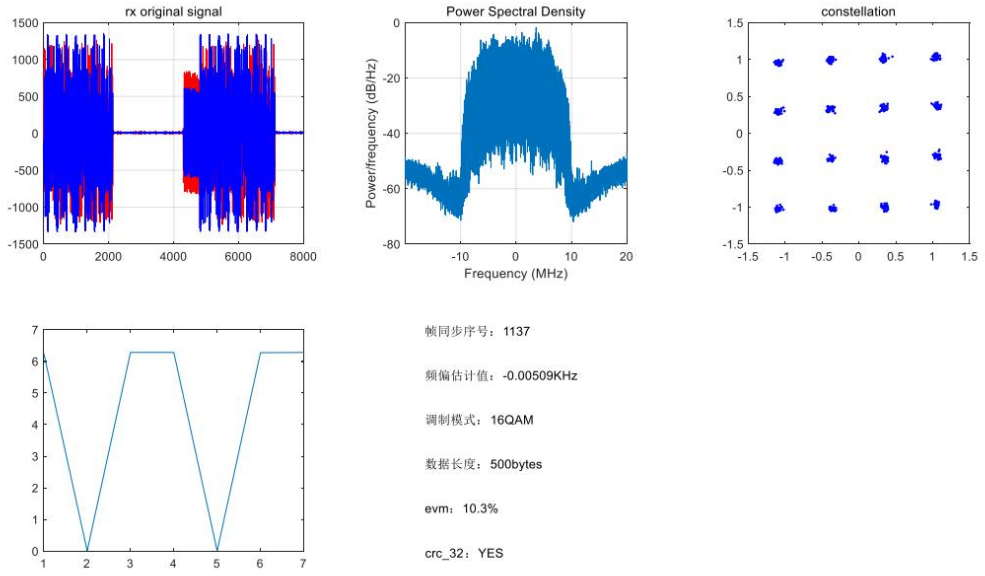
```
%% crc32 check        校验
ret=crc32(soft_bits_out(1:length(soft_bits_out)-32)).';
crc_bits_32=soft_bits_out(length(soft_bits_out)-31:length(soft_bits_out));
crc_outputs=sum(xor(ret,crc_bits_32),2);

if crc_outputs==0
    crc_32='YES';
else
    crc_32='NO';
end

msg=soft_bits_out(1:end-32).';
w = [128 64 32 16 8 4 2 1];            %%信源解码
Nbits = numel(msg);
Ny = Nbits/8;
y = zeros(1,Ny);
for i = 0:Ny-1
    y(i+1) = w*msg(8*i+(1:8));
end
a=[y zeros(1,4)];
b=reshape(a,16,8*2);

%% display
figure(2);clf;
subplot(231);
plot(real(rx_signal),'r');
hold on;
plot(imag(rx_signal),'b');
grid on;
title('rx original signal');
subplot(232);
pwelch(rx_signal,[],[],[],40e6,'centered','psd');

% text(0.15,1.0,['帧同步序号: ',b.']);
disp(char(b.'));
% plot(phase_curve);
axis square;
subplot(233);
plot(real(out_signal8),imag(out_signal8),'b.');
title('constellation');
axis([-1.5 1.5 -1.5 1.5]);
axis square;
subplot(234);
plot(phase_curve+pi*2);
subplot(235);
text(0.15,1.0,['帧同步序号: ',num2str(index_s,5)]);%,'FontSize',12
text(0.15,0.8,['频偏估计值: ',num2str(deltaf/1e3,3),'KHz']);
text(0.15,0.6,['调制模式: ', '16QAM']);
text(0.15,0.4,['数据长度: ', '500','bytes']);
text(0.15,0.2,['evm: ', num2str(evm.*100,3),'%']);
text(0.15,0.0,['crc\_32: ', crc_32]);
axis off;
% % figure(2)
% plot(cor_abs);
```

四. 实验结果与分析

运行程序得到的结果如下：



发送端的发送信号



接收端的接收结果

接收端收到的波形如图所示，第一幅图为收到的原始信号；第二幅图为信号的功率谱，可见其功率主要集中在-10MHz~10MHz 之间；第三幅图为接收信号星座图，可见清晰的 16QAM 的结构，说明信道条件很好，信噪比很高；第四幅图为每一帧的初始相位。

五. 思考题

1. 实验中的符号率、采样率、比特率各是多少？之间有怎样的对应关系？

码元速率 10M /Hz，采样率 40Mhz，比特率 40M b/Hz

由于存在四倍增采样，实际码元速率为 40/4=10M。比特率为符号率$*\log_2 M$（M 为符号个数，且独立等概率出现）

2. 你实现的 QAM 映射是格雷映射吗？格雷映射有什么优点？画出你的 QAM 星座图

实现的 QAM 映射是格雷映射。

QAM 信号点通常使用格雷码映射进行排列，这种排列方式可以最大程度地减小符号之间的误码率。使用格雷码映射可以保证任意相邻的两个信号点之间只有一个位差异，因此它大大地减少了由一个状态到下一个状态时逻辑的混淆，这样与其它编码同时改变两位或多位的情况相比更为可靠，即可减少出错的可能性。

QAM 星座图为：

Q
I
$S_3 S_2 S_1 S_0$
0101
0111
0110
0100
1011
1111
1110
1100
1001
1101
1010
1000
0001
0011
0010
0000
A_1
$-A_1$
A_2
$-A_2$