软件无线电系统基础设计

设计原理

1. 软件无线电的兴起和发展现状

从 1890 年左右第一次无线传输开始，无线电传输技术不断发展。

1930 年中期，模拟语音通信出现；50 年代，模拟电视广播出现，它消耗了更多带宽，提供了丰富的客户体验；60 年代，随着计算机变得越来越小，功能越来越强大，计算机开始成为远距离通信媒体，既使用通过 ARPANET 的有线连接，也使用无线卫星 ALOHANET 的连接；手机在同一时期出现，用户可以在任何公共场所或车辆上建立无线语音通信。

尽管多种技术实现了发展，但所有提到的设备都存在一个潜在的问题，即它们的无线电和协议大多是基于硬件的，因此，在无线电功能方面，重新编程或重新配置是困难的。这意味着它们缺乏灵活性，一旦硬件、固件或软件出现错误，那么通常没有合理的方法来纠正问题。事实上，这些设备的功能通常仅限于硬件组件，并且不能重新配置以执行硬件本身提供的以外的其他无线协议。

软件无线电旨在为上述问题提供解决方案，并提供许多其他好处。软件定义无线电(SDR)是一种用于无线通信设备的设计范例。它的创造者 Joseph Mitola 在 90 年代初将这个词定义为一类可以通过软件重新编程和配置的无线电的标识符[1]。他设想了一种理想的软件定义无线电，如图 1 所示，其物理组件只有天线和接收器侧的模数转换器(ADC)。同样，发射机也要有一个数字模拟转换器(DAC)和一个发射天线。其余的功能将由可重新编程的处理器处理。

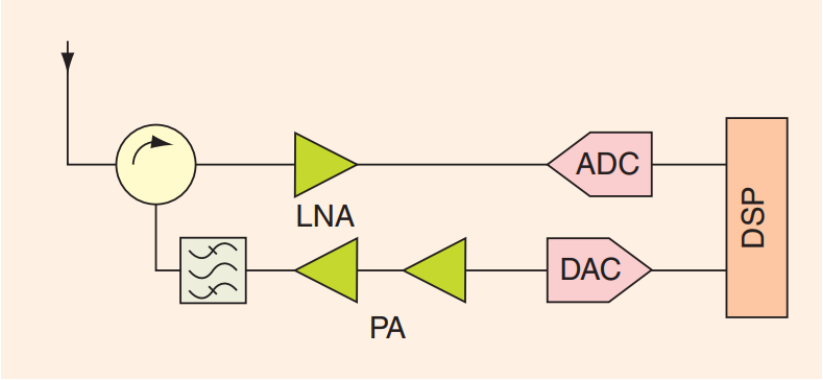


图 1 软件无线电概念的通用实现

与大多数技术一样，SDR 也从军事环境演变到民用环境。第一个可操作的 SDR，被称为 Speakeasy 在 1991 年和 1995 年之间由美国海军开发[2]，但它不能与它所设计的硬件以外的设备一起使用。而 SpeakeasyII，由于电子学、无线通信电路以及可重用和模块化编程技术的发展，取得了更大的成功。

在实际产品中利用 SDR 技术的进展比几年前预期的要慢。早在 2002 年就有人预测，到 2006 年，在商业移动终端中将“广泛采用 SDR 作为基线设计”，当然，这并没有发生。此外，美国政府的 JTRS 计划自 1997 年初始化以来，在不同时期“经历了成本和进度超支和性能不足”[3]。直到 2007 年，在 SDR 方面开始出现了一些积极的迹象和成就，这表明 SDR 越来越接近在商业产品中被更大规模地采用。例如，Vanu 公司获得 FCC 批准的

“Anywave”基站，第一个获得豁免的“SCA”军事通信产品 Thales AN/PRC-148 JEM，以及第一个获得豁免的“SCA”Harris Falcon III(TM) AN/PRC-152(C)。市场上还提供了越来越多的 SDR 研究和原型平台，以及 SDR 开发工具。

SDR 本身是一种新兴技术，它的前景与智能天线、网络、软件、半导体、信号处理和电池技术等其他重要的新兴技术有关。固态集成电路(IC)技术的快速发展正在推动商用无线通信系统的发展。这些新兴技术和进步正在使 SDR 在技术和商业上变得现实。今天，SDR 软件和硬件都可以以非常低的价格获得，这促使我们考虑在无线电解决方案中引入这种模式。

随着移动通信系统的发展，无线电技术也发生着演进：由 1G 所代表的模拟无线电技术，发展到 2G、3G 所分别代表的窄带数字无线电、宽带数字无线电技术，到 4G/5G 时代已经全面推广使用的软件无线电技术。

2. 软件无线电中的关键技术及其最新发展

2.1 可重构无线电接收机结构

接收机应将低功率的 RF 输入信号转换为复杂的基带信号。设计人员必须考虑所有的性能参数：动态范围、三阶交调点、噪声系数、增益、镜频抑制、滤波、信噪比和无杂散动态范围。这些问题在传统的固定频率、固定带宽收发器中很容易解决。但是可重构无线电不是这样，因为输入功率电平、频率、带宽和调制方案都是可变的。理想的 SDR 必须覆盖一定的中心频率范围，具有灵活的带宽，在较宽的动态范围内发射和接收信号，同时保持无杂散。

接收机架构根据用于将射频信号转换到基带的级的数量和类型来区分。

(i) 直接转换接收机(Direct Conversion Receiver)

这种接收机在单级完成接收射频信号到基带的正交下转换，在接收机架构中具有最低的复杂度。RF 预选择滤波要求不高，且不需要中频带通滤波器。如图 2 所示。由于正交下转换是在射频进行的，正交本振(LO)的振幅和 90°相位平衡应该保持到 6GHz，这在目前的 RF 技术中是不可能的。因此，在宽带直接转换接收机应用中，需要针对 SDR 相位不平衡校正的技术。

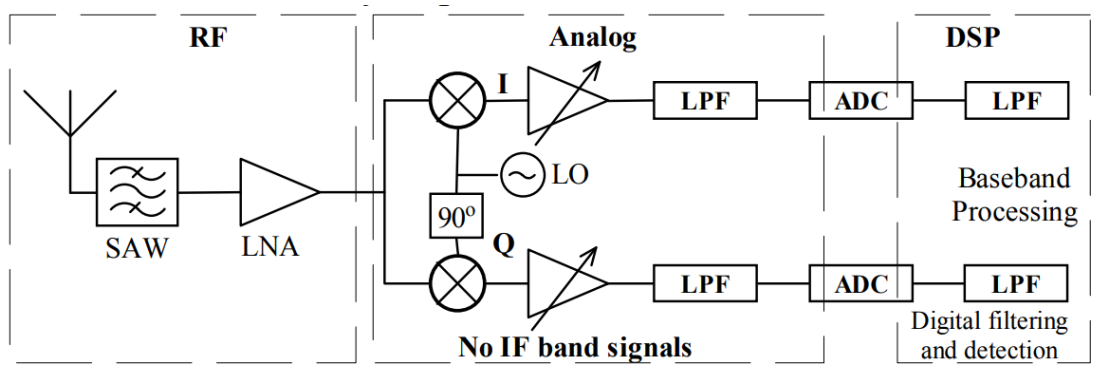


图 2 直接转换接收机架构

(ii) 超外差接收机(Superheterodyne Receiver)

该架构将输入射频频率下转换为中频，然后执行正交下转换。它具有良好的频率选择性，这得益于每个混频阶段之前和之后的滤波。如图 3 所示。正交下转换应用于 IF 信号，

因此，正交本振(LO)可以选择一个有良好的幅度和相位平衡的频率。几十年来，这种架构一直是主要的选择，然而，由于多个混频和滤波阶段，它的复杂性很高。由于在一个频率进行正交下转换，RF 信号需要下转换到相同的 IF。这限制了接收机的灵活性，使其不能同时容纳多个标准。

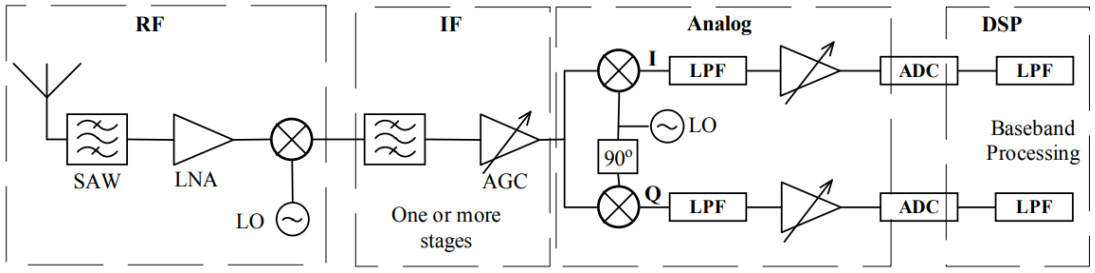


图 3 超外差接收机架构

(iii) 数字中频接收机(Digital IF Receiver)

类似于外差结构，但正交下转换是在数字中频载波上进行的。它通过更灵活的实现改善了了外差接收机的缺点。它可选择多个 IF 频带，用同一个高采样率 ADC 对其进行数字化处理。它使用了数字正交信号，和数字滤波，消除了下转换过程中相位不平衡的问题。如图 4 所示。这种结构可能是目前可重构接收机设计的最佳选择。

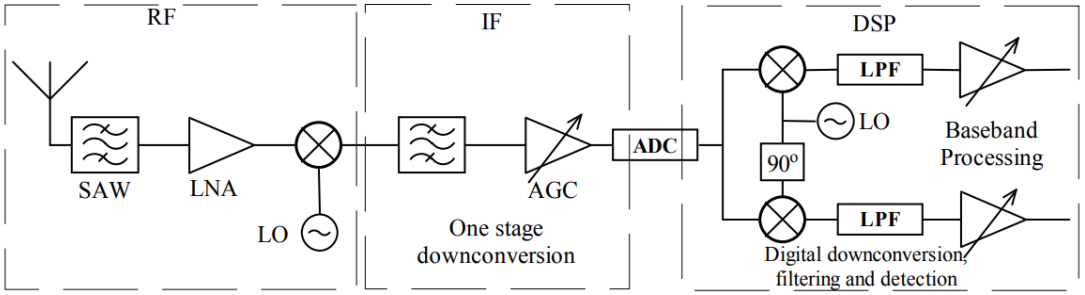


图 4 数字中频接收机架构

2.2 先进的无线电技术

(i) 模拟到数字转换器(Analogue to Digital Converters)

A/D 和 D/A 变换器在软件无线电系统所处的位置是非常关键的，它直接反映了软件电台的软件化程度。决定宽带模数转换器性能的关键是采样率和采样位数，采样速率由信号带宽决定，而量化位数则要求满足一定的动态范围和精度。在理想的 SDR 中，接收信号经过天线在 RF 位置进行数字化处理。然而，在微波频率执行直接数字化目前是无法实现的。当分辨率和采样率较高时，ADC 的功耗会急剧增加。以 ADC 技术目前的发展速度，RF 数字化还需要一段时间。

在一般的通信系统中所用到的信号都是带通信号，采用带通采样的方法可以大幅度降低数据速率。但在多个频带均存在信号的情况下，带通采样可能造成频谱混叠，通过带通滤波器来滤出特定的信号是一种可行的解决方法，但其又会增加设计的复杂程度。从另一方面来说，采样速率的提高，也可以提高采样量化的信噪比。因此，A/D 变换器采样率、采样精度、成本等因素的均衡对于数字 SDR 的设计至关重要。

(ii) 天线

天线通常设计为单频段工作。采用宽带天线会降低接收机的灵敏度，导致低噪声放大器(LNA)和混频器的输入信号噪声更大，它们的噪声抑制要求更高，因此这些设备将需要高线性度。目前，有几种类型的宽带天线可覆盖多个倍频带宽，但它们的物理几何形状不能在移动终端中被使用[5]。文献[6]中呈现了一种双端口/双频微带天线几何结构，具有较宽的工作带宽。这种架构可以进行扩展，以提供更灵活的天线结构。以目前的技术，宽带或可调谐天线不能覆盖 900MHz 到 5.5GHz 的频率，因此在 SDR 接收机中使用多天线是不可避免的。

(iii) RF 预选择滤波器

RF 级最高的滤波要求在 2GHz 频带左右。在 1805MHz 到 2175MHz 之间，有几个通信标准是重叠的。这些标准占用的频带从 15 到 75MHz 不等。为了获得更好的选择性和抗干扰能力，理想的选择是具有可变带宽和中心频率的可调谐 RF 滤波器。根据正在进行的研究，这可能在未来几年通过使用微机电系统(MEMS)技术实现[7]。

(iv)中频放大器

IF 放大器的选择不像 LNA 和混频器等其他元件那么困难。有几种高动态范围的 VGA，具有较高的三阶交调点(27-31dBm)，增益(高达 55dB)和低的噪声系数(4-7dB)。

(v)中频滤波器

如果信道化是从 IF 开始的，那么这个阶段的滤波器的带宽最好在 200kHz、1MHz、1.78MHz、5MHz 和 20MHz 左右，这些都是目前通信标准的信道带宽。有大量的声表面波(SAW)滤波器可以用于这一目的，它们可以从各种制造商获得不同的中心频率和带宽。可以假设，具有所需规格的 IF 滤波器可以很容易地得到。如果在 SDR 的设计中使用 SAW 滤波器，则需要一个滤波器组和一个开关机构。同样，可调滤波器将消除这些组件，并显著降低复杂性。

2.3 镜频抑制和滤波

传统接收机采用固定滤波器，通过适当的 RF 预选择滤波来抑制镜频信号。在 SDR 中，需要电子可调滤波器来实现这一功能。无线电设计中的滤波要求与所选择的 IF 频带密切相关。在可重构接收机中，很难选择一个同时对所有频带的信号都有用的 IF。一般的，RF 频率越高，保持镜频抑制性能所需的 IF 频带就越高。然而，增加 IF 频带的频率将需要更高采样率的 ADC。为了在 5.2GHz 无线局域网频段实现令人满意的效果，可能需要额外的频率转换。

2.4 高速数字信号处理(DSP)技术

高速 DSP 芯片是软件无线电的核心部分，主要负责完成电台内部数据处理、调制解调和编码解码等工作。由于电台内部数据流量大，进行滤波、变频等运算次数多，必须采用高速、实时、并行的数字信号处理模块或专用集成电路才能达到要求。

目前，可采用的技术方案主要是数字信号处理技术(DSP)、专用集成电路(ASIC)、现场可编程门阵列(FPGA)等，实际应用中往往会综合考虑运行速度、功耗、成本等因素将多种技术结合使用。利用 FPGA 来设计 DSP 核也是一种可行的方法，其在保存优良性能和灵活性的同时极大地降低了功耗。

2.5 干扰消除和线性化技术

输入信号最小时，宽带接收机平均输出-55dBm 的信号电平和约 45dB 的增益。以 GSM 为例，GSM 需要 9dB 的载波干扰比才能达到 10^{-3} 比特错误率(BER)，那么在接收端需要将 IM3 抑制到-64dBm。根据从 0.4-6GHz 的室外测量试验[8]，接收机的天线端口存在最高-13dBm 的潜在干扰信号。如果该信号不能被滤除，则需要保证接收机的 IM3 低于-64dBm。这要求接收机三阶交调功率有 80dBm，即使采用放大器和混频器线性化技术也很难实现，如图 5 所示。在可重构无线电设计中，除非干扰信号被可调谐的 RF 滤波器滤除，否则干扰抑制技术是必不可少的[9]，如图 6 所示。

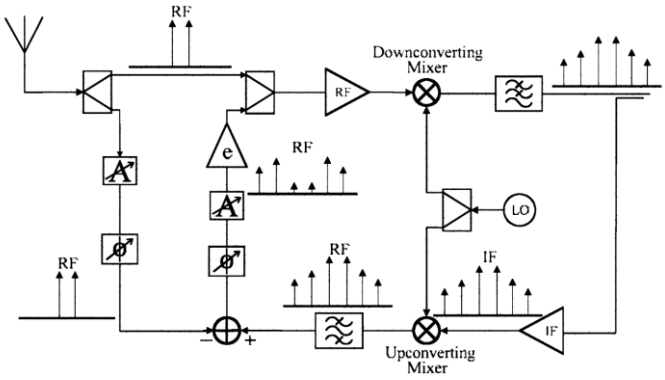


图 5 一种采用频率重平移来改善射频前端线性度的接收机

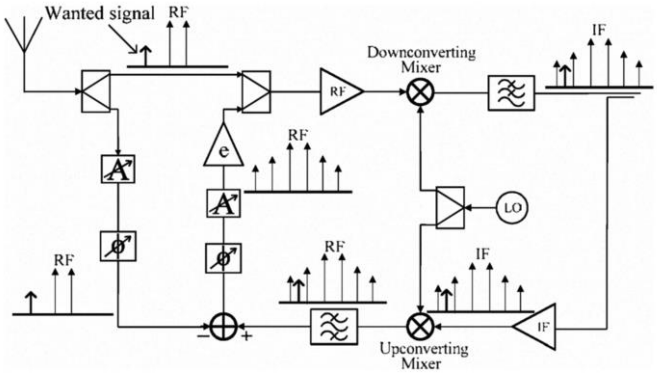
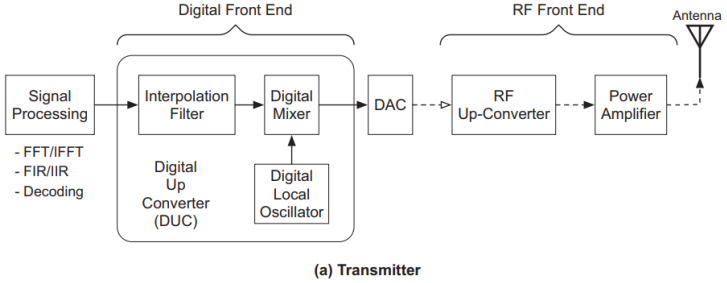


图 6 频率重平移用于接收机中的干扰抑制

3. 软件无线电系统的基本结构及其最新发展状况

如图 7 (a)和(b)所示，在高电平下，典型的 SDR 收发信机由信号处理、数字前端、模拟射频前端和天线组成。



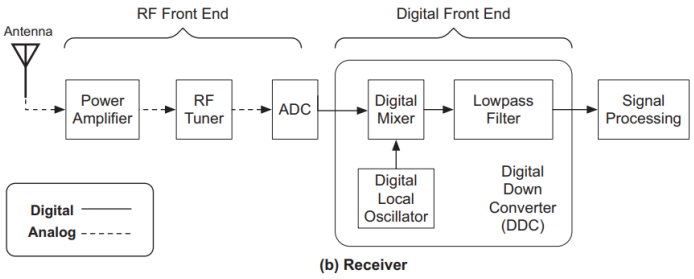


图 7 SDR 架构 (a)发送方 (b)接收方

3.1 天线

SDR 平台通常采用多根天线覆盖较宽的频段。天线能够选择频带并适应移动跟踪或干扰消除。在 SDR 的情况下，天线通常需要满足一定的要求，如自适应(即灵活地调谐到多个波段)，自对准(即波束形成能力)和自修复(即干扰抑制)[10]。

3.2 射频前端

射频前端的主要功能是在不同的工作频率发射和接收信号。它的另一个功能是将信号转换为中频(IF)信号。操作过程分为两个部分，取决于信号的方向(即 Tx 或 Rx 模式)[11]。

(i)在发射路径中，数字样本被数模转换器(DAC)转换成模拟信号，然后再传输到射频前端。该模拟信号与预设的 RF 频率混频，调制，然后发射。

(ii)在接收路径中，天线捕获 RF 信号。天线使用匹配电路连接到射频前端，以保证最佳的信号功率传输。通过位于天线附近的低噪声放大器，以放大微弱信号并将噪声水平降至最低。这个放大的信号与本地振荡器的信号一起被送入混频器，以便将其降转换到中频(IF)信号。

3.3 模数转换器

ADC 位于接收机侧，是无线电接收机的重要组成部分。

ADC 负责将连续时间的信号转换为离散时间的二进制编码信号。ADC 性能可以用各种参数来描述[12]，包括:信噪比、分辨率、每个样本的比特数、无杂散动态范围(SFDR)即载波信号与下一个最强噪声分量或 spur 的强度比、功耗等。软件无线电开发的进步为 ADC 性能的提高提供了动力。例如，由于 ADC 的功耗会影响电池供电的软件无线电的寿命，更多的节能 ADC 被开发出来。

3.4 数字前端

数字前端执行两个功能[13]:

(i)采样率转换(SRC)，将采样率从一个速率转到另一个。这是必要的，因为两个通信方频率必须同步。

(ii)信道化，分别包括发射端和接收端的上/下转换。还包括信道滤波，将按频率划分的信道提取出来。

在软件无线电收发机中，数字前端执行如下任务:

(i)在发射侧，数字上转换器(DUC)将基带信号转换为中频信号。DAC 与 DUC 连接，然后将数字 IF 采样转换为模拟 IF 信号。然后，RF 上转换器将模拟 IF 信号转到 RF 频率。

(ii)在接收端，ADC 将中频信号转换为数字样本。这些样本随后被送到下一个区块，即数字下转换器(DDC)。DDC 包括一个数字混频器和一个数控振荡器。DDC 从 ADC 中提取基带数字信号。该数字基带信号经过数字前端处理后，转发给高速数字信号处理块[4]。

传统方法的另一种替代方法是直接 RF 采样(Direct RF Sampling，DRFS)。在 DRFS 中，RF 采样 ADC 取代了混频器、本振和滤波器等模拟处理模块，将处理转移到数字域，如图 8 所示。这极大地改进了接收机的设计。在这里，信号被 ADC 转换后交给信号处理块，以便提取数据[14]。基于子采样或带通采样理论，利用信号的混叠来采样，它要求的采样率要低得多。带通滤波器放置在 ADC 的前面，以避免灵敏度损失。DRFS 的优点是支持非常宽的带宽和更高的功率效率[15]。

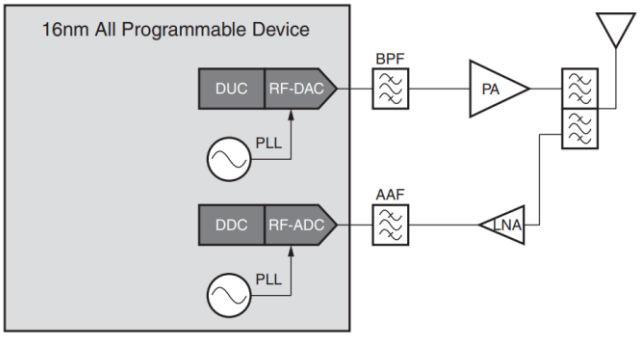


图 8 使用直接 RF 采样的软件定义无线电

3.5 信号处理

信号处理操作，如编码/解码、交织/解交织、调制/解调和扰码/解扰在该模块中执行。对信道的编码包含了纠错码，具体地说，编码后的信号包括冗余，接收方的解码器利用冗余从已损坏的接收信号重建原始信号。纠错码的例子包括卷积码、Turbo 码和低密度奇偶校验(LDPC)[16]。由于数据传输和存储方案的使用，解码器构成了信号处理块中计算最密集的部分[17]，如图 9 所示。在面积和功率方面，调制/解调部分被认为是高度复杂和昂贵的，采用快速傅里叶变换(FFT)和反 FFT(IFFT)调相。

信号处理块通常被称为基带处理块。在讨论 SDR 时，基带块是讨论的核心，因为它实现了大部分数字域的算法。这种实现一方面基于能够有效处理信号的硬件电路之上，如 ASIC，FPGA，DSP，GPPs 和 GPU 等。一方面基于软件编程，负责信号处理操作所需的函数和高级抽象。

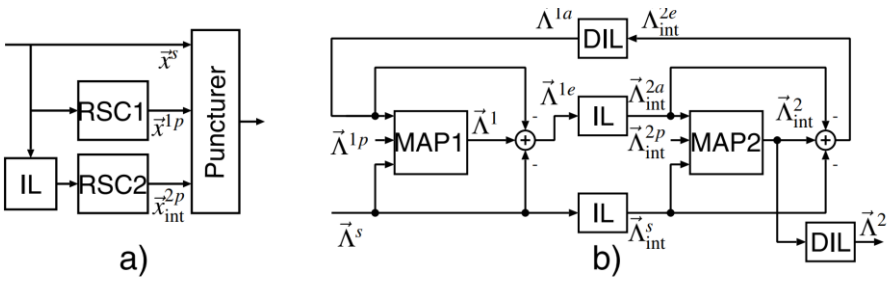


图 9 a) Turbo 编码器 b) Turbo 解码器

二、 设计目的

(1)掌握软件无线电技术的发展背景，基本概念及基本结构。
(2)通过学习软件无线电技术，掌握通信系统的结构及处理过程。
(3)利用 MATLAB 工具搭建软件无线电处理平台，进行信号处理仿真。

三、 设计内容

基于设计要求：根据 SPR（Software Programming Radio 软件编程无线电）或 SDR（Software Defined Radio 软件定义无线电）的系统结构，搭建一个软件无线电系统结构，利用 MATLAB 中 Simulink 工具或 M 语言，合理设计输入信号进行处理，仿真信号的处理过程。完成以下内容：
(1)需求分析（系统要包含哪些模块，取什么样频段进行仿真？应设置什么样的输入信号，待观察的输出信号形式什么样的？等等）
(2)方案设计（通过需求分析讨论出的几种方案（两种及以上））
(3)方案评审（每种方案优劣的讨论及最终选定某种方案的原因，及对方案的完善）
(4)实验研究（对所选定方案中各模块的设计及实现）
(5)数据测试（测试数据的选定，测试结果的显示）
(6)结果分析（测试结果分析）
(7)方案改进（根据结果反映出的问题，对方案进行改进）
(8)报告撰写（按要求完成相关内容）
(9)编写答辩 PPT（条理清楚，重点突出地将问题阐述清楚）

四、 设计结果

1. 完成课程设计的各步骤，并记录相应内容

1.1 需求分析

1.1.1 主要模块

发射机：信源编码、数据成帧、信道编码、脉冲成形、升采样、调制、射频前端
接收机：射频前端、解调、降采样、定时恢复、载波同步、抽样判决、信道译码、帧同步、信源译码

1.1.2 频段设计

基于现实软件无线电的频谱使用范围，可以假设使用频率范围 200-2000M 的调制信号，使用的 ADC 器件为 50-200M 采样速率的器件。
仿真过程可以同幅度减小上述数值，以减小最后仿真的计算量和计算耗时。
例如，假设软件无线电调制信号为 10-100k，ADC 采样速率为 10k。

1.1.3 信号形式

调试期间使用信号：随机产生的二进制信号
实际传输使用信号：音频信号、JPEG 格式的图片或 txt 格式的文本
待观察的输出信号：三种信源输入信后对应的二进制脉冲序列

1.2 方案设计

针对总体架构设计，设计重要的模块如下：
传输信号：文字、图片、语音
信源编码：哈夫曼编码、PCM 编码
信道编码：卷积码、线性分组码
调制方式：DQPSK、16QAM

1.3 方案评审

针对三种不同的信源，根据信号的特性，文字和图片可以采用哈夫曼编码，语音采用 PCM 编码。

由于我们的设计可以任意组合信道编码方式和正交调制解调方式，因此，我们选择以下两种方案。

1.图像信源，卷积码信道编码，16QAM 正交调制解调
2.文本信源，线性分组码信道编码，DQPSK 正交调制解调

1.4 实验研究

1.4.1 信源编码

信源有三种类型，文本、图像和音频，用不同的 type_info 标记，以便信源译码时确定译码方式。信源编码的代码如下。

```
function info = Sourcecode(type, text_max_len, img_size, fs, T)
% text_max_len = 8e3;  % 文本最长限制
% img_size = [60, 40]; % 图片大小
% fs = 4e3;            % 音频采样率
% T = 2;               % 音频录制时长

if type == 1
    type_info = [0,0,0,1,1,0,0,0].';
    info = get_text(text_max_len); % 文本信息
elseif type == 2
    type_info = [0,0,1,0,0,1,0,0].';
    info = get_img(img_size);      % 图像信息
elseif type == 3
    type_info = [0,1,0,0,0,0,1,0].';
    info = get_voice(fs, T);       % 音频信息
end
info = [type_info;info];
end
```

(1) 文本

对于文本，可以采取直接编码的方式。

通过 fileread() 函数读入文本信息并保存至类型为 char 的行向量 msgStr 中；通过 unicode2native() 函数将 unicode 编码（对于英文之外的文字，我们有一种统一的编码，叫做 Unicode 码，它几乎包含了所有国家的字符，其中就包括了大多数的汉字）转为本地(native)编码（ASCII 码）；用两个字节记录文本长度；使用 de2bi() 函数将数据转化为二进制信息序列。

文本采用直接编码的核心代码如下。

```
msgStr = fileread(pathfile);
native = unicode2native(msgStr);        % 转成本地编码

len = length(native);
b = mod(len, 256);
a = (len - b)/256;
msgBin = de2bi([a, b, native], 8).';    % 字节转成 8bit
text_code = double(msgBin(:));          % 转为 double 型列向量
```

除了直接将 ASCII 码转为八位二进制序列外，我们还可以使用霍夫曼编码来对 ASCII 码进行编码。

霍夫曼编码，又称为哈夫曼编码（Huffman Coding），是一种可变长编码（VLC, variable length coding）方式。比起定长编码来说，哈夫曼编码能节省很多的空间，因为每一个字符出现的频率不是一致的。是一种用于无损数据压缩的熵编码算法，通常用于压缩重复率比较高的字符数据。

霍夫曼编码的具体方法如下。把元素按出现概率进行从大到小排序，选取概率最小的两个元素赋以 0、1，并把这两个元素合并作为一个整体，概率为两者之和。然后再对，所有元素进行重新排序，重复以上步骤，直到合并到只剩两个元素，最终形成一个二叉树，回溯得到编码。

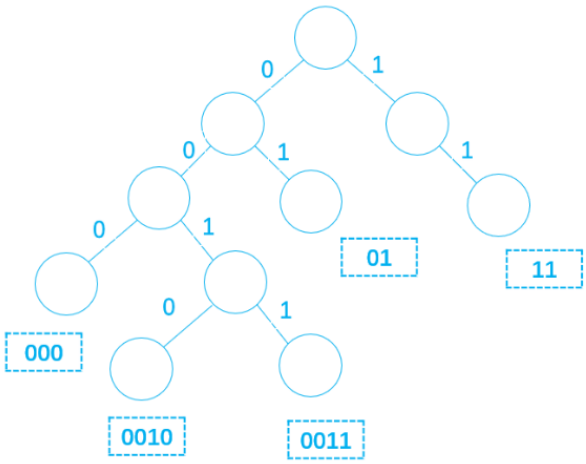


图 10 哈夫曼生成树图

文本采用霍夫曼编码的核心代码如下。

```
msgStr = fileread(pathfile);
native = unicode2native(msgStr);        % 转成本地编码

len = length(native);
b = mod(len, 256);
a = (len - b)/256;
msgBin = double([a, b, native].');
```

```
34.
35. % 霍夫曼编码
36. text_code = myhuffmanenco(msgBin);
```

在 matlab 进行如果使用 m 代码编写霍夫曼编码，其相关操作是元素级计算（在 matlab 中使用 for 循环效率低下），运行较慢，速度比不上直接编码。然而，一个好消息是，在 matlab 中我们可以使用 huffmandict 和 huffmanenco 这两个函数来提高霍夫曼编码的效率，它们是 matlab 中用 C 等其他语言写成的内建函数（built-in functions），效率很高。

使用 huffmandict 和 huffmanenco 编写的霍夫曼编码函数如下。首先统计各个元素出现的概率，接着使用 huffmandict()函数根据元素的出现频率生成编码字典(元素-编码对)，使用 huffmanenco()函数根据编码字典对输入序列进行编码，最后我们将生成的哈夫曼树（元素-编码对）加入到序列中（每个对包括 10bit 元素序列、10bit 编码序列长度和编码序列）。

```
37. function code = myhuffmanenco(data)%输入向量，返回信源编码的最终结果
38. % 统计各个元素出现的概率
39. max_n = max(data);
40. min_n = min(data);
41. p = zeros(max_n-min_n+1, 1);
42. for i = min_n:max_n
43.     p(i-min_n+1) = length(find(data == i));
44. end
45. p = p/sum(p);
46.
47. % 生成编码字典(元素-编码对)
48. a = min_n:max_n;
49. dict = huffmandict(a, p);
50. % 编码
51. encode = huffmanenco(data, dict).';
52.
53. % 考虑怎么把 dict 元胞数组加入到序列中,8bit,8bit,?, （a,对应矩阵长度，矩阵）
54. code = [encodedict(dict), encode];%最终输入到下一层的结果
55. code = code';% 转为列向量
56. end
```

值得注意的是，相比直接编码，使用霍夫曼编码并不一定能减少传输序列的长度，这是因为尽管我们能够使用哈夫曼编码来生成理论上平均码长（每个码元的平均比特数）最短的序列，但是我们不能忽略了还需要传递给接收方的哈夫曼树所占据的比特数。

(2) 图像

对于图像，可以采取直接编码的方式。

通过 imread() 函数读入图片并保存至类型为 uint8 的矩阵 img 中；将 img 的三通道信息分别保存至 R、G、B 三个矩阵，uint8 类型的数据范围为 0-255，则每个数据可以用 1 Byte（即 8 位二进制数）表示。最后，按顺序将矩阵大小、R、G、B 矩阵转化为一个十进制的行向量 info，使用 de2bi() 函数将十进制向量 info 的数据转化为二进制信息序列。

图像采取直接编码的核心代码如下。

```
57. I = imread(pathfile);
58. I = imresize(I, img_size);  % resize 图片大小
59.
60. if size(I, 3) == 3
61.     I1 = I(:,:,1);
62.     I2 = I(:,:,2);
63.     I3 = I(:,:,3);
64.     % I 的大小和三通道信息
```

```
    info = [size(I), reshape(I1, 1, []), reshape(I2, 1, []),
reshape(I3, 1, [])];
else
    % I 的大小和灰度信息
    info = [size(I), 1, reshape(I, 1, [])];
end

info = de2bi(info, 8).';     % 转成 8bit
img_code = double(info(:));  % 转为 double 型列向量
```

在上述直接编码的过程中，我们直接传输了所有像素点的信息，传输的数据量为 L1 = w*h*3*8bit，其中 w 和 h 分别是图像的宽和高。我们观察到，对于一个图像而言，它每个像素点的颜色的可能性是有限的(256^3)，那么一个直接的结论是，当我们传输一个非常大的图像时，比起直接传送其所有像素点的信息，我们可以选择传送这个图像的颜色集合（比如图像只有黑白两色，那么集合就只有两个点(0,0,0)和(255,255,255)）以及各个像素点的颜色索引（根据索引在集合中提取颜色）。以仅包含两种颜色的极端情况为例，这时传输的数据量为 L2 = w*h*1bit+2*3*8bit = (w*h+48)bit，当图像较大时，L2 约为 L1 的 1/24，压缩比非常可观。

然而，上述情况过于理想化了，实际上一个较大的图像，其颜色往往可以达到数万种，这种情况下压缩的效果不太明显。而极端情况下，颜色的种类可以靠近 256^3，使得传输的数据量比起 L1 更大。我们进一步想到，能否通过适当的减少颜色的种类来进行图像压缩，即使用更少的颜色来表示图像，这显然会给我们的信息带来损失，但我们期望的是，能否在我们可以接收的失真范围内，用有限的颜色，尽可能的还原原图。将一个图像的像素点的三通道信息绘制成点云图，如下图所示。点云图的每个点代表图像的每个像素点的颜色，可以看出，点云的分布呈现集中的趋势。

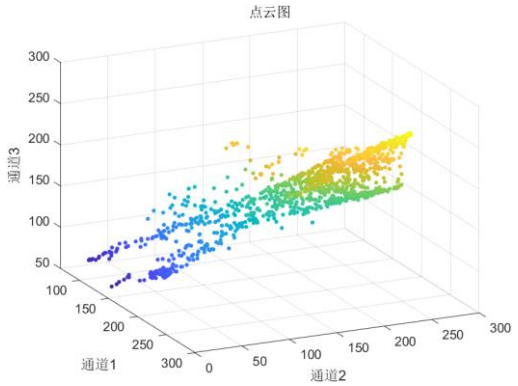


图 11 某个图像的三通道点云图

除去重叠的点外，我们依然可以得到大量的不同的点，而要以有限的颜色来表示图像，意味着我们必须将点云分割成不同的集群，并从各个集群中提取出一个特征点来代替集群，即我们需要对点云进行聚类。这里我们采用一种属于无监督学习范畴的算法——K-means，来实现聚类。K-means 是一种应用很广泛的聚类算法，它简单地将所有相似的数据点聚类在一起，然后用平均值替换它们的值。下面介绍其具体实现方法。

假设我们需要将点云分割成 k 个集群，那么我们首先从点云中选择 k 个点作为集群的初始点，此时集群的均值就是每个点的值。集群初始化的代码如下。

```
function initial_means = get_initial_means(array, k)
rowrank = randperm(size(array, 1));        % size 获得 a 的行数，randperm
打乱各行的顺序
initial_means = array(rowrank(1:k), :);    % 按照 rowrank 重新排列各行，选
择 k 个点
end
```

接下来，我们根据 k 个集群的均值对整个点云图进行分类，每个点都将分到一个集群中，之后我们将根据集群中的点重新计算集群的均值 new_means。这里分类的原则是，点选择距离其最近的集群，反映在点云图上就是点与集群代表点之间的欧氏距离。我们将上述过程称为 K-means 算法的单个更新/步骤，其代码如下。

```
function [new_means, clusters] = k_means_step(X, k, means)
[m, n] = size(X);

% 计算行向量的二范数(X 中每个行向量与 mean 的距离)
eucDist = zeros(m, k);
for i = 1:k
    eucDist(:, i) = sqrt(sum((X - means(i, :)).^2, 2));
end

% 进行聚类，X 中每个行向量与 mean 的最短距离的索引(1 - k)，即每个行向量分到的类
[~, clusters] = min(eucDist.') ;

% 计算每个类的新均值
new_means = zeros(k, n);
for i = 1:k
    points = X(clusters == i, :);
    if ~isempty(points)
        new_means(i, :) = mean(points);
    else
        new_means(i, :) = ones(1, n) * 0.5;
    end
end
end
```

现在我们已经完成了 K-means 步骤的实现，让我们尝试可视化如果多次重复这些步骤会发生什么。这里我们使用提前准备好的已经确定分类的 2D 数据，取 k=3，迭代 20 次，下面的图中给出了经过 1-4 次 K-means 步骤后的集群分布。

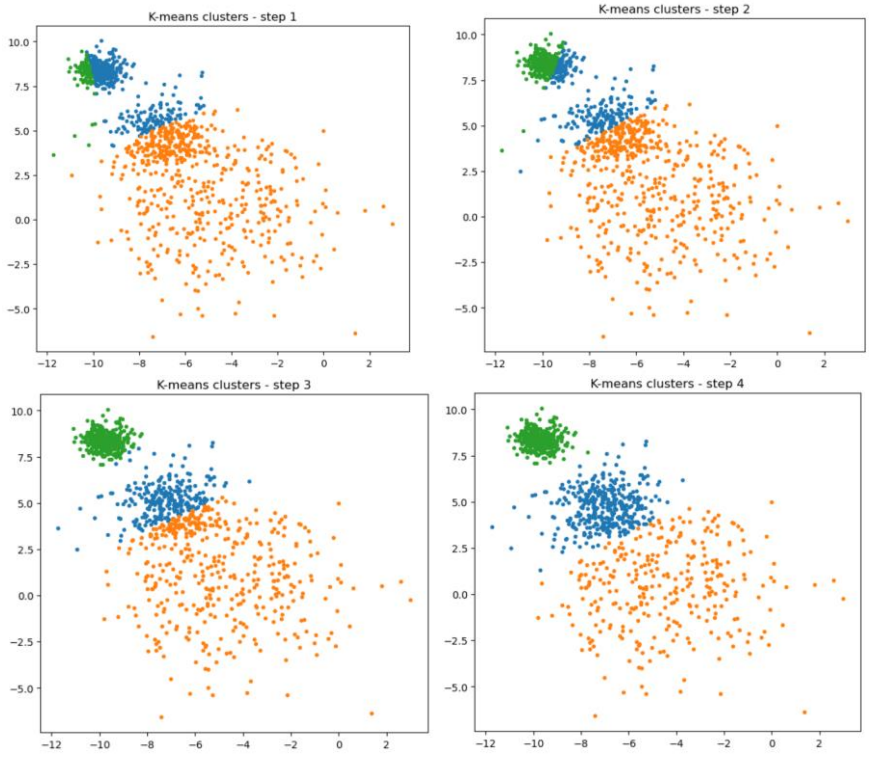


图 12 K-means 步骤实现的可视化图

经过多次运行 K-means 步骤后，我们发现集群的分布趋于稳定，它与真实分类的对比如下图所示。

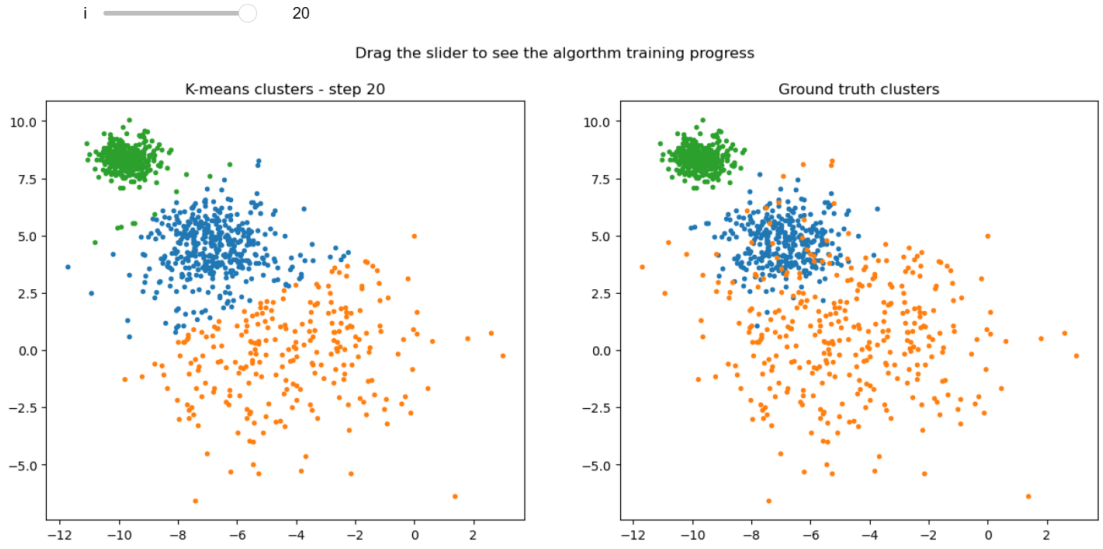


图 13 K-means 聚类结果与真实分类的对比图

2D 数据聚类看起来很不错，现在是时候使用 K-means 进行图像压缩了。我们将使用 K-means 算法将图像的 RGB 值分离到 k 个集群中，然后将图像的原始值替换为相应的集群中心值，重构图像以观察效果。相关代码如下。其中 image_values 是图像矩阵，经过 uint8 转为 double 型和归一化的操作得到，k 为集群数，max_step 为最大迭代次数。

```
function updated_image_values = k_means_segment(image_values, k,
max_step)
[r, c, ch] = size(image_values);

% 创建一个 k_means_step 可以处理的矩阵
dataPoints = reshape(image_values, r * c, ch);
% 初始化聚类均值
startMeans = get_initial_means(dataPoints, k);

currClusters = zeros(1, r * c);  % 集群
% 循环直到收敛
for i = 1:max_step
    prevClusters = currClusters;
    [updatedMeans, currClusters] = k_means_step(dataPoints, k,
startMeans);
    startMeans = updatedMeans;
    if all(currClusters == prevClusters) % 收敛时结束循环
        break
    end
end

% 将原始值替换为相应的集群值并重构图像
updated_image_values = updatedMeans(currClusters, :);
updated_image_values = reshape(updated_image_values, r, c, ch);
end
```

不断循环 K-means 单步直到收敛（集群分布不再更新）。为了避免不收敛的情况下一直运行程序，设定最大迭代次数为 256。分布取 k=4, 16, 64, 128, 256 进行测试，如下图所示。可以看到，集群数为 8 时，原图的轮廓已经能够还原出来；集群数为 16 时，已经能够还原出原图的色域分布以及线条层次感；集群数为 64 时，能够还原出原图的更多细节，以及一些比较小范围的颜色；继续增大集群数，图像颜色的饱和度和区分度更加明显，但提升的效果有限，且花费的时间大大增加。

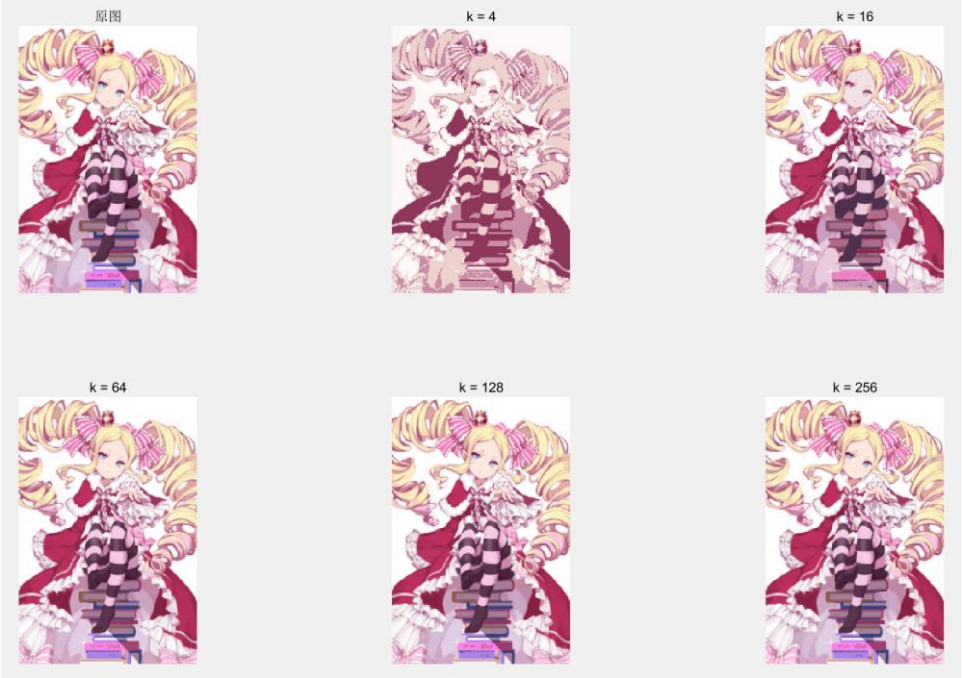


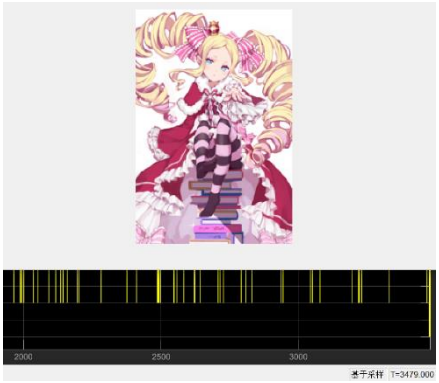
图 14 K-means 聚类压缩图像与原图对比图

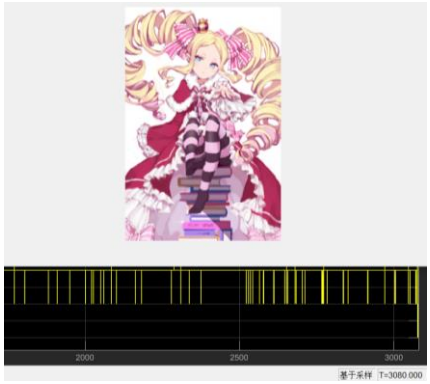
选择 k=256 对图像进行压缩，采用直接编码的情况下，传输的数据量为 L3 = w*h*8bit+256*3*8bit = (w*h+768)*8bit，对于一个 240*160 的图像，压缩比可以达到 (w*h+768)/ (w*h*3) = 0.34，图像越大，压缩比越接近 1/3。值得注意的是，集群索引的范围为 1-256，传输时应当-1 以使其范围为 0-255。然而，灰度图不能采用同样 k 值的压缩，原因在于灰度图传输的数据量为 L4 = w*h*8bit，如果采取同样的 k 值，压缩后灰度图传输的数据量为 L5 = w*h*8bit+256*8bit，L5 大于 L4。或者，我们应当更直观地指出，灰度图的“颜色”仅有 256 中可能，所以当 k=256 时我们并没有进行压缩。如果要使用 K-means 算法压缩灰度图，我们应当取更小的 k，这样才有可能传输更少的数据。例如，取 k=128，压缩后灰度图传输的数据量为 L6 = w*h*7bit+128*8bit，当图像足够大时，L6 小于 L4。

最终我们决定对于三通道图像，取 k=256，对于灰度图，取 k=16。

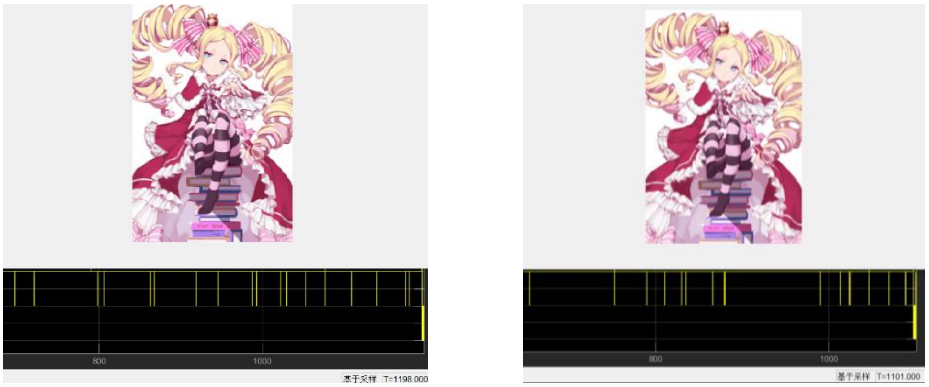
对于经过压缩得到的图像信息序列，我们采用霍夫曼编码。这一点也不突兀，事实上在文字信源编码时，霍夫曼编码的对象是 ASCII 码序列，范围是 0-255；而图像信息序列也是一个正整数序列，同样处于 0-255 的范围内（取 k=256 亦有此意）。理想情况下，图像信息序列中存在部分重复率比较高的数据，这时即使考虑到需要传输的哈夫曼树，信息序列的长度依旧短于直接编码。

接下来，我们进行测试来说明 K-means 算法压缩图像和采用霍夫曼编码的有效性。我们设定测试图像的大小是 240*160*3，在信源编码中分别采取不压缩直接编码、不压缩哈夫曼编码、压缩直接编码和压缩哈夫曼编码四种形式，信道编码选用线性分组码，调整方式为 16QAM，通过观察仿真时长来直观地评判四种形式生成的编码序列的长短。





(a) 不压缩直接编码　　(b) 不压缩哈夫曼编码



(c) 压缩直接编码　　(d) 压缩哈夫曼编码

图 15 传输图像信息的仿真时长对比图

可以看到，在最后还原的图像几乎完全相同的情况下，采用不压缩直接编码的情况仿真时长最长(3479s)，单独采用压缩(1198s)或者哈夫曼编码(3080s)仿真时长都会缩短，采用压缩哈夫曼编码的仿真时长最短(1101s)。这说明采用 K-means 算法压缩图像和采用霍夫曼编码能够有效地缩短信息序列的长度。

采用 K-means 算法压缩图像并进行霍夫曼编码的核心代码如下。

```
I = imread(pathfile);
I = imresize(I, img_size);  % resize 图片大小

if size(I, 3) == 3
    % 使用 k-means 聚类进行压缩
    k = 256;                % 集群数
    max_step = 256;         % 最大迭代次数
    [updatedMeans, currClusters] = k_means_segment(double(I)/255, k, max_step);
    updatedMeans = double(uint8(255*updatedMeans));
    % I 的大小和集群信息和索引
    info = [size(I).'; reshape(updatedMeans.', [], 1); (currClusters-1).'];
else
    % 使用 k-means 聚类进行压缩
    k = 16;                 % 集群数
    max_step = 256;         % 最大迭代次数
    [updatedMeans, currClusters] = k_means_segment(double(I)/255, k, max_step);
    updatedMeans = double(uint8(255*updatedMeans));
    % I 的大小和集群信息和索引
    info = [size(I).'; 1; updatedMeans; (currClusters-1).'];
end

% 霍夫曼编码
img_code=myhuffmanenco(info);
```

经过上面的讨论，我们可以看到，K-means 算法看起来已经取得了较好的效果，还原的图像和原图看起来几乎没有差别，然而它的缺点也较为明显，对于小图，它可以迅速收敛，训练时间较短，然而，对于大图，其时间成本大大增加。此外，K-means 算法似乎过于简单粗糙了，其本质不过是使用平均值对原始值进行替换。

既然已经动用了机器学习这个工具，我们不妨继续深入下去，实现一个完整的高斯混合模型。接下来，我们试着引入高斯分布来实现一个多元高斯期望最大化(Multivariate Gaussian Expectation Maximization)算法。EM 算法是一种通用算法，它允许人们迭代计

算统计分布的最大似然估计量。在我们的方案中，我们关注的是高斯混合模型的最大似然估计量(MLEs)。

上述算法听起来似乎有点复杂，但我们依然能通过拆分步骤来实现它。以下函数就构成了上述算法的所有要素（由于代码过长，这里不再进行展示），其中最为重要的就是 E_step 和 M_step，它们将在 train_model 中被反复调用进行训练。最后一个函数之所以要反复训练，是因为我们初始化时 k 个点的选取完全是随机的，实际上算法对这初始化的 k 个中心是非常敏感的。

```
initialize_parameters(X, k) % 初始化 k 个点，返回均值、协方差和初始混合系数
prob(X, mu, sigma)          % 根据均值和协方差给出了一个概率密度估计
E_step(X,MU,SIGMA,PI,k) % 单步 1，计算每个集群的概率密度估计（责任矩阵）
M_step(X, r, k)         % 单步 2，根据责任矩阵重新计算均值、协方差和混合系数
loglikelihood(X, PI, MU, SIGMA, k) % 计算训练模型的对数似然
train_model(X, k) % 训练混合模型，迭代 E_step 和 M_step 直到收敛(似然值判定)
cluster(r)        % 根据责任矩阵得到使得似然值最大的集群索引(分类)
segment(X, MU, k, r) % 每个数据点被替换为集群值(最大似然分量均值)
best_segment(X,k,iters) % 通过反复训练模型并计算其似然，确定图像的最佳分割
```

在上述方案中，我们选择模型的唯一标准是它是否使后验似然最大化，而不管它需要多少参数。因此，“最佳”模型可能只是具有最多参数的模型，这将对训练数据过度拟合。为了避免过拟合，我们可以使用贝叶斯信息准则(BIC)，它会根据模型使用的参数数量来惩罚模型。这里惩罚包括复杂度惩罚和精度惩罚，显然，复杂度惩罚与模型参数（最主要是集群数）有关，精度惩罚与似然值有关。

通过以上方案，我们有信心对于给定的每一张图片都获得一个集群数合适的“最佳”模型，并确定图像的最佳分割。

(3) 音频

对于音频信息，可以采用脉冲编码调制（PCM）。

PCM 包括采样、量化、编码三个步骤，其中量化是对抽样值的取值离散，根据量化间隔的不同选取分为均匀量化和非均匀量化。

我们知道，把 16bit 的音频数据转为 8bit，最简单的方式是均匀量化（>>8，右移 8 位），但这样做会使得声音的噪音变大。非均匀量化可以有效地改善信号的量化信噪比，因此最好的做法是使用非均匀量化。其原理是对于小音量的声音，其蕴含的信息量更大，人耳对小音量更敏感；而大音量部分则影响没那么大。因此使用非均匀量化的方式，对于小音量部分保留更多的数据，大音量部分则保留更少的数据。

语音信号的量化常采用 ITU 建议的两种对数形式的非均匀量化压缩特性：A 律和 μ 律。这里我们采用 A 律（北美和中国地区通用）进行非均匀量化压缩，它将 13bit 压缩为 8bit。

使用 A 律进行量化编码的代码如下。

```
function z = PCM_A(x)
% 归一化
maxz = max(x);
z = x/maxz;
% 量化
A = 87.6;
y = quantificat(z, A);
% 编码
z = a_pcm(y);
% 串行数据
z = z.';
z = z(:);
```

```
68. end
```

A 律进行量化的函数如下。其中，X 表示输入信号，Y 表示输出信号，且均归一化到（-1，+1）区间。

$$y=\begin{cases}\dfrac{Ax}{1+\ln(A)}, 0\le|x|\le\dfrac{1}{A}\\ \dfrac{1+\ln(Ax)}{1+\ln(A)}, \dfrac{1}{A}\le|x|\le 1\end{cases}$$

使用该函数进行量化的代码如下。

```
69.       function y = quantificat(x, A)
70.       a = 1/A;                % 0.0114
71.       p = 1 + log(A);
72.       abs_x = abs(x);        % 输入信号的绝对值
73.       flag = x./(abs_x+eps); % 量化信号的符号
74.       % 对数映射
75.       y = ((abs_x < a) .* abs_x*A +  (abs_x >= a) .*
(1+log(abs_x*A))) .* flag/p;
76.       end
```

A=87.6 时，函数图像接近原点，函数图像如下。将 x 轴(0,1)非均匀量化为为 8 段 1/2、1/4、…、1/64、1/128 最终到原点 0，共 8 段；同理每段对应到 Y 轴上也有 8 段。

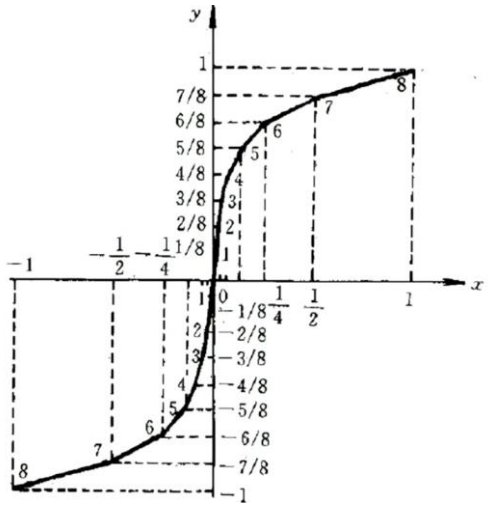


图 16 量化函数图

在以上基础上，我们将每一个(如 1/2-1/4)大段称为“段落”，在每个段落内部实现均分 16 分，这样在 X 轴上(-1,+1)就有 16 个段落 * 16 个均分 = 256 个量级。显然，不同段落的量化间隔并不相同，越靠近原点的段落，其量化间隔越小，即对小音量的量化更加精细，这与前面提到的小音量蕴含的信息更丰富相一致。

将归一化输入 x 乘上 2048，即输入的原量化电平需要 13bit 来表示（有 1bit 用于表示正负）。根据前面讨论的分段和量级，我们可以采用 3bit 来表示分段信息，4bit 来表示段内量化电平，1bit 来表示正负，这样就实现了从 13bit 到 8bit 的压缩。输入的音量大小与段落码的对应关系及段内码的权重如下图所示。

表 1 量化函数表

量化范围	归一化	段落码(3bit)	权重值
0~16	0~1/128	0 0 0	8 4 2 1
16~32	1/128~1/64	0 0 1	8 4 2 1
32-64	1/64~1/32	0 1 0	16 8 4 2
64~128	1/32~1/16	0 1 1	32 16 8 4
128~256	1/16~1/8	1 0 0	64 32 16 8
256~512	1/8~1/4	1 0 1	128 64 32 16
512~1024	1/4~1/2	1 1 0	256 128 64 32
1024~2048	1/2~1	1 1 1	512 256 128 64

以 2000 为例确定输出编码。首先是确定段落码，2000 在 1024~2048 段，所以段落码是 1 1 1；接着确定段内码，2000 - 1024 = 976，1024~2048 段内权重值为 512 256 128 64，也就是 512x + 256y+128z + 64w 趋近于 976，最终 x=1，y=1，z=1，w=1；极性取反 2000 的符号位是 0，所以极性为 1。最终 2000 的编码值为 1 1 1 1 1 1 1 1 = 0xff。

A 律编码的核心代码如下。

```
77.       function z = a_pcm(y)
78.       % 极性+段落码
79.       for i=1:len
80.           I=y(i)*2048;     % 转换为量化单位
81.           if I>0
82.               z(i,1) = 1;   % 极性码-正-1
83.           else
84.               z(i,1) = 0;   % 极性码-负-0
85.               I=-I;
86.           end
87.           % 段落码
88.           if I > 1024
89.               z(i,2) = 1;z(i,3) = 1;z(i,4) = 1;
90.           elseif I > 512
91.              ......
92.           end
93.           % 段内码
94.           idx=z(i,2)*4+z(i,3)*2+z(i,4)+1;  % 段落位置序号
95.           slevel = stalevel(idx);          % 该段落起始电平
96.           deta = detas(idx);               % 该段落的量化间隔
97.           level = slevel + (0:15) * deta;
98.           if I > level(16)
99.               z(i,5) = 1;z(i,6) = 1;z(i,7) = 1;z(i,8) = 1;
100.          elseif I > level(15)
101.              ......
102.          end
103.      end
104.      end
```

以上讨论中，我们实现了 A 律对音频信号进行量化和编码。关键的音频信号我们将通过录制的方式进行获取，并在编码信号前加上两个字节用于记录采样频率和录制时间，以便能够在接收端恢复出音频。

音频的获取和编码代码如下。

```
105.      function wave_code = get_voice(Fs, T)
106.      myVoice = audiorecorder(Fs, 8, 2, 0);   % 创建录制对象
```

```
107.      recordblocking(myVoice, T);              % 录制 Ts
108.      y = getaudiodata(myVoice);               % 获取声道数据
109.
110.      sfd = de2bi([Fs/1e3; T], 8).';           % 采样频率和录制时长转成
8bit
111.      wave_code = PCM_A(y(:, 1));              % 选择左声道编码
112.      wave_code = [sfd(:); wave_code];
113.      end
```

1.4.2 信道编码

假设发送序列长度为 n，其中信息码元数为 k，监督码元数为 n-k，则编码效率（码率）为 k / n，冗余度为 (n-k) / k；

信道编码分为分组编码和卷积编码。两者都是将原始信息流按长度 k 分组，再按照各自的编码规则映射成长度为 n 的码组进行发送。不同的是，对于每个码组的监督码元(长度为 n-k)，分组编码仅与本码组的 k 个信息码元有关，而卷积编码则同时与前 N-1 个码组及本码组的（共 N*k 个）信息码元有关（其中 N 被称为编码约束度）。

于是，根据上述定义的变量，对于上述两类编码有命名：(n，k)分组码；(n，k，N)卷积码；

在信道编码模块，我们将采用采用线性分组码和卷积码两种形式，用不同的 type_info 标记，以便信道译码时确定译码方式。

信道编码的代码如下。

```
114.      function chcode = Channelcode(x, type)
115.      G = [1,0,1;1,1,1];   % 卷积码生成多项式
116.
117.      % 信道编码
118.      if type == 1
119.          type_info = [0,0,1,1,1,1,0,0].';
120.          chcode = hamming_code(x); % 线性分组码
121.      elseif type == 2
122.          type_info = [1,1,0,0,0,0,1,1].';
123.          chcode = conv_code(x, G); % 卷积码
124.      end
125.      chcode = [type_info;chcode];
126.      end
```

(1) 线性分组码

M=(m_{k-1}， m_{k-2}，…， m_1， m_0)为单个码组中的信息码元；

C=(c_{n-1}， c_{n-2}，…， c_1， c_0)为分组编码后的分组码；

线性分组码的编码过程为 k 个信息码的线性组合：

$$C = MG = [m_{k-1}, m_{k-2}, \ldots, m_0] * [g_{k-1}, g_{k-2}, \ldots, g_0]^T$$

其中，G 是生成矩阵，由 k 个线性无关的行向量（基底）组成，注意到每一个基底都包含 n 个元素，它的系统形式（一种标准形式）为：

$$G = [I_k \quad Q]$$

则编码过程进一步写为：

$$C = [M \quad MQ]$$

其中 M 是信息码元，MQ 是监督码元。至此，我们完成了将 k 维空间的信息码元映射到 n 维空间的编码过程。这多出来的 n-k 维元素将帮助我们在接收端进行纠错。

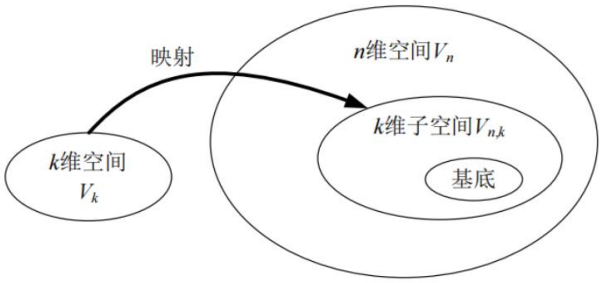


图 17 码空间与映射

线性分组码的编码代码如下。

```
function A = hamming_code(x)
%分组码各部分长度
k = 8;   % 信息码元数
% r = 4; % 监督码元数
% n = 12;  % 发送码元数

% IR=eye(r);
IK=eye(k);
P=[1 1 1 1 0 0 0 0
   1 0 0 0 1 1 1 0
   0 1 1 0 1 1 0 1
   0 1 0 1 1 0 1 1];
Q=P';
% H=[P,IR]; %监督矩阵
G=[IK ,Q];  %生成矩阵

x = reshape(x, k, []).';
% len 行-n 列
A = x*G;
% 模 2 加
A = mod(A, 2);
% 串行数据
A = A.';
A = A(:);
end
```

这里我们采用(12, 8)线性分组码，将输入序列中每 8 位编码为 12 位，并通过给定的矩阵 Q 确定了生成矩阵。

(2) 卷积码

卷积码编码器由三种主要元件构成，包括(N-1) k 级移存器、n 个模二加法（异或）器和一个旋转开关。在每个时隙，旋转开关输出 n 位后回到原点，移存器的值集体向右挪 k 位（最左边移存器存入新输入的值）。其编码的原理如下图所示。

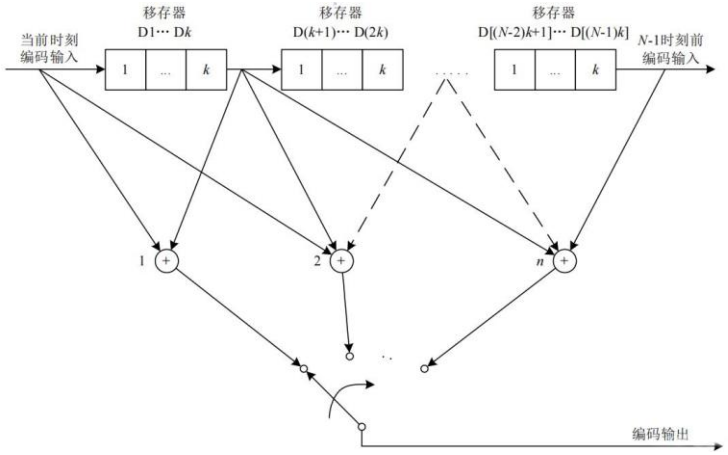


图 18 卷积编码的通用电路原理图

由于 k 只是决定了每一组寄存器的数量，为了降低复杂度，我们将采取(2,1,3)卷积编码。其编码原理如下图所示。

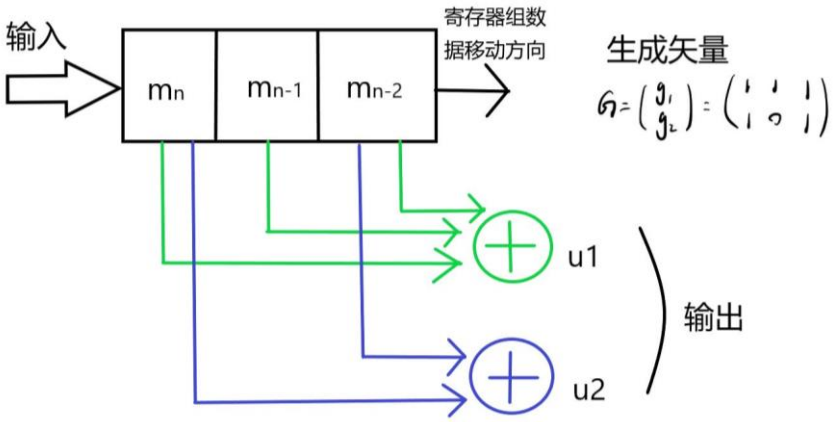


图 19 (2,1,3)卷积编码示意图

异或器的输入端取决于生成矢量（如上图右侧所示）。对于硬件电路来说，每个时钟周期到来时，从左侧高位输入一位数据，同时两个异或器输出两个数([u1,u2])。

对于软件逻辑来说，就是每次循环先计算输出，然后将数据[m_n，m_{n-1}]移到[m_{n-1}，m_{n-2}]，然后更新[m_n]。

新输入的数据在最左端。可以看出：n 为生成矢量的行数；k 为每组数据包含寄存器的个数；N 为生成矢量的列数。

于是对于每 k(取值为 1)个数据输入，有发送端输出：

$$U=\begin{bmatrix}u_1 & u_2\end{bmatrix}=\begin{bmatrix}m_{n-1},m_{n-2},\ldots,m_0\end{bmatrix}*G^T=M*G^T$$

根据上述公式，卷积码编码的代码如下。

```
152.      function C = conv_code(m, G)
153.      % 原始序列 m
154.      % 生成矢量 G
155.      len = length(m);
156.      k = 1;              % 表示每次对 k 个码元进行编码
157.      [n, N] = size(G);  % k 个输入码元拥有 n 个输出，N 表示每次监督的输入码
元数
158.
```

```
159.      % 生成序列 C
160.      C = zeros(n/k*len, 1);
161.      % 在头尾补 0，方便卷积输出和寄存器清洗
162.      m_add0 = [zeros(1,N-1), m.', zeros(1,N+1)];
163.
164.      % 循环每一位输入符号，获得输出矩阵
165.      C_reg = fliplr(m_add0(1,1:N));
166.      for i =1:len+N
167.          %生成每一位输入符号的 n 位输出
168.          C(n*i-(n-1):n*i) =  mod(C_reg*G.',2);
169.
170.          %更新寄存器序列+待输出符号（共 N 个符号）
171.          C_reg = circshift(C_reg, [0, 1]);
172.          C_reg(1) = m_add0(i+N);   % 添加新符号
173.      end
174.      end
```

1.4.3 交织

在移动通信这种变参信道中，比特差错经常成串发生(即突发错误），信道编码在检测和校正单个差错和不太长的差错串(随机错误）时有效，而交织技术就是把信道中的突发错误转变为随机错误，两者结合形成抗突发错误的有效形式。

交织的实质是将突发错误分散开来，而且交织深度（交织前相邻的符号在交织后的最小距离）越深，抗突发错误的能力越强。

这里我们采用较为简易的交织方式，将输入序列进行分组，组数为序列长度的开方值（序列不足则补 0），之后将每组中索引相同的数据重新分为一组，其代码如下。

```
175.      scramble_N = ceil(sqrt(length(chcode)));  % 交织矩阵深度
176.      scrambledcode = [chcode ; zeros(scramble_N^2-
length(chcode),1)];  % 补 0
177.      scrambledcode = reshape(scrambledcode,
[scramble_N,scramble_N]);   % 重新分组
178.      scrambledcode = reshape(scrambledcode.',
[scramble_N^2,1]);        % 串行数据
179.      chcode = scrambledcode;
```

1.4.4 数据成帧

采用以太网的帧格式，其包含：Preamble、SFD、帧长和 IFG。

图 20 以太网帧格式

图 21 以太网帧长部分包含字段

Preamble 是前导码，作用是在发送方和接收方之间进行时钟同步。当发送方发送数据时，加上前导码作为报文首部，发送给接收方；当接收方收到前导码时，会按照协议规定，调整自己的字节时钟，准备接收发送方来的数据。

SFD 是帧开始定界符，长度为 1 字节。这段码的意思是通知接收方，当收到 SFD 时，后边的内容不再是同步信号，而是真正的数据了。

IFG 是“帧间隙”，它表示两个 Frame 之间相隔大小。发送端发送完一个 Frame 之后，不会立即发送下一个，而是等待 IFG 时间之后才继续发送。

为了降低复杂度，我们仅发送一帧，帧组成如下：前导码（由 pre_core 重复 100 次构成）、帧开始定界符、帧长字段（3 字节）、帧数据。

组成数据帧的代码如下。

```
180.      pre = repmat(pre_core,100,1);   % 前导码
181.      len = scramble_N^2;             % 帧长
182.      c = mod(len, 65536);
183.      a = (len - c)/65536;
184.      len = c;
185.      b = mod(len, 256);
186.      c = (len - b)/256;
187.      len_type = de2bi([a;c;b], 8).'; % 帧长字段
188.      frame_code = [pre; sfd; len_type(:); chcode];
```

1.4.5 帧同步

帧同步的目的是从接收到的帧中提取出发送的数据序列。最直接的方法是找到由 pre_core 和 sfd 构成的 detframe 序列的位置，若为找到该序列，则认为未接收到帧。

检测到 detframe 序列后，我们将该序列后 3 个字节的数据先提取出来，计算帧中数据序列的长度，根据帧长计算交织矩阵宽度 scramble_N（这里利用宽度是正整数的性质，在一定程度上降低帧长字段出错的几率），由 scramble_N 修正帧长，最后提取出数据序列。

帧同步的代码如下。

```
189.      % 寻找 detframe 序列位置
190.      detframe=[pre_core; sfd];
191.      pos = 1:length(code);
192.      for i=1:length(detframe)
193.          pos_temp=find(code(i:end-length(detframe)+i)==detframe(i));
194.          pos=intersect(pos,pos_temp);
195.      end
196.      if isempty(pos)  % 未找到有效信息
197.          decode = [];
198.          return
199.      end
200.      pos = pos(1);
201.
202.      frame_syn_code = code(pos+length(detframe):end);
203.      len_type = reshape(frame_syn_code(1:24), 8, []).';         % 帧
长字段
204.      len_type = bi2de(len_type);
205.      len_type = len_type(1)*65536+len_type(2)*256+len_type(3);  % 计
算帧长
206.      scramble_N = floor(sqrt(len_type));  % 计算交织矩阵宽度
207.      frame_syn_code = frame_syn_code(25:scramble_N^2+24);       % 提
取数据序列
```

1.4.6 解交织

根据交织的方式，对帧同步提取出的数据序列进行解交织。

解交织的代码如下。

```
descrambledcode = reshape(frame_syn_code, [scramble_N
scramble_N]);
descrambledcode = reshape(descrambledcode.', [scramble_N^2 1]);
```

1.4.7 信道译码

首先根据数据序列第一个字节的数据确定编码方式，确定原则为码距最小，两种编码方式的 type_info 的码距为 8，至少要发生 4 位错误才会导致对编码方式的判断出错。之后根据对应的编码方式进行译码。

信道译码代码如下。

```
type_info = descrambledcode(1:8);
type = decode_type(type_info);
if type == 1
    decode = linear_decode(descrambledcode(9:end));
elseif type == 2
    decode = conv_decode(G, 1, descrambledcode(9:end));
end

% 确定编码方式
function type = decode_type(type_info)
linear_pss = length(find(type_info ~= [0,0,1,1,1,1,0,0].'));
conv_pss = length(find(type_info ~= [1,1,0,0,0,0,1,1].'));
% 根据码距确定编码方式
if linear_pss < conv_pss
    type = 1;
else
    type = 2;
end
end
```

(1) 线性分组码译码

对于经过(n, k)线性分组码编码的信息，我们对 n 个码元中出现的差错定义为错误图样 $E=[e_{n-1},e_{n-2},...,e_0]$，发生错误的地方为 1，其余为 0。则接收序列为 R = C + E 。

由于生成矩阵 G 是由 k 个线性无关的行向量组成，所以必然存在一个校验矩阵 H，使得 $G*H^T=0$，其中

$$H=[P \quad I_{n-k}]=[Q^T \quad I_{n-k}]$$

则解码过程为

$$R*H^T=C*H^T+E*H^T=E*H^T\begin{cases}=0\\ \neq 0\end{cases}$$

我们称上式为伴随式。显然，当伴随式为 0 时，我们可以认为没有出错（也可能发生多个错误），当伴随式不为 0 时，出现差错。由于 H 是已知的，所以根据接收结果我们可以得到伴随式，进而可以知道对应的错误图样是什么，这就完成了纠错。

当然了，这个纠错能力是有限的，定义最小码距为 d，则纠错能力的大小为(d-1)的一半向下取整。而我们采用的(12,8)线性分组码只能纠正一位错误，12 个错误图样加上 1 个无错图样对应 13 个伴随式（共计 16 个伴随式）。

根据伴随式 pe 及其对应的错误图样 SE，我们可以得到线性分组码译码的核心代码如下。

```
    % 生成伴随式
    S = mod(x*H.', 2);
    len = size(x, 1);
    result = zeros(len, n);
    for i=1:len
        flag = 1;
        % 根据伴随式确定错误图样并纠错
        for j=1:size(SE, 1)
            if all(S(i,:) == pe(j,:))
                result(i,:) = mod(x(i,:) + SE(j,:), 2);
                flag = 0;
                break;
            end
        end
        % 未找到对应伴随式，错码数大于 1，无法纠正错误
        if flag == 1
            result(i,:) = x(i,:);
        end
    end
    % 提取信息码元
    A = result(:,1:k);
```

(2) 卷积码译码

对于卷积码译码，我们采用 Viterbi 译码算法。

Viterbi 译码是一种最大似然译码算法，其关键在于我们要找出一种出现差错概率最小的译码结果，即我们乐观地认为，经过编码的序列在接收端会出现少量差错，而我们对接收序列进行一定的修正可以得到正确的编码序列。我们使用码距这一概念来度量接收序列与译码序列对应的编码序列的距离，距离越小，出错越少（当然，接收序列可能出现大量差错，以至于我们按照差错概率最小的原则得到完全错误的译码结果）。换言之，我们的目标是首先找到一个可能的编码序列，使得它与接收序列的码距最小。

这听起来似乎有些困难，但是别忘记了卷积码区别于线性分组码最重要的一点，卷积码是有“记忆”的编码，因此，讨论译码时我们必须重点关注的是寄存器组的状态，以及状态之间的转移。对于(n,k,N)卷积码，其寄存器组的状态是由后(N-1)个寄存器组决定的，第一个寄存器组保存输入序列。

进行译码前，我们需要做一些准备工作，了解在不同的输入下状态的转移及寄存器组的输出。为此，我们首先生成三个矩阵，输入矩阵 input、状态转移矩阵 nextstate 和输出矩阵 ouput。输入矩阵表示当输入为 input(i,j)时，由状态 i 转移到状态 j；状态转移矩阵表示当输入为 j 代表的序列时(当 k=1 时，nextstate 仅有两列，分别代表输入 0 和输入 1)，由状态 i 转移到状态 nextstate(i,j)；输出矩阵表示当输入为 j 代表的序列时，处在状态 i 下，寄存器组的输出为 ouput(i,j)。为了便于记录，我们将上述提到的输入和输出的二进制序列均采用用十进制数表示。

准备工作的核心代码如下。

```
    %%%% 对各个状态进行运算，得到输入矩阵、状态转移矩阵与输出矩阵 %%%%
    for s = 0:number_of_states-1
        % 对前一时刻状态到下一时刻状态之间的各条支路进行运算
        for t = 0:2^k-1
            % next_state_function 函数产生移存器跳转到的下一状态及当前时刻编码器内容
            [next_state,memory_contents] = next_state_function(s,t,N,k);
            % 内容为经由支路编号
            input(s+1,next_state+1) = t;          % 输入矩阵
```

```
                % 各条支路编码输出
                branch_output = rem(memory_contents*G',2);
                % 内容为下一时刻状态 s
                nextstate(s+1,t+1) = next_state;      % 状态转移矩阵
                % 内容为相应分支输出编码
                output(s+1,t+1) = bin2dec(branch_output); %输出矩阵
            end
        end
```

为了降低复杂性，我们依旧以(2,1,3)卷积码为例进行讨论，寄存器组共有 2^(1*(3-1))=4 个状态。

以接收序列 11 11 01 10 11 11 为例。定义寄存器组（后 N-1 个组）全 0 的状态为零状态，我们从零状态出发，逐步初次抵达所有的状态，并根据输出的编码序列计算其与接收序列的码距。如下图（网格图）所示。图示左边第一列 4 个数表示寄存器组的状态；框中的数表示到达该状态时的累积码距；右下角上面的数表示输入为 0 时的输出，下面的数表示输入为 1 时的输出；箭头表示状态转移，由箭头构成的从零状态到任何一个状态的路径称为状态转移路径。

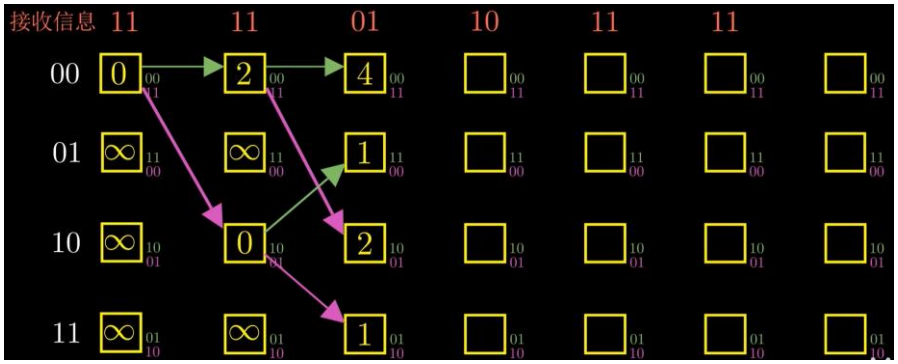


图 22 卷积码译码示意图 1

在这个过程中，网格图从全零状态出发，直到所有状态都有路径到达，我们保存初次到达每一个状态的路径和码距。

下一步，我们继续相同的操作，获得到达下一时刻状态的所有路径，可以发现这时路径的总数将是状态数的两倍，我们当然可以将它们都保存下来，直到最后才将各个路径的码距进行计算，然而这将耗费过多的内存和时间。我们很容易能发现从当前时刻某一状态出发的到达下一个时刻的最短（累计码距最小）路径必然来自于到达当前时刻该状态的最短路径，这说明我们只需要保存到达每个状态的最短路径及其累计码距即可。这一步的操作如下图所示，显然，从状态 0 到状态 0 的路径累计码距为 5，而从状态 1 到状态 0 的累计码距为 2，因此我们删去前一条路径，而保留了后一条路径，我们称保留的路径为幸存路径。

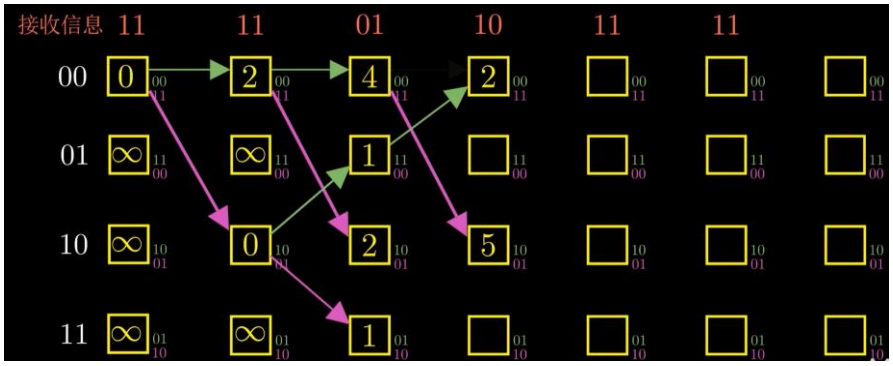


图 23 卷积码译码示意图 2

各个状态的路径度量（码距）更新的核心代码如下。其中我们用状态度量矩阵 state_metric 记录码距，其第一列为当前时刻各状态的路径度量，第二列为下一时刻各状态的路径度量（即更新后的状态度量）。此外，我们使用幸存状态矩阵 survivor_state 来保存幸存路径，以便最后能够回溯路径。幸存路径的存储方式为，一维坐标表示下一时刻状态，二维坐标表示该状态在网格图中的列位置，其值为当前时刻状态。

```
266.      for i = N:depth_of_trellis-(N-1)
267.          % 记录某一状态的路径度量是否更新过
268.          flag = zeros(1,number_of_states);
269.          for s = 0:number_of_states-1
270.              for t = 0:2^k-1
271.                  branch_metric = 0;
272.                  bin_output = dec2bin(output(s+1,t+1),n);
273.                  for j = 1:n
274.                     branch_metric = branch_metric +
metric_hard(decode_input_matrix(j,i),bin_output(j));
275.                  end
276.                  % 若某状态的路径度量未被更新
277.                  % 或一次更新后的路径度量大于本次更新的路径度量
278.                  % 则进行各状态路径度量值的更新
279.
if((state_metric(nextstate(s+1,t+1)+1,2)>state_metric(s+1,1)+branch_metric
) || flag(nextstate(s+1,t+1)+1) == 0)
280.                      state_metric(nextstate(s+1,t+1)+1,2) =
state_metric(s+1,1)+ branch_metric;
281.                      survivor_state(nextstate(s+1,t+1)+1,i+1) = s;
282.                      % 一次更新后 flag 置为 1
283.                      flag(nextstate(s+1,t+1)+1) = 1;
284.                  end
285.              end
286.          end
287.          state_metric = state_metric(:,2:-1:1);
288.      end
```

为了便于回溯，我们设定网格图上的各条路径最终需要回归到零状态，要实现这一点我们需要在编码时在输入序列结束后继续填充 0 直到寄存器到达零状态。填充 0 的过程中可以看到寄存器组可拥有的状态数不断减半（因为不断有寄存器组的状态已经确定为 0），因此这部分的译码只需要在上一步的代码中稍加修改，将 s 从全部状态可选削减为每过一个时刻，可选状态数减半（上一比特存留的状态数），最后只有零状态即可。

到达零状态后，我们根据幸存状态矩阵开始逐步向前回溯，就可以得到译码输出了，其核心代码如下。

```
289.      sequence = zeros(1,depth_of_trellis+1);
290.      for i = 1:depth_of_trellis
291.         sequence(1,depth_of_trellis+1-i) =
survivor_state(sequence(1,depth_of_trellis+2-i)+1,depth_of_trellis+2-i);
292.      end
293.      % 译码输出
294.      decode_output_matrix = zeros(k,depth_of_trellis-N);
295.      for i = 1:depth_of_trellis-N
296.          % 由输入矩阵得到经由支路编号
297.          dec_decode_output =
input(sequence(1,i)+1,sequence(1,i+1)+1);
298.          % 将支路编号转为二进制码元，即为相应的译码输出
299.          bin_decode_output = dec2bin(dec_decode_output,k);
```

```
            % 将每一分支的译码输出存入译码输出矩阵中
            decode_output_matrix(:,i) = bin_decode_output(k:-1:1)';
        end
        % 重新排列译码输出序列
        decode_output =
reshape(decode_output_matrix,1,k*(depth_of_trellis-N)).';
```

最终，我们得到了一条码距最小的幸存路径，并据此得到了编码前的序列，要注意的是，最后重新排列时需要去掉填充的 0。

1.4.8 信源译码

首先根据数据序列第一个字节的数据确定信源格式，确定原则为码距最小，三种信源的 type_info 的码距为 4，至少要发生 2 位错误才会导致对信源格式的判断出错。之后根据对应的信源进行译码。

信源译码代码如下。

```
        function [type, info] = Sourcedecode(x)
        p = mod(length(x), 8);
        if p ~= 0
            x = [x; zeros(8-p, 1)];
        end

        type_info = x(1:8);
        type = decode_type(type_info);

        if type == 1
            info = bits_to_str(x(9:end));
        elseif type == 2
            info = img_decode(x(9:end));
        elseif type == 3
            info = audio_decode(x(9:end));
        end
        end
        % 确定信源格式
        function type = decode_type(type_info)
        text_pss = length(find(type_info ~= [0,0,0,1,1,0,0,0].'));
        img_pss = length(find(type_info ~= [0,0,1,0,0,1,0,0].'));
        audio_pss = length(find(type_info ~= [0,1,0,0,0,0,1,0].'));
        pss = [text_pss; img_pss; audio_pss];
        % 根据码距确定信源格式
        [~, type] = min(pss);
        end
```

(1) 文本

文本信息译码的过程比较简单。对于直接编码，我们将信息序列每 8 位转为十进制数 info 序列；如果是霍夫曼编码，则将信息序列进行霍夫曼解码得到十进制 info 序列。根据 info 前两个点的信息计算得到文本字符数 len，将 len 个 ASCII 字符通过 native2unicode()函数得到原字符。

霍夫曼解码的核心代码如下。首先提取出编码字典（元素-编码对），接着使用 huffmandeco()函数根据编码字典和接收序列恢复出编码前序列。

```
        function data = myhuffmandeco(code)%返回解码结果
        code = code'; % 转为行向量
        [dict, k]=decodedict(code);
        code(1:k)=[]; % 删除字典占的位置
        data = huffmandeco(code,dict).';
```

```
336.        end
```

文本信息译码的代码如下。

```
337.    function msgStr = bits_to_str(msg_bits)
338.    % 霍夫曼解码
339.    native = myhuffmandeco(msg_bits);
340.
341.    len = native(1)*256 + native(2);
342.    msgStr = native2unicode(native(3:len+2)); % 转成 unicode
343.    end
```

(2) 图像

图像信息译码的过程稍微复杂。

对于经过直接编码的序列，我们将信息序列每 8 位转为十进制数，根据前三个字节的信息得到图像的高 h、宽 w 和通道数 channel，并创建一个大小为 channel*w*h 的 uint8 矩阵 I 用于保存图像信息，之后将 info 中各通道的信息写入 I 即可。

对于使用 K-means 聚类进行压缩和霍夫曼编码的序列，我们首先进行霍夫曼解码，接着根据图像的通道数确定集群数和集群平均值的维度，从序列中提取出集群平均值和图像的集群分布(w*h 个索引值)，最后将索引值替换为相应的集群值并重构图像。

图像信息译码的代码如下。

```
344.        function I = img_decode(x)
345.        % 霍夫曼译码
346.        info = myhuffmandeco(x);
347.
348.        h = info(1);
349.        w = info(2);
350.        channel = info(3);
351.
352.        % 提取集群值和索引序列
353.        info = [info;zeros(channel*w*h+3-length(info), 1)];
354.        if channel == 3
355.            updatedMeans = reshape(info(4:771), 3, 256).';
356.            currClusters = info(772:771+w*h) + 1;
357.        else
358.            updatedMeans = info(4:19);
359.            currClusters = info(20:19+w*h) + 1;
360.        end
361.
362.        % 将索引值替换为相应的集群值并重构图像
363.        updated_image_values = updatedMeans(currClusters, :);
364.        updated_image_values = reshape(updated_image_values, h, w,
    channel);
365.        I = uint8(updated_image_values);
366.        end
```

(3) 音频

音频信息的译码过程为，根据前两个字节的信息得到音频的采样频率和录制时长，据此计算音频的采样点数(Fs*T*8)，提取出音频信息序列。根据极性码、段落码和段内码恢复出量化电平，即将 8bit 还原为 13bit。最后通过量化函数的反函数恢复出量化前的归一化信号即可。

值得注意的是，这里返回的音频信息 audio 是一个结构体，它包含三个属性，分别用于表示采样频率、录制时长和音频采样信息。

音频信息译码的代码如下。

```
function audio = audio_decode(x)
x = reshape(x, 8 ,[]).';
audio.Fs = bi2de(x(1,:)) * 1e3;
audio.T = bi2de(x(2,:));
len = audio.Fs * audio.T;
x = x(3:len+2,:);

len = length(x);
z = zeros(1, len);
parcode = 1:8;              % 段落位置序号
stalevel = 2.^(parcode+2); % 起始电平
zhishu = parcode-2;
zhishu(1) = 0;
detas = 2.^zhishu;          % 量化间隔
for i=1:len
    idx = x(i,2)*4+x(i,3)*2+x(i,4)+1;    % 该段落位置序号
    slevel = stalevel(idx);              % 该段落起始电平
    deta = detas(idx);                   % 该段落量化间隔
    secode = x(i,5)*8+x(i,6)*4+x(i,7)*2+x(i,8);  % 段内位置序号
    I = slevel+secode*deta+deta/2;        % 译码电平
    z(i) = I/2048;                        % 实际电平
    if x(i,1)==0
        z(i) = -z(i);
    end
end

%还原量化前电平
audio.wave = quanreduction(z);
end

% 还原量化前信号
function x = quanreduction(y)
A = 87.6;
a = 1/(1+log(A));
p = 1+log(A);
abs_y = abs(y);        % 输入信号的绝对值
flag = y./(abs_y+eps); % 还原信号的符号
x = ((abs_y < a) .* abs_y*p +  (abs_y >= a) .* exp(abs_y*p-
1)) .* flag/A;
end
```

1.4.9 QDPSK 正交调制解调

QPSK 是英文 Quadrature Phase Shift Keying 的缩略语简称，意为正交相移键控，是一种数字调制方式。在数字信号的调制方式中 QPSK 四相移键控是目前最常用的一种卫星数字信号调制方式,它具有较高的频谱利用率、较强的抗干扰性、在电路上实现也较为简单。

所谓四相绝对移相调制，是利用载波的四种不同相位来表征四种数字信息。因此，对于输入的二进制数字序列先进行分组，将每两个信息数字编为一组，然后根据其组合情况用四种不同的载波相位序列承载信息。由于每一种载波相位代表两个比特信息，故每个码元(四进制码元)常被称为双比特码元，并把组成双比特码元的前一信息比特用 A 代表，后一信息比特用 B 代表。双比特码元中两个信息比特 AB 是按格雷码（即反射码)排列的，因

此，在接收端检测时，如果出现相邻相位判决错误，只造成一个比特的差错，有利于提高传输的可靠性。

而 DQPSK 是由 QPSK 经过差分编码后调制的四相调制。

QPSK 调制信号可以表示为 $S_i(t) = A\cos(\omega_c t + \theta_i), i \in [1,4]$

其中，$\theta_i, i \in [0,3]$ 为余弦载波相位，有 4 种取值，因此，通过 2 比特 I 路和 Q 路数据的组合，对载波相位进行调制。初始相位为 0 的 QPSK 信号的矢量图和初始相位为 $\frac{\pi}{4}$ 的 QPSK 信号的矢量图如图所示。

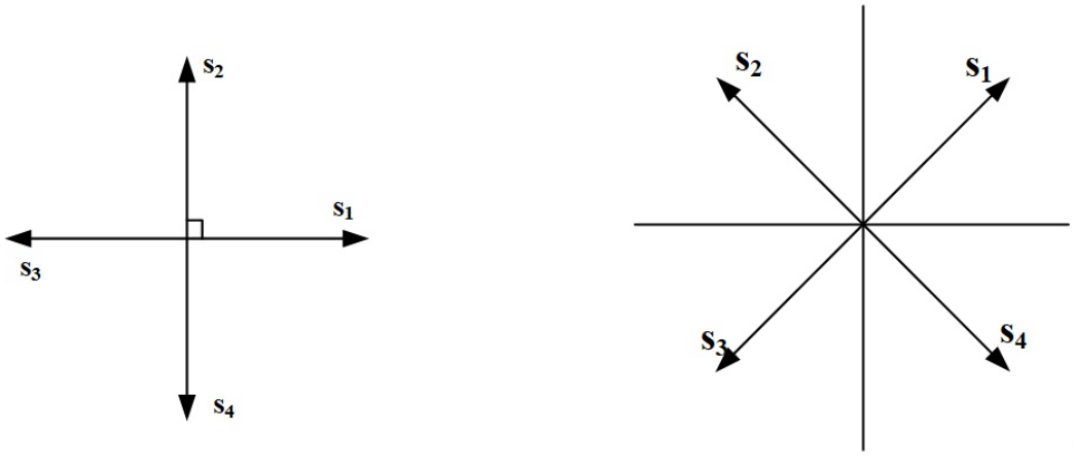


图 24 QPSK 信号的两种矢量图

QPSK 正交调制信号产生的原理框图如图所示

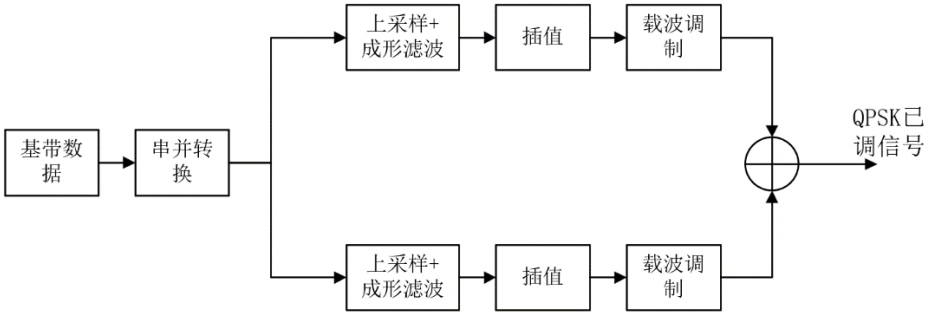


图 25 QPSK 正交调制信号产生的原理框图

QPSK 正交调制信号解调的原理框图如图所示

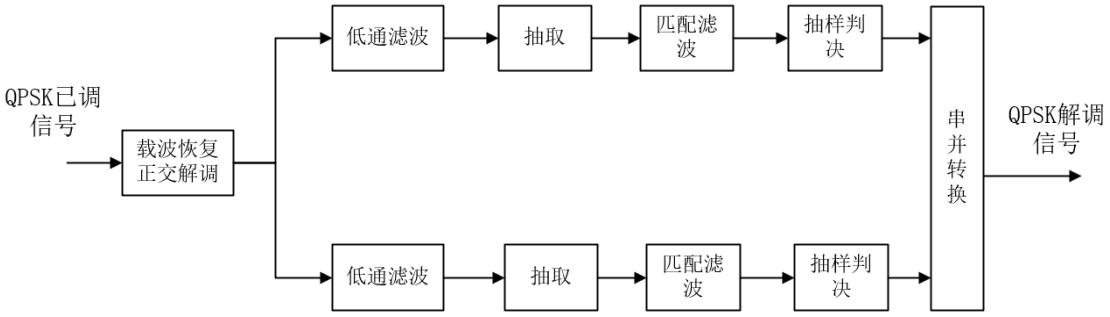


图 26 QPSK 正交解调信号产生的原理框图

(1) 发射机部分

发射机部分主要由以下模块组成

a. 串转并模块

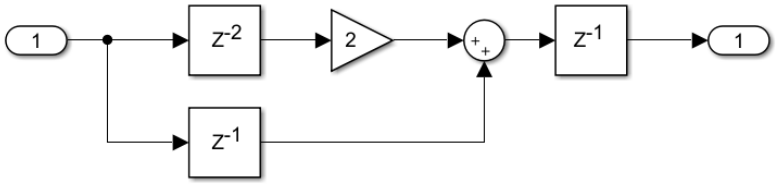


图 27 DQPSK 串转并 simulink 仿真图

QPSK 调制方式需要将一路串行基带数据转换为两路并行的 I 路和 Q 路数据，因此需要通过该模块将数据由串行转换为并行。

QPSK 调制方式需要将一路串行的二进制数据转换为两路并行的 I 路和 Q 路数据。其中，I 路和 Q 路各传输 1bit 数据。因此需要通过该模块将数据由串行数据转换为 2 并行数据。

b. 差分编码模块

通信中在使用 QPSK 调制时，为了避免收发端相干频率相位不一致，导致相位差为π的相位模糊，会在发送端对发送的比特数据进行差分编码，将绝对相位转为相对相位；同时在接收端对接收的符号进行解差分，将相对相位转换为绝对相位。

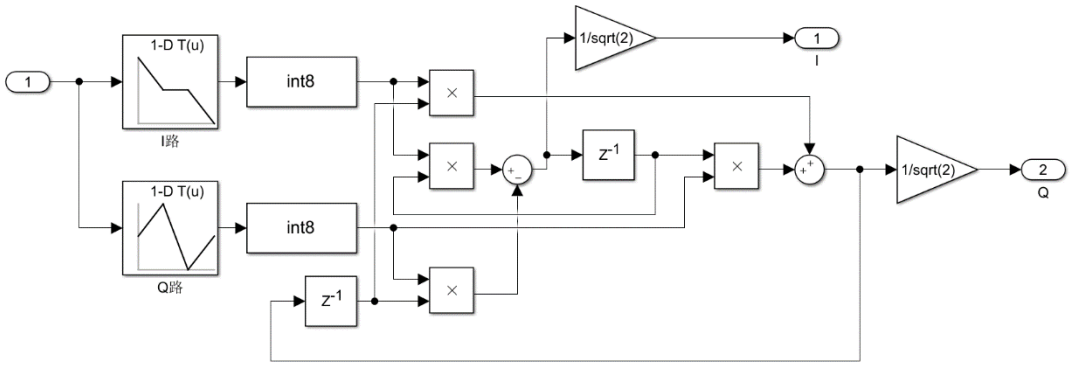


图 21 DQPSK 差分编码 simulink 仿真图

我们选用初始相位为$\pi/4$的 QPSK 调制方式。两个寄存器分别储存了上一次 I、Q 两路的调制结果，它们的初始相位是$\pi/4$。信号输入后，先通过前面两个查找表，实现了初始相位为 0 的 QPSK 调制。然后将调制结果与上一次 QPSK 的调制结果进行复数乘法，使它们相位相加，即可得到差分编码的输出。两次输出的相位之差，即是初始相位为 0 的 QPSK 调制结果。

c. 正交调制发射

正交调制发射部分包括上采样、成形滤波和正交上变频三个部分。

将基带 01 数据序列经过串转并处理和差分调制后的数据序列进行 10 倍上采样，然后通过根升余弦滤波器进行成型滤波，然后将分别将同相分量信号和正交分量信号分别与 cos 信号和 sin 信号相乘后相加（或者相减）形成 DPSK 调制信号。

为了减少码间串扰，使用了根升余弦滚降滤波器。设计符号的数字频率和载波的数字频率相等，同时设计载波的数字频率是-0.2π。所以将符号 10 倍上采样，通过滚将系数为 0.2，长度为 6 个符号，采样率为 10 倍的根升余弦滚降滤波器。得到成型滤波结果后，使用数字频率-0.2π的 NCO 序列进行正交上变频。

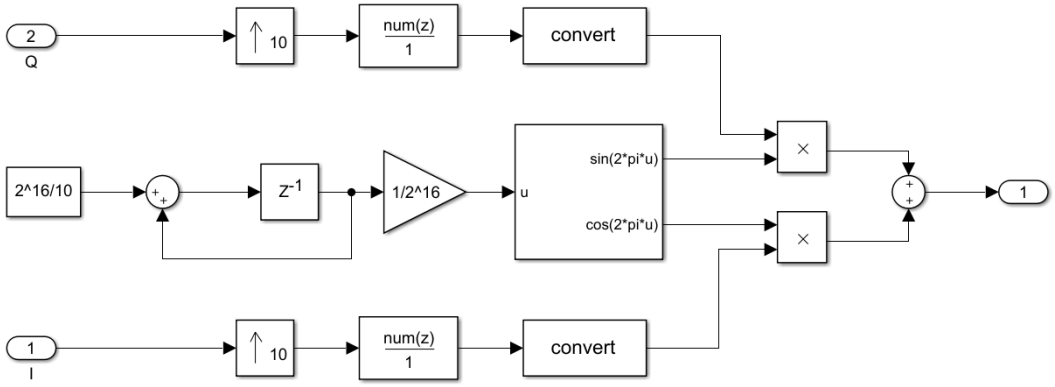
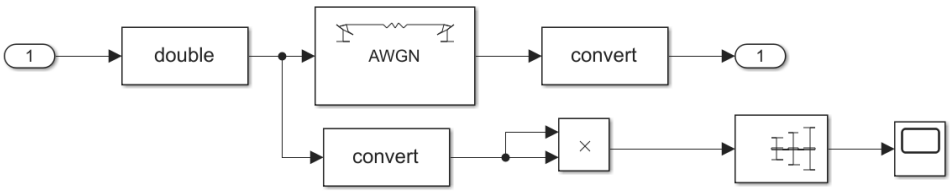


图 22 DQPSK 正交调制 simulink 仿真图

d. 信道



只考虑 AWGN 信道的影响。通过功率测量模块检测信号功率，测得传输随机信号时，信号功率约为 0.0504W。以此为基准，控制了 AWGN 信道的信噪比。

(2) 接收机部分

a. 载波同步

采用极性科斯塔斯环进行载波恢复

对于 BPSK 的相干解调，常常使用 Costas 环进行载波恢复。Costas 环的组成原理如下图所示，它是由输入信号分别乘以同相和正交两路载波信号而得名。输入信号分为上、下支路，分别乘以同相和正交载波，并通过低通滤波后再相乘,完成鉴相功能，最后经环路滤波器输出控制本地振荡器的误差电压，通过反馈环路可以获得相干解调波。Costas 环是一种数字通信系统中常用的解调和同步电路，它可以对特定的脉冲序列进行同步，并且具有抗干扰性能优良的特点。Costas 环由一个包含两个 NCO 的闭环系统构成，其中 NCO 的频

率输出受到相位误差信号的修正。在 Costas 环系统中，输入信号与 NCO 的输出信号进行正交下变频，然后通过相位检测器得到相位误差信号，并经过环路滤波器的处理，进而控制 NCO 的输出频率以消除相位误差。通过反复迭代这个过程，Costas 环最终可以实现对特定脉冲序列的解调和同步。

其基本原理框图如下所示。

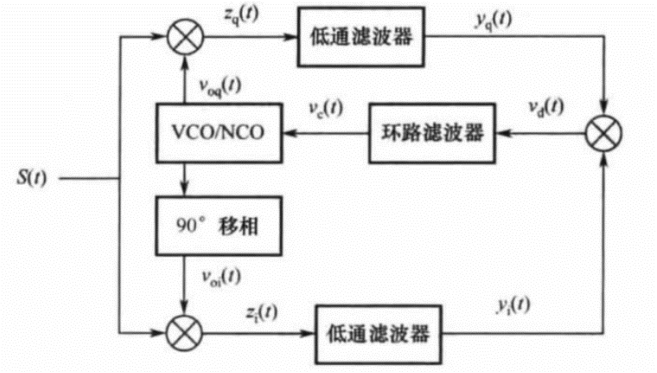


图 30 Costas 环示意图

对于 QPSK 的相干解调，需要修改反馈回路以满足解调要求，实际应用中常常使用极性 Costas 环进行相干解调，普通 costas 环的输出相位模糊是π，无法处理 I、Q 两路都有信号的情况。极性 costas 环拥有不同的相位检测结构，这使它的相位模糊精确到$\pi/2$。因而它可以对 QPSK 信号进行载波同步。

利用极性 costas 环的正交下变频结构，将混频后的滤波器设置为匹配滤波器，可以直接输出匹配滤波后的基带信号。

其基本原理框图如下所示。

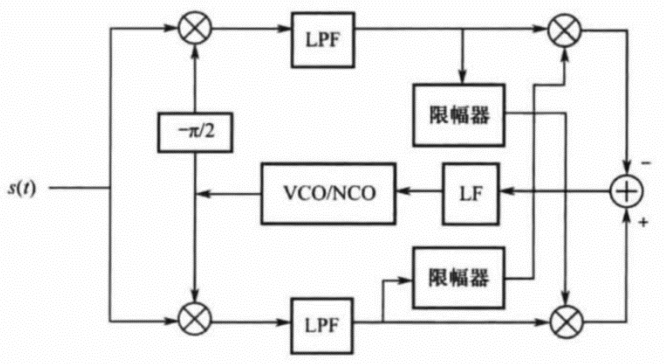


图 31 极性 Costas 示意图

以极性 Costas 环为例，可以推导出已调信号通过正交下变频后得到的同相和正交分量分别为

$$y_1(t)=\frac{I(t)}{2}\cos\left[\theta_e(t)\right]-\frac{Q(t)}{2}\sin\left[\theta_e(t)\right]$$

$$y_2(t)=\frac{I(t)}{2}\sin\left[\theta_e(t)\right]+\frac{Q(t)}{2}\cos\left[\theta_e(t)\right]$$

经过低通滤波器滤除高频分量后，计算出相位误差信号为

$$V_c(t)=y_2(t)\times sign\left[y_1(t)\right]-y_1(t)\times sign\left[y_2(t)\right]$$

最后可以得到锁定后的相位

$$D\left[\theta_e(t)\right]=\begin{cases}K_d sin\left[\theta_e(t)\right], -\pi/4\le\theta_e(t)\le\pi/4\\ -K_d\cos\left[\theta_e(t)\right], \pi/4\le\theta_e(t)\le 3\pi/4\\ -K_d\sin\left[\theta_e(t)\right], 3\pi/4\le\theta_e(t)\le 5\pi/4\\ K_d\cos\left[\theta_e(t)\right], 5\pi/4\le\theta_e(t)\le 7\pi/4\end{cases}$$

式中，$\theta_e(t)$ 为相位误差，鉴相特性曲线以 $\frac{\pi}{2}$ 为周期，锁定状态仍然可能处于 $0,\frac{\pi}{2},\pi,\frac{3\pi}{2}$ 附近，即载波相位提取存在四重不确定性。这也验证了进行差分编码的意义和重要性。

最后搭建出的载波同步仿真图为

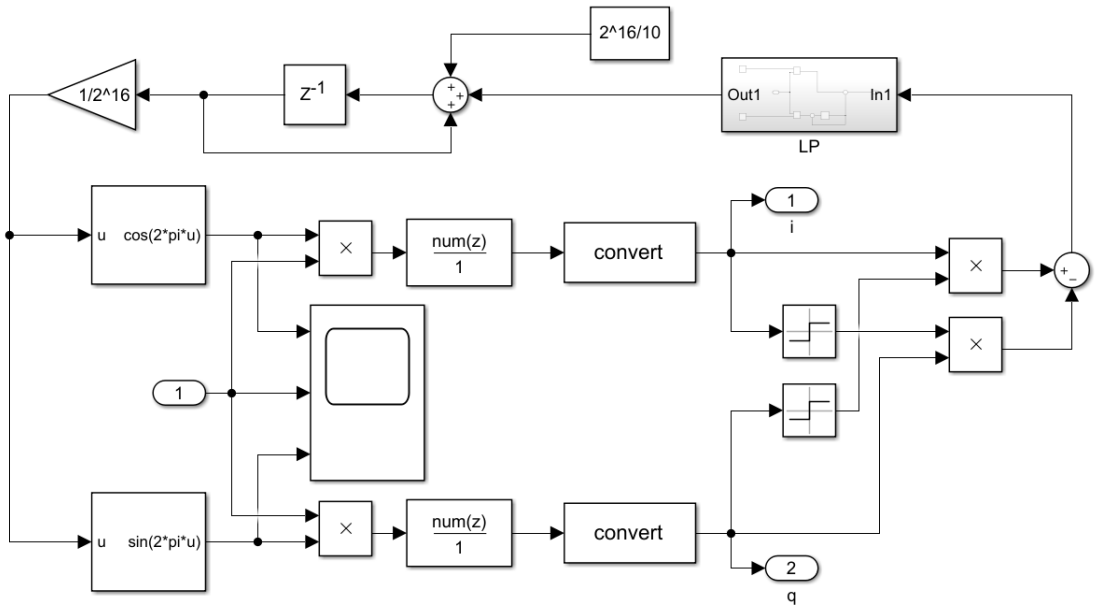


图 32 DQPSK 载波同步 simulink 仿真图

b. 2.位同步

位同步采用插值位同步算法，其基本原理框图如下所示

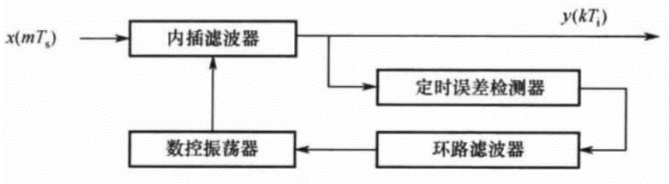


图 33 插值位同步算法示意图

内插滤波器是一种插值算法，其作用在于根据输入的信号，通过插值获得最佳时刻（插值时刻由数控振荡器控制产生）的采样值信号。由于接收端的采样时钟与发送端时钟

不同步，因此接收端时钟采样点的值可能没有所需的最佳采样值，但插值算法却可以根据采样信号值，以及数控振荡器送出的采样时刻信号和误差信号，通过插值算法获取最佳采样值;定时误差检测器在于检测本地时钟采样时刻与最佳采样时刻之间的相位差;检测出的相位差经环路滤波器滤波后，送数控振器产生下一个内插时刻。因此，我们只要知道定时相位误差，就可以计算出最佳判决点的采样值，从而实现码元同步。

其中位同步主要由三部分组成，Farrow 插值滤波器、Gardener 定时误差估计模块和 NCO 模块组成。

Farrow 立方型插值滤波器从最接近最佳内插时刻的连续 4 个输入的信号 $x\left[\left(m_k-1\right)T_s\right]$、$x\left[m_kT_s\right]$、$x\left[\left(m_k+1\right)T_s\right]$和$x\left[\left(m_k+2\right)T_s\right]$计算出内插值，其一般结构如下所示。

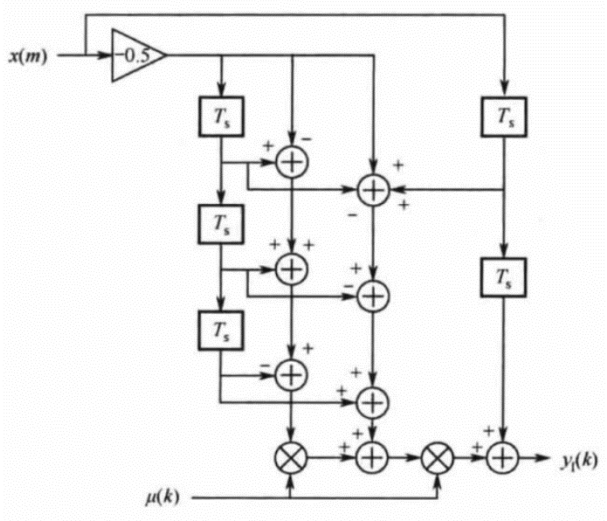


图 34 Farrow 内插滤波器结构图

Gardner 定时误差检测器可以检测出环路的位定时误差其作用类似于锁相环路中的鉴相器。Gardner 算法是非面向判决的，定时恢复独立于载波相位，通常用于同步的二进制基带信号或者 BPSK、QPSK 信号。考虑 QPSK 信号的I路和 Q 路支路，内插器在每个码元间隔内输出两个采样点，且序列对之间的采样点在时间上是一致的，因此，可以通过前后两个码元和两个码元之间的值进行定时误差的估算。

$$\mu_t(k)=y_I\left(k-\frac{1}{2}\right)\left[y_I(k)-y_I(k-1)\right]+y_Q\left(k-\frac{1}{2}\right)\left[y_Q(k)-y_Q(k-1)\right]$$

如果没有定时误差，那么中间点的值应该为零。如果中间点的值不为零，就可以用它的值来表示定时误差的大小，但是因为中间点处的斜率可能为正也可能为负，所以算法还需要两侧的两个峰值符号正负性来提供关于误差的正负方向。

$$\mu_t(k)=y_I\left(k-\frac{1}{2}\right)\left[sgn\left(y_I(k)\right)-sgn\left(y_I(k-1)\right)\right]+y_Q\left(k-\frac{1}{2}\right)\left[sgn\left(y_Q(k)\right)-sgn\left(y_Q(k-1)\right)\right]$$

数控振荡器 NCO 是一个相位递减器，可以通过数控振荡器的溢出时刻作为内插时刻点用于控制其他的模块工作。

最后搭建的 simulink 仿真图如下所示。

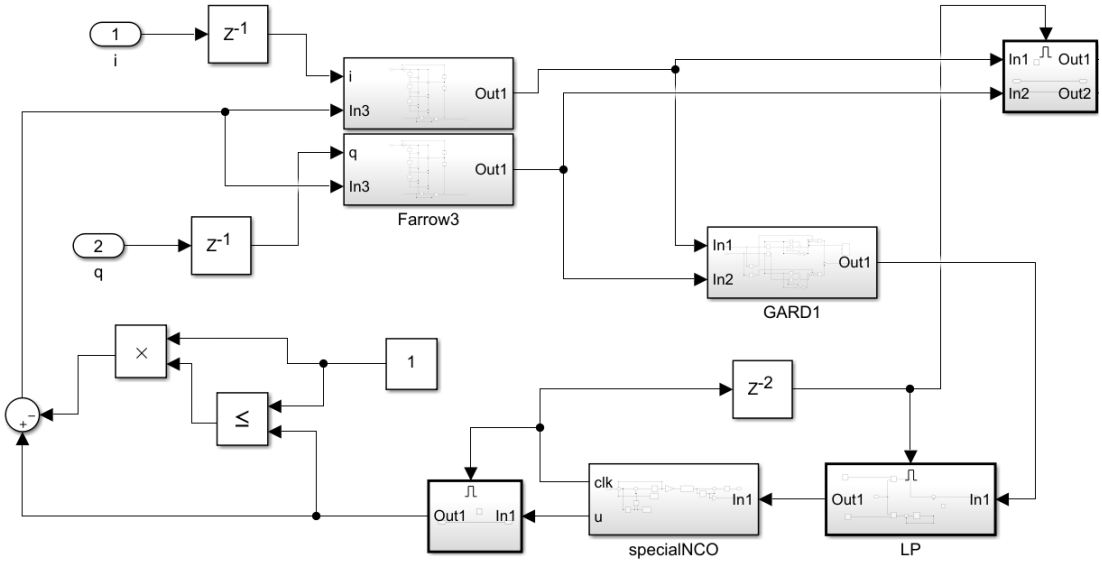


图 35 DQPSK 位同步 simulink 仿真图

3. 抽样判决

（我没看到？）

c. 差分解调

将本次判决结果和上一次的结果进行复数除法，获得相位差。将相位差当作初始相位为 0 的 QPSK 进行解调。

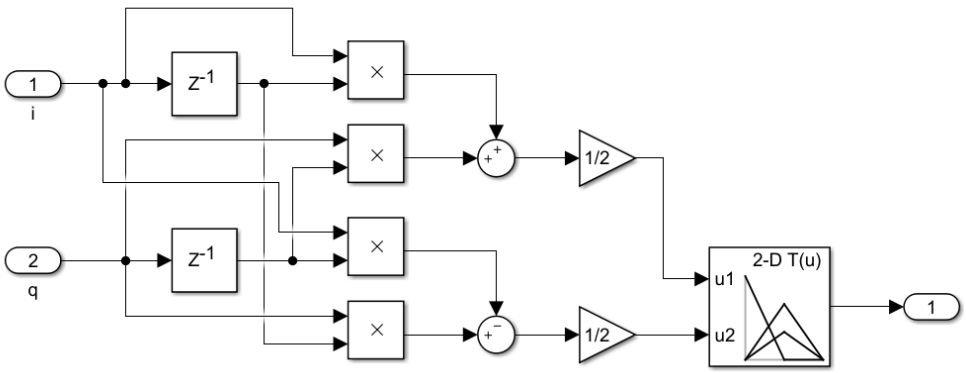


图 23 DQPSK 差分解调 simulink 仿真图

d. 并转串

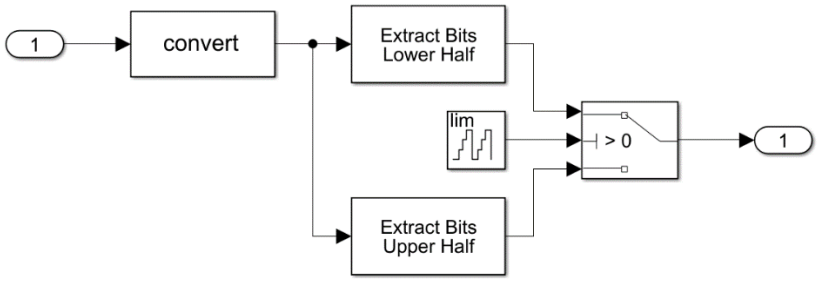


图 24 DQPSK 并转串 simulink 仿真图

1.4.10 16QAM 正交调制解调

QAM 是将信号加载到 2 个正交的载波上（通常是正弦和余弦），通过对这两个载波幅度调整并叠加，最终得到相位和幅度都调制过的信号。这两个载波通常被称为 I 信号，另一个被称为 Q 信号，所以这种调制方式也被称为 IQ 调制。

QAM 正交调制信号产生的原理框图如图所示

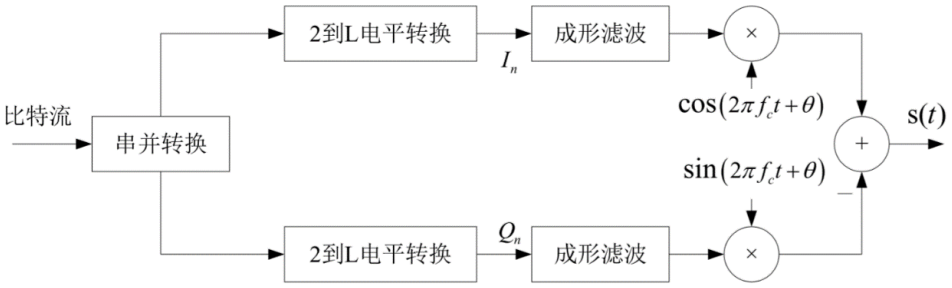


图 25 QAM 正交调制框图

QAM 正交调制信号解调的原理框图如图所示

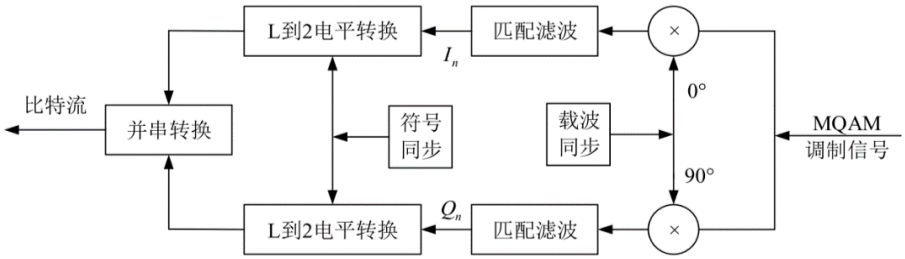


图 26 QAM 正交解调框图

(1) 发射机部分

a. 串转并

16QAM 调制方式需要将一路串行的二进制数据转换为两路并行的 I 路和 Q 路数据。其中，I 路和 Q 路各传输 2bit 数据。因此需要通过该模块将数据由串行数据转换为 4 并行数据。

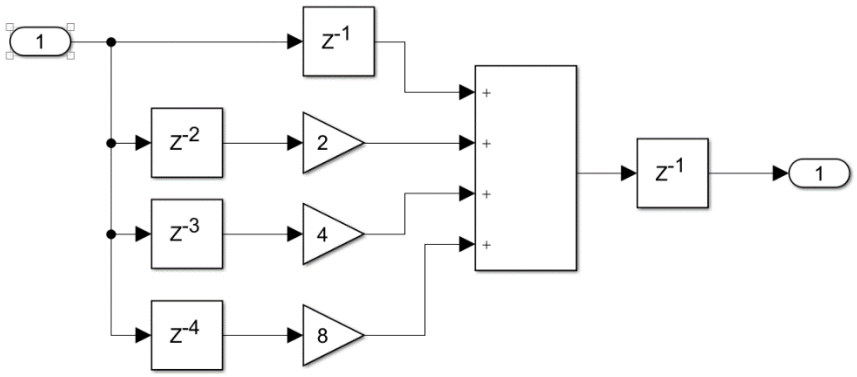


图 40 16QAM 串转并 simulink 仿真图

b. 差分调制

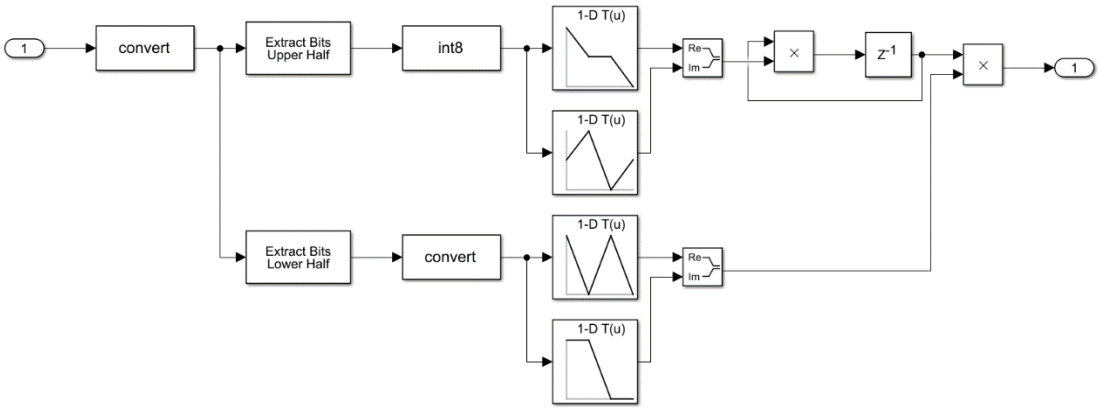


图 41 16QAM 差分调制 simulink 仿真图

c. 正交调制发射

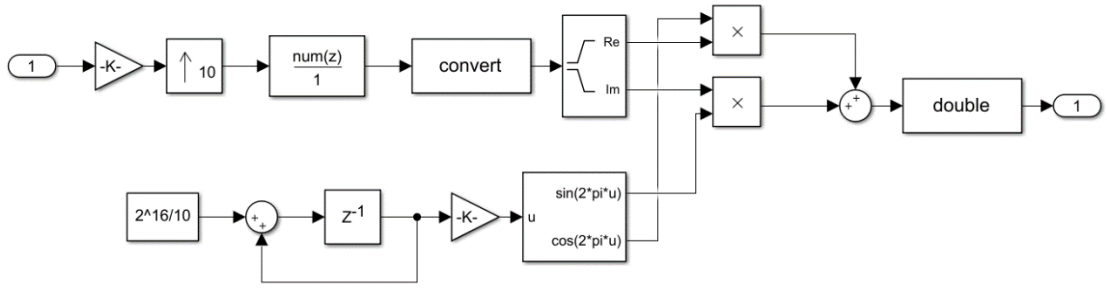


图 42 16QAM 正交调制 simulink 仿真图

正交调制发射部分包括上采样、成形滤波和正交上变频三个部分。

与 DQPSK 正交调制原理类似，将基带 01 数据序列经过串转并处理和差分调制后的数据序列进行 10 倍上采样，然后通过根升余弦滤波器进行成型滤波，然后将分别将同相分量信号和正交分量信号分别与 cos 信号和 sin 信号相乘后相加形成 16QAM 调制信号。

为了减少码间串扰，使用了根升余弦滚降滤波器。设计符号的数字频率和载波的数字频率相等，同时设计载波的数字频率是-0.2π。所以将符号 10 倍上采样，通过滚将系数为 0.2，长度为 6 个符号，采样率为 10 倍的根升余弦滚降滤波器。得到成型滤波结果后，使用数字频率-0.2π的 NCO 序列进行正交上变频。

(2) 接收机模块

a. 正交下变频

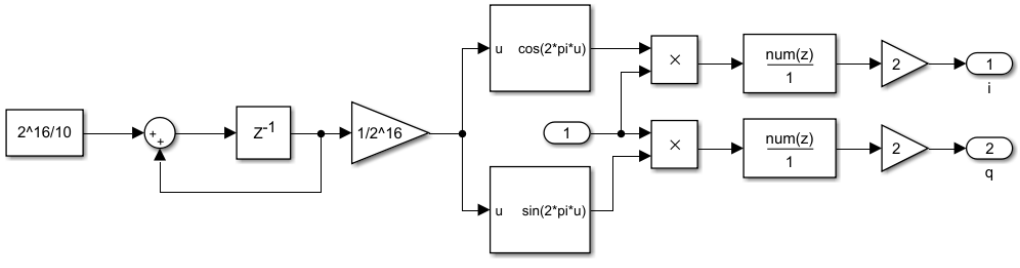


图 27 16QAM 正交下变频 simulink 仿真图

正交下变频的实现通过 DDS 原理，通过相位累加器的累加与溢出控制 sin 和 cos 的查找表控制产生同频的 sin 和 cos 信号从而实现已调信号的正交下变频。

b. 位同步

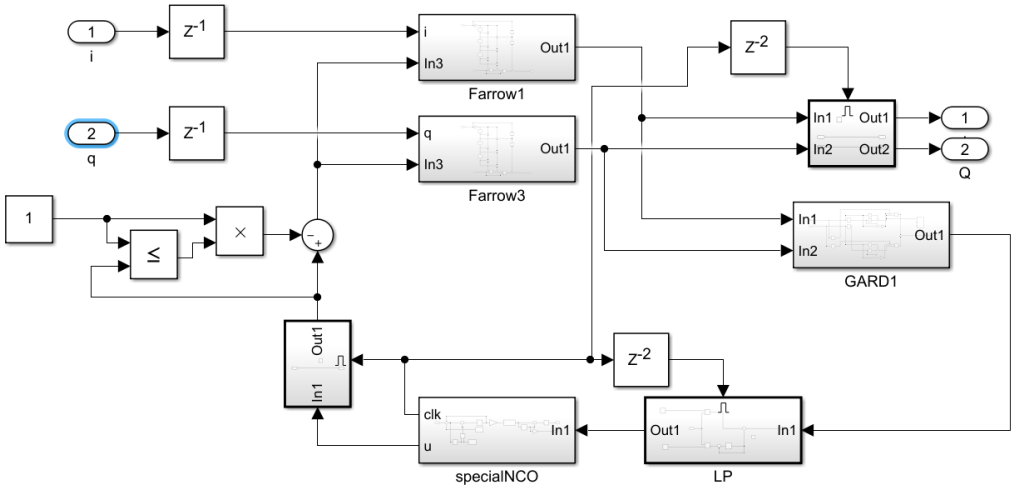


图 44 16QAM 位同步 simulink 仿真图

与 DQPSK 位同步原理类似，位同步仍然采用插值位同步算法。

其中位同步同样由三部分组成，Farrow 插值滤波器、Gardener 定时误差估计模块和 NCO 模块组成。

唯一的不同在于，由于 16QAM 调制信号可能会出现前后码元之和不等于 0 的情况，因此需要对 Gardner 算法进行完善和修正。

$$\mu_t(k)=\left[y_I\left(k-\frac{1}{2}\right)-a_I\right]\left[sgn\left(y_I(k)\right)-sgn\left(y_I(k-1)\right)\right]+\left[y_Q\left(k-\frac{1}{2}\right)-a_Q\right]\left[sgn\left(y_Q(k)\right)-sgn\left(y_Q(k-1)\right)\right]$$

其中，$a_I=\dfrac{y_I(k)+y_I(k-1)}{2}$，$a_Q=\dfrac{y_Q(k)+y_Q(k-1)}{2}$

c. 载波同步

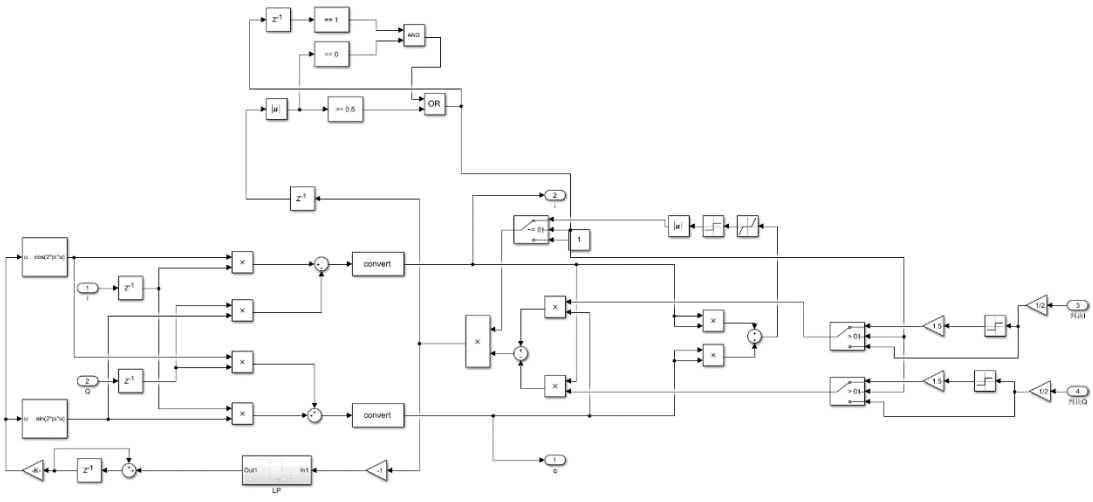


图 45 16QAM 载波同步 simulink 仿真图

对于 QAM 的载波同步算法，常用的算法有 DD 算法、RC 算法等。几乎所有的 QAM 载波同步算法的基础仍然是锁相环，QAM 各种载波同步算法中的组成与经典的锁相环路几乎完全相同，区别仅在于鉴相器的设计。

DD 算法的基本思想是将接收到的信号根据最近原则判到最近的量化星座点上。该方法将接收到的信号星座图与理想的星座图做比较，把两者的相位差值作为误差信号。它能有效去除加性噪声，并且适合所有形式的星座图，它的缺点是存在一个误码率阀值，当误码率过高时，采用此方法会导致大量错误的误差判决信号，所以一般在误差很小时用 DD 算法。

RC 算法就是对于 16QAM 和 64QAM 等方形星座，判断其相位是否锁定，仅仅比较四个角点符号和一个预先给定值的相位差，而不用检测星座图上的所有符号。比较所有接收符号的幅度平方与一个预定门限值来检测特别的角点符号。如果接收到的信号的平方大于等于这个门限，那么才对该符号和它在星座图上理想值进行比较，否则输出 0。

我们选取了 DDPLL 和 PolarPLL 相结合的方法。在频偏大于设定阈值时，采用时 PolarPLL 快速收敛，否则选用 DDPLL 减少相位误差。

d. 抽样判决

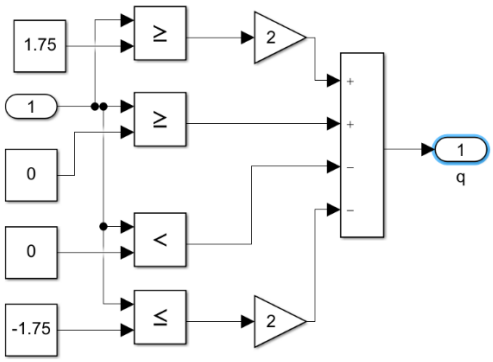


图 28 16QAM 抽样判决 simulink 仿真图

通过对解调之后的信号进行抽样，并以 1.75 为门限值进行判决最后得到基带信号。

经过位同步和载波同步后，可以直接进行判决输出。判决结果也用于 DDPLL 的鉴相。

e. 差分解调

将判决结果的象限关系进行差分解调，参考 QPSK 差分解调方式，获取信号的高 2bit。将象限内的位置进行直接解调，获得信号的低 2bit。

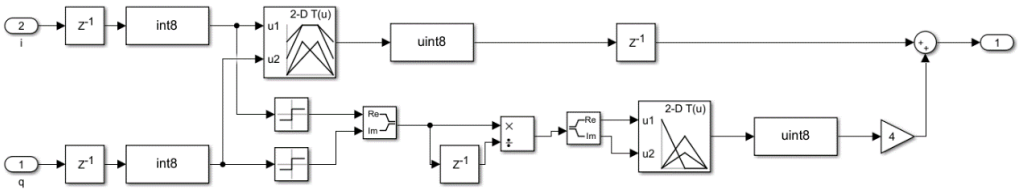


图 29 16QAM 差分解调 simulink 仿真图

f. 并转串

将 4 并行数据转换为串行数据。

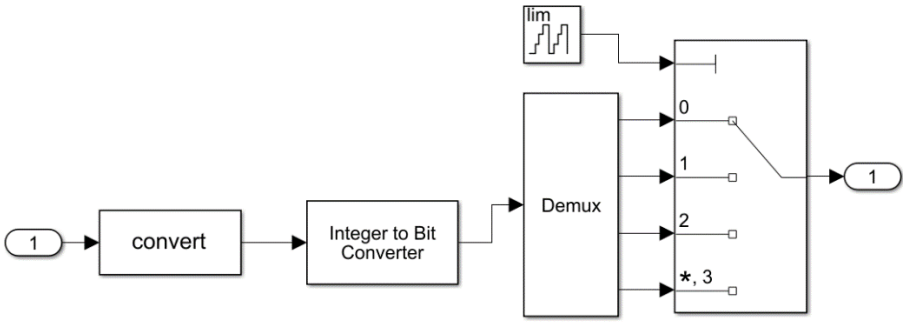


图 30 16QAM 并转串 simulink 仿真图

1.4.11 数据测试

(1) 文本信号

我们选取一份具有文本的 txt 文件作为我们的信源输入。

“碧翠丝是长月达平所创作的轻小说《Re:从零开始的异世界生活》及其衍生作品的登场角色。

简介

住在罗兹瓦尔宅邸内只有通过机遇门才能进入的隐藏房间禁书库。

拥有管理 10 万 3000 本禁书库义务的萝莉幼女。

可爱的外貌和穿着豪华的礼服，看上去给人的印象就像妖精一样。

总是不高兴、自大且讨厌人类，对精灵帕克有着哥哥的爱称。

与昴初次见面的互相印象都很差，即便是趣味相投也没见关系加深。

平时在禁书库内一个人看守，坐在凳子上等待着谁的到来。

有着极高的阴魔法造诣。”

采用线性分组码的信道编码方式和 DQPSK 的正交调制解调方式。GUI 选择信道编码方式和调制方式如下如所示。

图 31 文本信源测试 GUI 设置

其中，DQPSK 中间信号如下所示

星座图在位同步和载波同步完成之前和完成之后的星座图分别如下所示

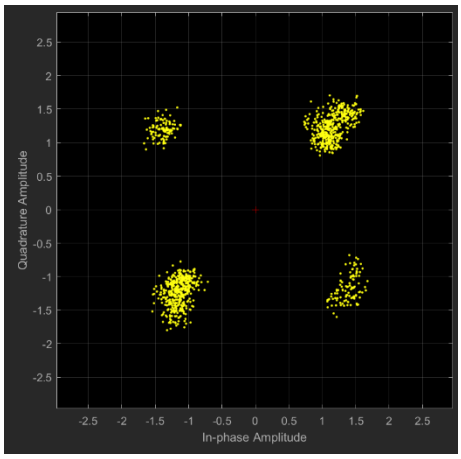


图 50 DQPSK 同步前星座图

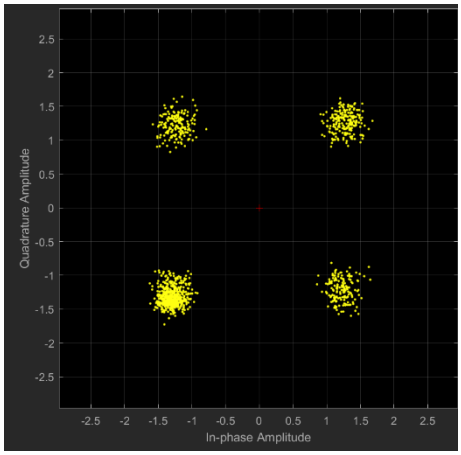


图 51 DQPSK 同步后星座图

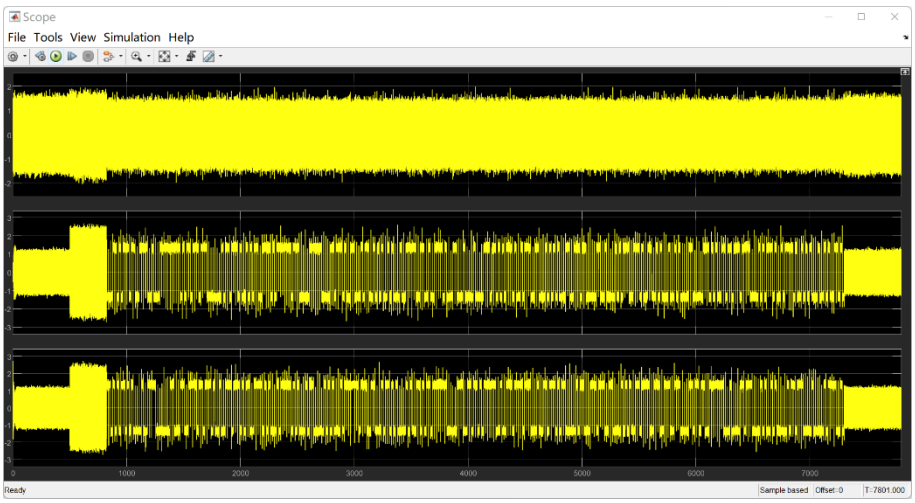


图 52 I 路和 Q 路信号载波同步结果图

从星座图看出，位同步和载波同步实现了对信号同步的恢复。

最终输入和输出信号之间的信号关系如下所示：

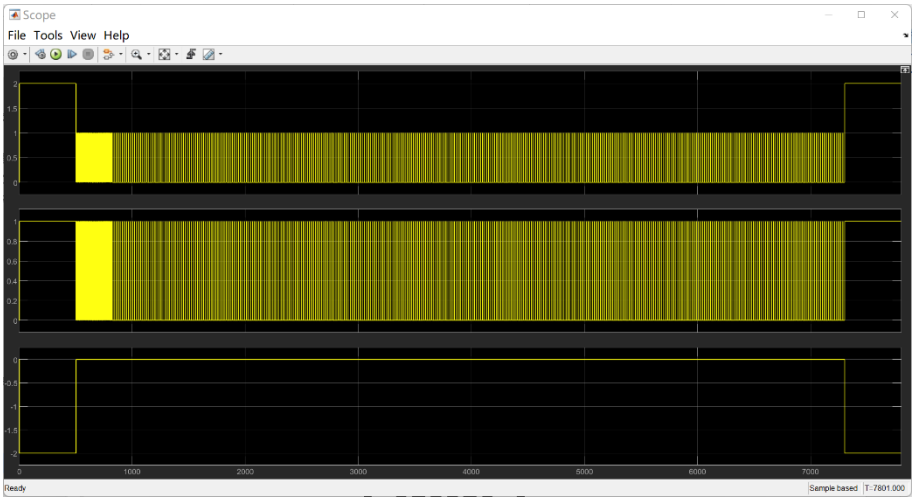


图 53 文本信源 DQPSK 调制最终结果图

可以看出最终输出信号与输入信号之间的误码率非常低。

(2) 图像信号

我们选取如下图像进行测试：

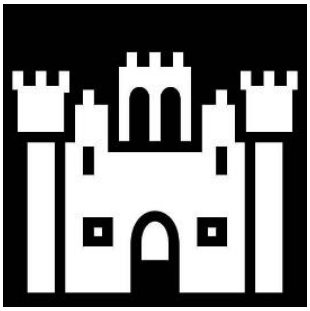


图 54 图像信源测试图

采用卷积码的信道编码方式和 16QAM 的正交调制解调方式。GUI 选择信道编码方式和调制方式如下如所示。

图 55 GUI 设计选择界面图

其中，16QAM 中间结果图如下

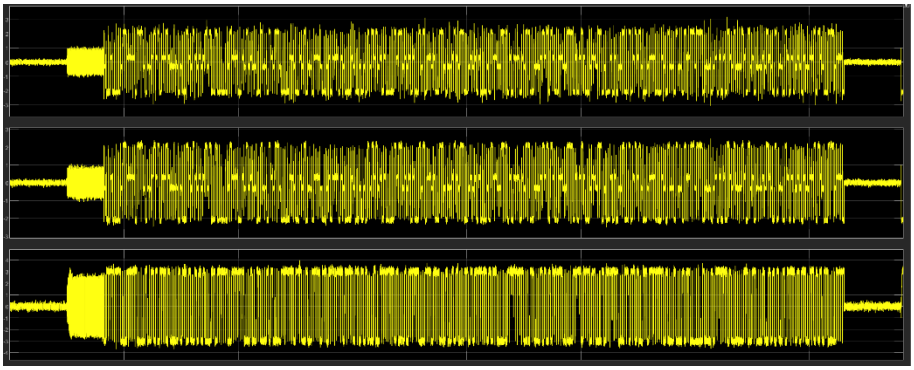


图 32 16QAM 信号中间结果图

可以看到，以上为通过正交下变频、位同步、载波同步之后的 I 路信号。经过不断处理后信号得到载波恢复和位同步恢复的信号，其星座图得到恢复矫正，如下所示

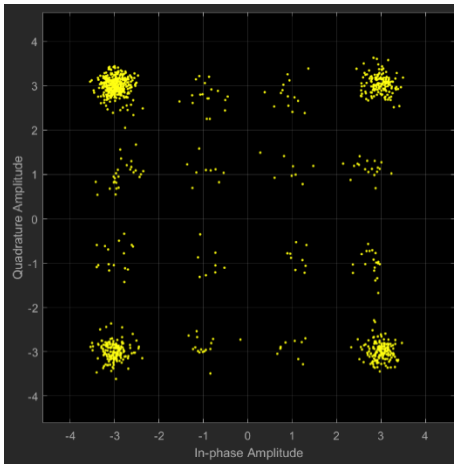


图 33 16QAM 同步恢复后星座图

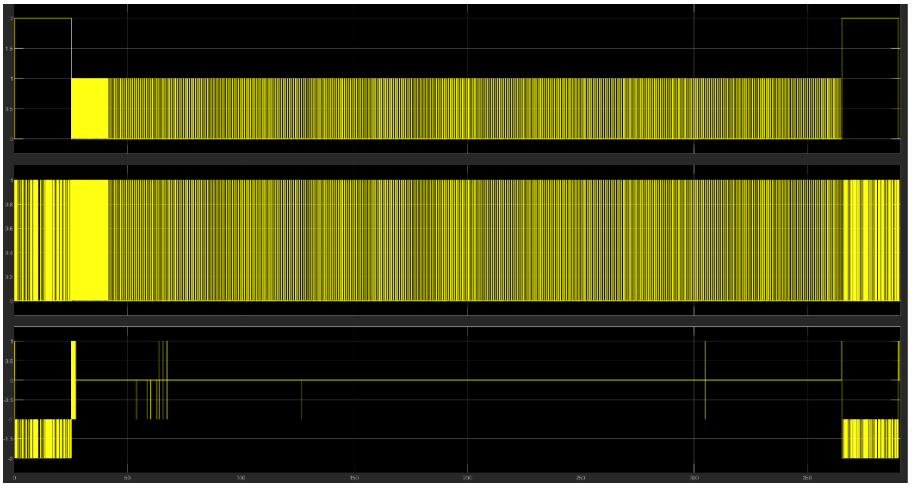


图 34 最终输入输出对比图

如上所示，最终输入和输出信号之间误差并不大，误码率并不高，可以通过后续信道译码进行纠错和恢复，再通过帧同步分析后最终得到如下的结果图。

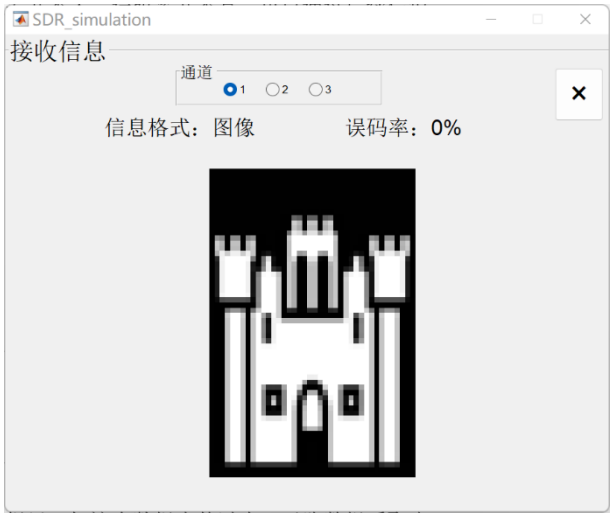


图 35 最终接受结果图

1.5 结果分析

软件无线电总体上来说是一种通信系统架构的设计思想，将许多通信的模块进行软件化的实现，从而让整个通信过程更加灵活。

通过软件无线电的通用结构使用信源编码、信道编码、数据成帧、正交调制与解调、帧同步、信道解码和信源解码后，实现了基本的通信收发的过程。其中，使用了数字通信中的各种同步技术，位同步技术和载波恢复技术等，扩展增加了系统的功能和复杂度。最终可以看到可以以低误码率实现通信双方文本、图像、语音等信号的传输。

1.6 方案改进

1.6.1 信源编码

霍夫曼编码的一个核心问题是，接收方该怎么知道发送方的霍夫曼树。发送方读取不同文本，不同文本的符号的出现频次不同，即权重不同，构造的树也不一样，接收方怎么知道这个树？

一种方案是规定一个确定的权重来构造树。实际上这也是常用的方法，对于一个功能确定的传输系统，我们可以大致确定各个符号出现的频率，从而给出一个较为合理的权重。但这对我们设计的传输系统似乎并不适用，如果输入的数据频次和确定的权重相差太大，这样做反而会使编码序列变长。

另一种方案是构造的树存在一个元胞数组里，把这个数组也传过去，元胞数组采取直接编码的方式（二进制矩阵）。这个方案的问题在于，接收方接受时这部分一旦出现错码，将对信源译码造成较大的影响。理论上这个影响可以通过增加重复码的方式减小，但是这与霍夫曼编码压缩数据的初衷相悖。因此，我们只能通过在信道编码阶段增加纠错码的方式尽量降低这个影响。

1.6.2 帧同步

最初提出帧同步主要是为了解决时延问题，经过 simulink 仿真模型后，序列会产生时延。对于一个模型，我们当然可以知道其时延的具体值，但是我们希望我们的系统能够启

用多种模型（这也是因为我们需要实现多种调制方式）的同时，能够自行计算其延时，而不是提取告知代码延时的具体值。于是，我们决定将数据序列包装在帧中，即通过成帧和帧同步来解决这个问题。

第一个方案是在序列的头尾和中间部分插入帧间隔符 0 1 1 1 1 1 1 0。具体方法是，在成帧时，遇见 5 个 1 就插 0，在接收端提取数据时见 5 个 1 去 0。这个方案存在的问题是经过信道后出现的误码，可能导致帧间隔符的丢失，以及产生错误的帧间隔符，这对提取的数据会产生较大影响，实际实现后效果也并不好。

第二个方案是在序列的头尾和中间部分插入帧间隔符 1 0 0 1 1 0 1 0 1 1 1 1 0 0 0。成帧时，我们将序列分割为 100 个帧，每个帧等长，不够则用 0 补齐。帧同步时，计算接受数据和帧间隔符的相关性（变为双极性后），将大于门限值（如帧间隔符长度减 2）的位置储存为可能的帧头位置。接着，我们计算所有可能帧头位置的差值，取其中的众数作为帧长（数据长度加间隔符长），再将不等于帧长的差值的索引记录下来，根据正常的索引值及帧长进行修正(即去除错误的帧头位置)，最后得到帧头位置，根据帧头位置提取数据。这个方案经过代码实现后，具有较好的效果，大多数情况下也能够提取出发送的数据序列。然而，代码中关于修正帧头位置的部分过于繁琐，且有可能丢失一个帧头，以至于产生较大的影响。通过规定帧数为 100，则帧头有 101 个来尝试解决这个问题，进行修正后若帧头数少于 101，则从当前的帧头两端出发，降低门限值，寻找可能的帧头。然而，这样做带来了更加冗长的代码，同时这也不符合实际情况。

第三个方案采用了以太网帧的结构。规定了数据帧是定长，不够则填充 0。在数据序列前加上前导码和帧开始定界符，接收端通过检测特定序列来进行帧同步，之后提取出定长的数据即可。我们设想可以发送多个帧，帧与帧之间用特殊的电平进行间隔，即仿真时发射机仅在非该电平时进行发送。然而，这意味着需要反复调用模型进行仿真，因此最后我们决定只发一个帧，帧长根据当时仿真采用的图片信源的固定 size 得到。之后，我们遇到了一个新问题，信道的突发差错，如下图所示，信道中成串出现特殊差错。我们采用了交织，在一定程度上缓解了该问题，由于帧长固定，交织矩阵的宽度也是一个定值，为帧长的开方。

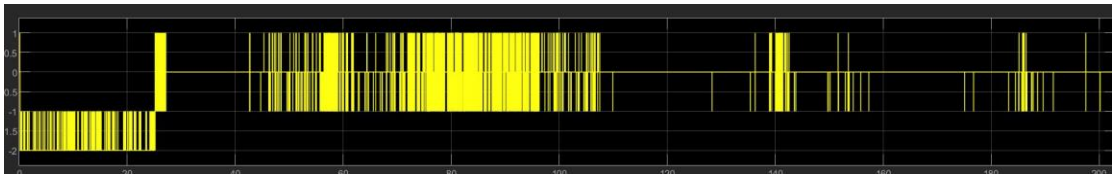


图 60 某次仿真输入和输出序列的差值

接着，一个新问题出现，我们发现当使用图片或音频作为信源时，误码率较低，而使用文本作为信源时，误码率较高。我们一直不明白其中的原因，直到一次测试数据，发现使用线性分组码对文本进行编码时，误码率竟然比卷积码低，收到的文本信息几乎没有差错，这种反常已经足够让我们意识到数据组织结构出了问题。

我们很快注意到，当采用图片或音频信源时，信息序列几乎占据了固定帧长的所有点，只填充了很少的 0；然而，当我们采用文本信源时，由于文本编码序列的点数较少，填充的 0 的数量反而远大于文本编码序列的长度，这不仅使得无意义的仿真时间增加，还对卷积译码造成了影响。卷积码译码采用了概率译码，理论上我们填充的 0 可以被视为零状态，不会影响有效信息的译码结果，然而，填充 0 的部分经过信道后会出现差错，它们

并非经过编码的序列，却依旧被用于译码，这意味着我们强行添加了许多不应该出现的错误状态，给译码带来了影响，未编码序列越长，这种影响越显著。

最终，我们决定在第三个方案的基础上稍加修改，在帧中增加帧长字段，并调整交织矩阵宽度由输入序列的长度计算得到。

五、 设计结论

小组成功完成了软件无线电课程设计基本要求。在本软件无线电系统设计之初，我们考虑进行文本、图像、音频三种信源的传输。对于文本信源，我们直接进行 unicode 编码和使用两个字节记录文本长度，再转换成为二进制序列。对于图像信源，我们考虑了两种方法。一种是直接把图像矩阵转换成向量并在开头加入图像大小，再转换成为二进制序列。另一种方式是使用霍夫曼编码，但是需要传输编码表，这在图像较小时会削弱压缩效果。对于音频信源，我们使用脉冲编码调制（PCM），但是由于传输音频需要较多的点数，我们最终测试时每次只传输几秒的音频。

信道编码部分，我们采用了卷积码以及线性分组码两种方式。卷积码的纠错性能较好，但算法复杂度较高且编码效率低；而线性分组码的纠错性能不如卷积码，但算法复杂度低且编码效率较高。此外，我们还需要解决如何较为准确地提取出经过编码的序列，以实现正确地译码的问题，而不能直接将包括非编码序列在内的仿真输出全部进行译码。

交织部分，我们采用较为简易的交织方式，将输入序列进行分组，组数为序列长度的开方值（序列不足则补 0），之后将每组中索引相同的数据重新分为一组。

成帧部分，我们加入了前导码方便接收方进行时钟同步，解决了系统的时延问题；加入帧长字段，则便于我们提取有效信息序列。此外，为了方便仿真，我们仅发送一帧。这在实际中并不可取，但的确减小了系统仿真时的负担，是可行的。

在正交调制解调部分，我们分别实现了 DQPSK 和 16QAM 的正交调制解调模型，并且实现了简易的载波同步与位同步算法实现，在中等信噪比的情况下也能比较好实现通信的收发功能。

在设计软件无线电软件中，我们也遇到了许多需要深思的问题。比如，直接把信源转换成为二进制序列是否符合信源编码的初衷？在信道较高误码率的信道条件下，我们的发帧方式是否能支持纠错码进行有效纠错？我们将会在今后的学习中继续探索这些问题的答案。

六、 总结及心得体会

经过本课程的软件无线电设计，我们收获良多。首先，我们对软件无线电的结构有了一个全面的了解，对软件无线电在无线电领域的优势和前景有了更深刻的理解。其次，在完成我们题目一课程设计的过程中，我们深入学习了信源编码、信道编码、调制解调、载波同步等相关内容，并且使用了 m 语言和 simulink 进行了相关模块的实现和仿真，把所学知识及时应用到实践中去，最终成功完成了课程设计。

在完成课程设计的过程中，我们也遇到了一些困难。比如怎样解决 m 语言和 simulink 的连接问题，以及系统传输较大文件耗时过长和怎样设计一个 GUI 界面完整展示整个软件无线电通信的过程等等。但经过我们的共同努力，这些问题都被很好解决了。

最重要的是，在完成课程设计的过程中，我们进行了模块划分，每个人负责一部分，最后再交流完成情况。这样既锻炼了我们独立解决困难和完成任务的能力，也提升了我们与别人合作学习意识。这些经历和经验对于我们将来的学习有着重要意义。

参考文献

[1] Mitola J. Software radios: Survey, critical evaluation and future directions[J]. IEEE Aerospace and Electronic Systems Magazine, 1993, 8(4): 25-36.

[2] Lackey R I, Upmal D W. Speakeasy: the military software radio[J]. IEEE Communications Magazine, 1995, 33(5): 56-61.

[3] Oppenheim J, Bittinger K, Bowman R, et al. Defense Acquisitions: Restructured JTRS Program Reduces Risk, but Significant Challenges Remain[R]. United States Government Accountability Office Washington United States, 2006.

[4] Sadiku M N O, Akujuobi C M. Software-defined radio: a brief overview[J]. Ieee Potentials, 2004, 23(4): 14-15.

[5] Ray K P, Kumar G. Multi-frequency and broadband hybrid-coupled circular microstrip antennas[J]. 1997.

[6] Deepukumar M, George J, Aanandan C K, et al. Broadband dual frequency microstrip antenna[J]. Electronics Letters, 1996, 32(17): 1531-1532.

[7] Carey-Smith B E, Warr P A, Beach M A, et al. Wide tuning-range planar filters using lumped-distributed coupled resonators[J]. IEEE Transactions on Microwave Theory and Techniques, 2005, 53(2): 777-785.

[8] Watkins G. Potential interfering signals in software defined radio[C]//6th IEEE High Frequency Postgraduate Colloquium (Cat. No. 01TH8574). IEEE, 2001: 41-46.

[9] Nesimoglu T, Charalampopoulos Z, Beach M A. Interference suppression in radio receivers by using frequency retranslation[C]//2009 IEEE 10th Annual Wireless and Microwave Technology Conference. IEEE, 2009: 1-5.

[10] Haghighat A. A review on essentials and technical challenges of software defined radio[C]//MILCOM 2002. Proceedings. IEEE, 2002, 1: 377-382.

[11] Abidi A A. Evolution of a software-defined radio receiver's RF front-end[C]//IEEE Radio Frequency Integrated Circuits (RFIC) Symposium, 2006. IEEE, 2006: 4 pp.

[12] Walden R H. Analog-to-digital converter survey and analysis[J]. IEEE Journal on selected areas in communications, 1999, 17(4): 539-550.

[13] Hentschel T, Henker M, Fettweis G. The digital front-end of software radio terminals[J]. IEEE Personal communications, 1999, 6(4): 40-46.

[14] Collins A. All programmable RF-sampling solutions[J]. Xilinx White Paper, 2017.

[15] Nguyen A Q, Kisomi A A, Landry R. New architecture of Direct RF Sampling for avionic systems applied to VOR and ILS[C]//2017 IEEE Radar Conference (RadarConf). IEEE, 2017: 1622-1627.

[16] Choo L C, Lei Z. CRC codes for short control frames in IEEE 802.11 ah[C]//2014 IEEE 80th Vehicular Technology Conference (VTC2014-Fall). IEEE, 2014: 1-5.

[17] Berns F, Kreiselmaier G, Wehn N. Channel decoder architecture for 3G mobile wireless terminals[C]//Proceedings Design, Automation and Test in Europe Conference and Exhibition. IEEE, 2004, 3: 192-197.

[18] A. Kale, J. Sturm and V. S. R. Pasupureddi, "0.9 to 2.5 GHz Sub-Sampling Receiver Architecture for Dynamically Reconfigurable SDR," 2018 IEEE Computer Society Annual Symposium on VLSI (ISVLSI), Hong Kong, China, 2018, pp. 316-320, doi: 10.1109/ISVLSI.2018.00065.

[19] Zong Wang, A. T. Erdogan and T. Arslan, "A SDR platform for mobile Wi-Fi/3G UMTS system on a dynamic reconfigurable architecture," 2009 17th European Signal Processing Conference, Glasgow, UK, 2009, pp. 2678-2682.

[20] J. Oza et al., "Optimized configurable architecture of modulation techniques for SDR applications," International Conference on Computer and Communication Engineering (ICCCE'10), Kuala Lumpur, Malaysia, 2010, pp. 1-5, doi: 10.1109/ICCCE.2010.5556800.