无线通信系统的
DSP 实现

王金龙　沈良　任国春　蔡跃明　陈瑾　吴启晖编著

《无线通信系统的 DSP 实现》内容简介

随着数字技术的迅速发展和市场需求的强劲推动，基于数字信号处理器（Digital Signal Processor，简称 DSP）的实时信号处理技术得到了越来越广泛的应用。本书在简要介绍数字信号处理理论与方法的基础上，以目前国内外使用广泛的 TI 公司 DSP 芯片为代表，讨论了无线通信中数字滤波、数字调制、信道编码、同步、均衡、分集接收等算法的设计、DSP 实现以及工程实际中可能遇到的问题，同时给出了许多算法的 MATLAB 仿真程序和 DSP 汇编程序。最后，简要介绍了软件无线电的基本概念、发展概况和关键技术，并以一种实用的软件无线电台为例，分析了该电台基带处理的关键算法及 DSP 实现。

本书可作为电子学和通信类各专业本科高年级学生和研究生的教材或参考书，也可供相关专业的教师和工程技术人员阅读参考。

前言

近年来，随着微电子技术、数字信号处理技术的飞速发展，数字信号处理器（Digital Signal Processor，简称 DSP）得到了日新月异的进步，在处理速度、运算精度、处理器结构、指令系统、指令流程等诸多方面都有了较大提高，并迅速在语音、雷达、声纳、地震、图像、通信系统、系统控制、生物医学工程、遥感遥测、地质勘测、航空航天、电力系统、故障检测、自动化仪表等众多领域获得了极其广泛的应用。当前，以 DSP 芯片及外围设备为主，正在形成一个具有较大潜力的产业和市场。

数字化是当今无线通信系统的主流发展方向，而 DSP 的重要特点是其处理速度远远高于一般的微处理器，适用于实时数据处理。因此，DSP 在无线通信系统的实现中日趋重要，无论是基地台、终端设备，还是手机、背负式电台，都要依靠高性能的 DSP 来实现。DSP 技术已成为现代无线通信系统的核心技术之一。

伴随无线通信和数字信号处理技术的发展，国内外出版了大量有关无线通信或数字信号处理的著作和教材，但结合无线通信和数字信号处理器两方面的专著和教材还甚少，国内缺乏系统介绍数字信号处理方法与无线通信应用的书籍。本书拟在填补这一空白，为从事通信和数字信号处理应用的工程科研人员、高年级学生、研究生和教师提供一部富有使用和参考价值的教材和参考书。

本书各章在简明扼要介绍数字信号处理理论与方法的基础上，以目前国内外使用广泛 DSP TMS320C54x 为代表，讨论无线通信中的各种信号处理算法的实现及实践中可能遇到的问题。针对无线通信系统不同功能模块，重点介绍了数字滤波器、数字调制、信道编码、同步、均衡、分集接收等算法及 DSP 实现，同时给出了许多算法的 MATLAB 仿真程序；最后，简要介绍了软件无线电的基本概念、发展概况和关键技术，并以一种实用的软件无线电台为例，分析了该电台基带处理的关键算法及 DSP 实现。

本书由王金龙主编。全书共分 9 章，其中第 1 章由王金龙编写，第 2 章由沈良编写，第 3~6 章由任国春编写，第 7 章由陈瑾编写，第 8 章由蔡跃明编写，第 9 章由吴启晖编写，全书由王金龙统稿。在编写过程中，得到了人民邮电出版社的大力支持，我校研究生龚玉萍、高瞻、童晓兵、程云鹏、李悦、李琳和汤宏伟等做了许多工作。在此一并表示感谢。

由于编著者水平有限，错误和疏漏之处在所难免，敬请广大读者批评指正。

作者

2002 年 2 月于南京

第一章　DSP 系统导论

1.1 概述

数字信号处理（Digital Signal Process，简称 DSP）是利用专用或通用数字信号处理芯片，通过数字方法实现信号处理。与模拟信号处理相比，数字信号处理具有灵活、精确、可靠性好、体积小、功耗低、易于大规模集成等优点。

数字信号处理的基础是算法和数字计算机或数字信号处理芯片。算法一旦建立，设计者就要寻找合适的计算机或数字信号处理芯片来最有效地实现它们，最开始的目标是在可以接受的时间内对算法做仿真，随后是将波形存储起来，然后再加以处理。随着计算机技术和数字信号处理技术与大规模集成电路技术的发展，这种仿真和脱机处理逐步演变成为实时信号处理。

实时信号处理是指系统必须在有限的时间内对外部输入信号完成指定的处理功能，即信号处理速度应大于信号更新速度，这主要取决于数字信号处理芯片的处理速度与功能，下面主要叙述 DSP 芯片的简单情况及其应用。

1.1.1 DSP 芯片及其特点

DSP 芯片，也称数字信号处理器（Digital Signal Process，也简称 DSP，后面大部分缩写均属此含义），是一种特别适合于进行数字信号处理的微处理器，它强调运算处理的实时性，因此 DSP 芯片除了具备普通微处理器所强调的高速运算和控制功能外，针对实时数字信号处理，在处理器结构、指令系统、数据流程上做了大的改动，其特点如下：

1. DSP 芯片普遍采用了数据总线和程序总线分离的哈佛结构及改进的哈佛结构，比传统处理器的冯・诺依曼结构有更高的指令执行速度；
2. DSP 芯片大多采用流水技术，即每条指令都有片内多个功能单元分别完成取指、译码、取数、执行等多个步骤，从而在不提高时钟频率的条件下减少了每条指令的执行时间；
3. 片内有多条总线可以同时进行取指令和多个数据存取操作，并且有辅助寄存器用于寻址，它们可以在寻址访问前或访问后自动修改内容，以指向下一个要访问的地址。
4. DSP 芯片大多带有 DMA 通道控制器以及串行通信口等，配合片内总线结构，数据块传送速度大大提高。
5. 配有中断处理器和定时控制器，可以方便地构成一个小规模系统。
6. 具有软、硬件等待功能，能与各种存取速度的存储器接口。

7. 针对滤波、相关、矩阵运算等需要大量乘法累加运算的特点，DSP 芯片大都配有独立的乘法器和加法器，使得同一时钟周期内可以完成乘、累加两个运算。
8. 低功耗，一般为 0.5~4W，采用低功耗技术的 DSP 芯片只有 0.1W，可用电池供电。

正是 DSP 芯片的以上特点决定了其运算速度比通用微处理器（MPU）要高，例如 FIR 滤波器的实现，每输入一个数据，对应每阶滤波器系数需要一次乘、一次加、一次取指、两次取数，有时还需要专门的数据移位操作，DSP 芯片可以单周期完成乘加并行操作以及 3~4 次数据存取操作，而普通 MPU 至少需要 4 个指令周期，因此，在相同的指令周期和片内指令缓存条件下，DSP 是 MPU 运算速度的 4 倍以上。

1.1.2 DSP 芯片种类

DSP 芯片的采用是为了达到实时信号的高速处理，为适应各种各样的实际应用，出现了多种类型、档次的 DSP 芯片。从用途上可以把 DSP 芯片分为通用 DSP 芯片和专用 DSP 芯片，通用 DSP 芯片一般指可以用指令编程的 DSP 芯片，而专用 DSP 芯片只针对一种应用，只能通过加载数据、控制参数或在管脚上加控制信号来使其具有有限的可编程能力。按数据格式可以把 DSP 芯片分为定点 DSP 芯片和浮点 DSP 芯片，数据以定点格式工作的 DSP 芯片称为定点 DSP 芯片，数据以浮点格式工作的 DSP 芯片称为浮点 DSP 芯片。评价专用 DSP 芯片性能的主要指标是它完成相应处理任务的速度以及字长，评价通用 DSP 芯片最常用的指标是 MIPS（每秒执行百万条指令）。对大多数定点 DSP 芯片来说，单周期内可以完成一次甚至两次运算。对浮点 DSP 芯片单周期可以完成 2 次甚至多次运算，通常衡量浮点 DSP 芯片的指标是 MFLOPS（每秒百万次浮点运算）。TI 公司的 TMS320C30 每指令周期可以执行乘法和加法各一次，因而其 MFLOPS 指标是其 MIPS 指标的两倍。AD 公司的 ADSP21020 与 Motorola 公司的 DSP96002 在一个指令周期内可以完成乘、加、减各一次，因而其 MFLOPS 指标是其 MIPS 指标的三倍。DSP 芯片内除了运算单元外还有许多其它功能部件，MOPS（每秒百万次操作）就成了衡量 DSP 芯片片内功能强弱的又一指标。这一指标可以达到 MIPS 指标的 5~10 倍，但这一指标并不能与 DSP 芯片的实际处理速度等同，因此执行 FFT、FIR 滤波等算法的执行时间就成为一个比较客观的评价标准。一般情况所说的通用 DSP 芯片的性能指标往往只是在 DSP 芯片执行片内程序及读写片内存储器数据的条件下得来的，实际上如果将程序和数据放在片外存储器，DSP 芯片的处理速度要慢 2~3 倍，因此片内存储器的大小对 DSP 芯片性能影响较大。大容量片内存储器配置的 DSP 芯片对系统设计简化和性能提高非常有效。

通用 DSP 芯片的运算和处理是用软件实现的；而专用 DSP 芯片的运算是用硬件直接实现的，其内部结构规则简单，通常可以容纳很多相同的运算单元，如多个乘加器，因此专用 DSP 芯片在做指定运算时，速度远高于通用 DSP 芯片。其缺点是灵活性差，几乎都是定点型的，精度和动态范围有限，需要较多外围控制器件和严格的时钟同步信号，并且专用 DSP 芯片几乎不具备自适应处理能力。

1.1.3 DSP 芯片的应用

DSP 芯片的应用几乎已遍及电子与信息的每一个领域，常见的典型应用有：

1. 通用数字信号处理

数字滤波、卷积、相关、FFT、希尔伯特变换、自适应滤波、窗函数、谱分析等。

2. 语音识别与处理

语音识别、合成、矢量编码、语音鉴别、语音信箱等。

3. 图形/图像处理

二维、三维图形变换处理、模式识别、图像鉴别、图像增强、动画、电子地图、机器人视觉等。

4. 仪器

暂态分析、函数发生、波形产生、数据采集、石油/地质勘探、地震预测与处理等。

5. 军事

雷达与声纳信号处理、导航、导弹制导、保密通信、全球定位、电子对抗、情报收集与处理等。

6. 计算机

阵列处理器、图形加速器、工作站、多媒体计算机等。

7. 家用电器

数字电视、高清晰度电视（HDTV）、高保真音响、玩具与游戏、数字电话等。

8. 医学工程

助听器、X-射线扫描、心电图/脑电图、病员监护、超声设备等。

9. 自动控制

磁盘/光盘伺服控制、机器人控制、发动机控制、引擎控制等。

10. 通信

纠错编译码、自适应均衡、回波抵消、同步、分集接收、数字调制解调、软件无线电、扩频通信等。

1.2 几种不同厂家的 DSP 芯片

面对 DSP 的巨大市场和广阔发展前景，世界上几个大的半导体公司都在 DSP 上开展竞争，如 AD、AT&T、Motorola、NEC、TI 等公司都在全力开发和生产 DSP 芯片。下面就简单介绍几种不同厂家的 DSP 芯片。

1.2.1 AD 公司

美国 AD 公司是有影响的通用 DSP 生产厂家之一。AD 公司的 DSP 芯片产品推出时间较晚，而综合性能较高。AD 公司的定点 DSP 芯片为 ADSP21XX 系列，浮点产品为 ADSP21020/ADSP2106X 系列，下面我们分别对其进行介绍。

一、ADSP21XX

ADSP21XX 为 AD 公司的定点 DSP 系列，其不同型号的存储器、外部接口各不相同。下面介绍 ADSP218X 的性能特点，以常用 40MHz 的 ADSP2183 为例，其主要性能为：

- 指令周期 25ns；
- 片内存储器分为两部分：程序 16K×24 bit，数据 16K×16 bit，共计 80KB；
- 片内寻址能力 16K；
- 片内设备：2 个串口，一个定时器；
- 运算单元字宽，乘法器 16 bit×16 bit 输入，32 bit 结果，累加器（ACC）40 bit，ALU 16 bit；
- 寻址方式：循环，位反序；
- DMA 2 个；
- 硬件堆栈 16 级，允许 7 级中断；
- 主机接口，13 个状态输入/输出管脚，EZ-ICE 仿真口（非 JTAG 标准）；
- 外部中断 4 个；
- 封装 128QFP。

ADSP21XX 系列主要面向通信系统等对处理数据精度和动态范围要求适中、更强调产品低成本和低功耗的应用领域。

ADSP218X 系列定点 DSP 相对于其它定点 DSP 的突出优点是片内高速存储器容量大、寻址能力强、运算速度快，对于需要较大存储器（40~80KB）的应用，ADSP218X 可以构成外围器件少的系统。

二、ADSP21020

ADSP21020 是 AD 公司推出的 32 bit 浮点 DSP 芯片，主要性能如下：

- 指令周期 30ns（33MHz 主频）；
- 相互独立两条总线，数据总线：32 bit 地址，40 bit 数据，程序总线：24 bit 地址，48 bit 数据；
- 片内无存储器；
- 指令 Cache：32×48 bit；
- 存储空间：4G 数据空间，16M 程序空间，读写均为单周期；
- IEEE 754 标准的 32 bit/40 bit 浮点格式；
- 32 bit/40 bit 的乘法器和 ALU，32 bit 移位器，可以乘、加、减并行工作，乘法器定点输入 32 bit，结果 80 bit；
- 32 个 40 bit 运算寄存器；
- 6 级可嵌套零开销循环；
- 延迟或条件跳转/调用/返回；
- 寻址：循环（8 个），位反序；
- 硬件堆栈 20 级可扩展；
- 1 个定时器；
- JTAG 仿真接口；
- 软/硬件等待状态；

- 外部中断 4 个；
- 加载方式：8 bit PROM，JTAG；
- 1024 点复数 FFT 0.64ms（基 2），0.58 ms（基 4）；
- 浮点倒数 180ns；
- 浮点平方根倒数 270ns；
- 封装 223PGA，223QFP。

三、ADSP2106X

AD 公司在 1995 年推出了并行浮点 DSP 芯片 ADSP21060，它在 ADSP21020 的处理器核上增加了片内 RAM 和链路口。它的突出特征表现为：大容量片内存储器，完全的片上总线控制逻辑可以将 6 片 DSP 直接相连，构成高效的共享存储器并行系统，在指令执行、跳转/调用、FFT 蝶形运算方面性能好。ADSP21060 的其主要性能为：

- 指令周期 25ns（40MHz 主频）；
- 片内有 4 条 32/48 bit 总线，供程序、数据、I/O、DMA 使用，片外一条 48 bit 总线；
- 片内 4M bit SRAM，可以灵活地配置成 16/32/48 bit 格式，用于数据/程序存储；
- 寻址 4G 空间，片外单周期读写；
- 指令 Cache 32×32 bit；
- IEEE 标准的 32/40 bit 浮点格式；
- 乘法器 32/40 bit 浮点输入，40 bit 结果，或 32 bit 定点输入，80 bit 结果；
- ALU 32/40 bit 浮点加减，32 bit 定点加减，允许同时求 2 个操作数的和/差；
- 32 bit 移位器；
- 单周期乘、加、减并行操作；
- 16 个 40 bit 运算寄存器；
- 6 级零开销块循环嵌套；
- 寻址：同时 8 个循环寻址，同时 2 个位反序寻址；
- 1 个定时器；
- 2 个串口；
- 软/硬件等待状态，1~7 周期软等待；
- 外部中断 3 个；
- JTAG 仿真接口，允许多片 DSP 仿真；
- 外部标志管脚 4 个；
- 加载方式：8 bit EPROM/16 bit 主机/链路口/48 bit 总线；
- 1024 点复数 FFT 0.46ms；
- 浮点倒数 150ns；
- 浮点平方根倒数 225ns；
- 6 个 4 bit 链路口（各带 2 个控制线），每个带宽 40MB/s；
- 10 个 DMA 通道，2 对 DMA 请求/应答信号；
- 主机接口；
- 封装 240QFP。

AD 公司同类的其它浮点并行 DSP 芯片包括 ADSP21062/61/65，它们是 ADSP21060 的简易低成本型，分别在片内存储容量以及链路口上作了简化和省略，ADSP21062/61/65 的片内存储容量依次减为 2 M bit、1 M bit、0.5 M bit，ADSP21061/65 省去了所有链路口。尽管如此，所有 ADSP2106X 保留了组成共享存储系统的片内总线仲裁功能。由于 ADSP2106X 采用了更先进的超级哈佛结构，因此也称为 SHARC 处理器。

1.2.2 AT&T 公司

AT&T 公司（现在的 Lucent 公司）是拥有高性能 DSP 芯片的另一家美国公司。其 DSP 芯片包括定点和浮点两大类。定点 DSP 芯片中有代表性的主要包括 DSP16 系列，浮点 DSP 芯片中比较有代表性的是 DSP32 系列。下面我们分别对其进行介绍。

一、DSP16 系列

AT&T 公司生产的定点 DSP 芯片 DSP16 系列已广泛应用于高速编解码器、数据中继、交换机、蜂窝电话等通信领域。此系统的基本型 DSP16A 主要性能为：

- 指令周期 25ns（40MHz 主频）；
- 片内存储器 12K×16 bit ROM，2K×16 bit 数据 RAM；
- 片外寻址空间 64K×16 bit；
- 乘法器 16 bit 输入，结果 32 bit；
- 累加器 36 bit；
- ALU 32 bit；
- 指令缓存 15×16 bit；
- 串口 1 个；
- 无 DMA；
- 循环寻址方式；
- 外部中断 1 个；
- 低功耗工作模式；
- 封装 84PLCC，133PGA。

DSP16A 功能简单，而陆续推出的 DSP16XX 系列其它型号在不同方面的性能有很大提高。DSP16C 片内集成了 A/D、D/A、JTAG 测试口，DSP1610、DSP1616 有片内仿真口，而且 DSP1616 可工作于 3.3V，后来推出的 DSP16XXX 系列则具有双乘法器和 3 输入加法器。

二、DSP32C

DSP32C 是 AT&T 公司推出的浮点 DSP 系列芯片，它是目前最简单的 32 bit 浮点 DSP，其结构并非采用流行的哈佛总线结构，而采用改进的冯·诺依曼结构，其主要性能为：

- 指令周期 80 ns（50 MHz 主频，4 个时钟 1 个周期）；
- 单条总线 4M 存储空间，可以用 8 / 16 / 32 bit 寻址；
- 片内存储器 3×512×32 bit；
- 非标准 32 bit 浮点格式；
- 一个串口，一个 8 / 16 bit 并口；

- 浮点乘法器：IEEE 浮点格式，输入 32 bit，结果 45 bit；
- 加法器 输入 / 输出 40 bit 浮点，或者 16 / 32 bit 定点输入；
- 4 个 40 bit 运算寄存器；
- 零开销循坏；
- 寻址位反序；
- 无硬件堆栈；
- 硬件等待状态；
- 外部中断 2 个；
- DMA 通道 2 个，仅用于串口，不能与处理器核同时访问；
- 1024 点复数 FFT 1.9 ms；
- 浮点除法 660 ns。

AT&T 公司在 DSP32C 的基础上又推出了 DSP3210 等增强型，在性能上有所提高，如 DSP3210 内部具有 2 个 1K 字的 RAM 块和 512 字的引导 ROM，外部寻址空间达 4G 字节，可以用软件编程产生等待状态，具有串行口、定时器、DMA 控制器等。

1.2.3 Motorola 公司

Motorola 公司是又一个有影响的 DSP 芯片产家。其 DSP 芯片主要可分为定点 DSP 和浮点 DSP 两类，其中定点 DSP 芯片以 MC56000 系列为代表，浮点 DSP 以 MC96002 为代表。下面我们分别对其进行介绍。

一、DSP56000 系列

Motorola 公司的 DSP56K 系列定点 DSP 在功能上与 TI、AD 公司的定点 DSP 相似，它的一个显著特点是数据字宽 24 bit，乘法器 48 bit，加法器 56 bit，因此能提供较大的动态范围和更高的处理精度。其名称 56K 与加法器具有 56 bit 有关，下面我们以 66 MHz 的 DSP56002 为例，其主要性能为：

- 指令周期：30.3 ns（66MHz 主频，2 个时钟一个周期）；
- 片内存储：512×24 bit 程序 RAM，64×24 bit 引导 ROM；
  两个 256×24 bit 数据 RAM；
  两个 256×24 bit 数据 ROM，固化有正弦表、A 律和 μ 律压扩表；
- 片外寻址能力：3 个 64K×24 bit 空间；
- 1 个串口，1 个 8 bit DMA 并口，1 个定时器；
- 乘法器输入 24 bit，结果 48 bit，2 个 56 bit 加法器，56 bit ALU，具备多精度运算能力，完成 48 bit×48 bit→96 bit 乘法仅需 6 个指令周期；
- 寻址方式有循环及位反序；
- 系统堆栈 15 级；
- 外部中断 3 个；
- 1024 点复数 FFT 时间 1.8 ms，硬件支持 FFT 浮点算法；
- 支持 32 路 TDMA 通信；
- 三种加载方式：外总线、主机接口、串口；

- 封装 132QFP；

DSP56002 指令周期为两个时钟周期，新推出的 DSP56301 在性能上有了很大提高,可在一个时钟内执行一条指令，各项主要性能提高如下：

- 指令周期 10ns（100MHz 主频），单周期内可同时执行 3 条指令和 6 个 DMA 通道操作；
- 片内存储 4K×24 bit 程序 RAM，4K×24 bit 数据 RAM，程序 RAM 中可将 1K×24 bit 设置为指令缓存，另有 192×24 bit 引导 ROM，4K 数据 RAM 分成两个 2K 区；
- 片外存储 16M×24 bit 的程序区，两个 16M×24 bit 数据区；
- 通过软件可以设置成 16 bit 数据运算形式；
- 2 个 10Mb/s 的同步串口；
- 1 个可与 Codec 和 I/O 接口的 10Mb/s 串口；
- 3 个定时器；
- 6 个 DMA 通道；
- 片内集成 32 bit PCI 总线控制器，其传输速度可达 133Mb/s；
- 可以直接与 DRAM 接口，仅需缓冲器就可以连到 ISA 总线上；
- 具有 JTAG 测试口和 OnCE 口，OnCE 口用于仿真程序和支持开发系统的调试。

此外，Motorola 公司为移动手机提供了低成本、低功耗（1.8V 工作电压）的 DSP56600 系列，其数据字宽减少为 16 bit，DSP56652 内部集成了一片 16 bit 的 DSP56600 和一个 32 bit 的 RISC 处理器，专门用于移动通信基带处理的 IS—136 标准的 TDMA 应用，完成信号收、发处理和频率合成，指令执行速度 60MIPS（16.7 ns），运算单元包括了一个 16 bit×16 bit 乘法器和 2 个 40 bit 累加器。DSP56652 是专用 ROM 型的，相对应的 DSP56651 则是基于 RAM，用于程序设计阶段的编程调试。

DSP56800 系列则是更低成本、低功耗的 16 bit 定点 DSP，结构功能上做了简化设计，可用于通信、电机控制领域。DSP56100 也是 16 bit 定点 DSP。

二、DSP96002

DSP96K 系列 DSP 是 Motorola 公司推出的高性能 32 bit 浮点 DSP 芯片。与 DSP56K 系列一样，DSP96K 名称与其 96 bit 加法器有关，采用哈佛结构，其主要性能如下：

- 指令周期 50ns（40MHz 主频）；
- 片内总线：3 个 32 bit 地址总线，5 个 32 bit 数据总线，片外两条总线；
- 片内存储器：1K×32 bit 程序 RAM，2×512×32 bit 数据 RAM；
- 寻址空间 3×8×0.5G；
- 2 个主机接口；
- IEEE 754 标准的 32 bit 浮点格式；
- 乘法器输入 32 bit 浮点，结果 44 bit ，或输入 32 bit 定点，输出 64 bit 结果；
- 加法器对 10 个 96 bit 寄存器或 30 个 32 bit 寄存器操作；
- 32 bit 移位器
- 零开销循环；
- 15 层堆栈；
- 软/硬件等待状态；

- 在线仿真口；
- 外部中断 3 个；
- 1024 点复数 FFT 1.047ms，1024 点实数 FFT 0.65ms；
- 浮点除法 300ns；
- 功耗 2W；
- 封装：223PGA，256QFP。

1.2.4 NEC 公司

NEC 公司也是一家有影响的 DSP 芯片厂家。其生产的 DSP 芯片以定点 DSP 为主，包括μPD77CXX 系列、μPD770XX 系列和μPD772XX 系列，下面我们分别对其进行介绍。

一、 μPD77C25

μPD77C25 是 NEC 在 D7720 之后的第二个 16-bit 定点 DSP 芯片，它是一种 CMOS 器件，但其管脚及芯片代码与老芯片兼容。μPD77C25 有 2K×24 bit 指令字、512 字数据 RAM 和 2K 数据 ROM。程序和数据 ROM 同样也可以用作 EPROM 或 OTP 存储器。其主要性能如下：

- 2 个 16 bit 加法器 ；
- 2K×24 bit 程序 ROM/EPROM/OTP ；
- 2K 数据 ROM/EPROM/OTP ；
- 0.5K 数据 RAM ；
- 16 bit ALU/加法器 ；
- 2 条内部数据总线 ；
- 4 电平 硬件存储器；
- 2 个串行口 ；
- 2 个 I/O 引脚, 1 个外部中断 ；
- 8/10 MHz 时钟 ；
- 4 MHz 串行 I/O 。

μPD77C25 可以在单机模式下或与主机处理器协同运行。它有一个 8 bit 并行 I/O 主机口用来与主机处理器交换数据或 DSP 状况。μPD77C25 有两个串行口，它能够与串行外围设备如 A/D、D/A 和其他μPD77C25 DSP 接口。可以将串行口设置成对单或双波特传输方式。

二、NEC μPD77017 16 bit 定点 DSP

NEC 推出的μPD77017 16 bit 定点 DSP 芯片是以数字蜂窝、 语音、传真和调制解调器应用为目标。其核心运算单元包括三个 40 bit 功能单元——乘法器、ALU 和管状移位器。其主要性能如下：

- 33/16.5/8.25/4.125 MHz 时钟 (2 时钟脉冲/周期) ；
- 3V 工作电压；
- 32 bit 指令，16 bit 数据 ；
- 3 级 管道，81 条指令 ；

- 8 个 40 bit 寄存器/加法器；
- 2 个 2K 字数据 RAM 块 ；
- 12K × 32 bit 程序 ROM ；
- 256 × 32 bit 程序 RAM ；
- 3 条内部总线 ；
- 1 个外部存贮器总线 ；
- 15 电平硬件堆栈 ；
- 2 个串行口 ；
- 4 个外部中断；
- JTAG 仿真接口；
- 嵌入式 PLL 。

μPD77017 有两个外部存储器口，一个用于 16 bit 数据，另一个用于 32 bit 程序。存储器读 / 写可在同一个周期内完成。与 NEC 公司的很多 DSP 芯片一样， μPD77017 有一个 8 bit 主机 I/O 口，可以通过该 I/O 口与主机之间交换数据。

三、 μPD77220

NEC 推出 24 bit 定点 DSP μPD77220 处于 NEC 的 16 和 32 bit 结构之间，它是 NEC 公司 32 bit DSP 芯片 μPD77240 的简化版本。其主要性能如下：

- 8/10 MHz 时钟 (2 时钟脉冲/周期) ；
- 32 bit 指令, 24 bit 数据 ；
- 3 状态管道, 51 条指令 ；
- 8 个 47 bit 累加器 ；
- 2K× 24 bit 程序 ROM/EPROM ；
- 2 个 768 字 数据 RAM 块；
- 3K 字 数据 ROM 块 ；
- 16 bit 主机口 ；
- 外部存储器接口：13 bit 地址, 32 bit 数据；
- 4 个 I/O 管脚, 2 个外部中断 ；
- 所有指令都是 32 bit 字 ；
- 10 bit 循环控制 ；
- 8 电平硬件存贮器 。

μPD77220 结构比较复杂。其指令设置和代码类似于 16 bit 的 DSP 芯片，而基于寄存器的体系结构则与其后的 32 bit DSP 芯片相类似。

1.2.5 TI 公司

TI 公司是最有影响力的 DSP 芯片生产产家之一，其产品用 TMS320 系列表示，其中 TMS320C1X/C2X/C5X/C2XX/C54X/C62X 为定点 DSP，TMS320C3X/C4X/C67X 为浮点 DSP，TMS320C8X 针对多媒体应用的图像、视听数字处理领域，其应用正在被新推出的 TMS320C6XX 所取代，TMS320C1X/C2X/C5X 是系列定点产品，保持了指令的兼容性，

目前普遍使用 C5X 系列。TMS 系列种类很多，覆盖范围广泛，下面我们分别对其进行介绍。

一、TMS320C1X 系列 16 bit 定点 DSP

TMS320C1X 系列是 TI 公司于 1982 年推出的 16 bit 定点 DSP 芯片，我们以 C14 为例，其主要性能如下：

- 16 bit 指令、数据
- 59 个指令，3 级管道
- 2×16 bit 寄存器、32 bit 加法器
- 512 字节 RAM
- 4K 字节 ROM/EPROM/OTP
- 2×16 bit 内部总线
- 4K 字节地址空间
- 16 bit 移位器
- 3 个捕捉/比较定时器
- 1 个串口，16 个 I/O 管脚
- 2 个外部中断
- 20/25/35 MHz 时钟
- 4 周期指令；114 ns（35 MHz）
- 组合移位 / ALU 周期
- 单周期乘法器
- 8 周期 MAC 指令
- 存储器读 / 写操作指令 4 周期
- 看门狗定时器
- 中断应答最多 7 个周期
- 4 级指令堆栈

虽然 C1X 系列有一些局限性，如：CPU 基于加法器，并且其结构依赖于共享资源（如总线和存储器），而不是用乘法来加快处理，但这是许多早期芯片都具有的。改进的哈佛结构将程序和数据分开存取，同时为了共享资源，它还允许代码和数据空间之间的转移。芯片上的存储器很小，片内寻址范围只有 4K 字节。

二、TMS320C2X 系列 16-bit 定点 DSP

TI 公司于 1986 年推出的 TMS320C2X 是 TI 的第二代 16 bit 定点 DSP。C2X 继承了 C1X 基于加法器的结构，但增强了处理器的性能，并使得芯片更易于编程和设计。以 TMS320C28 为例，其主要功能如下：

- 133 条指令，3 级管道
- 8 个辅助寄存器
- 2 个 544 字节数据 RAM 块
- 64 字节数据 RAM
- 16K 字节程序 ROM
- 2×64K 字地址空间

- 16 bit 外部总线
- 外部总线通过 DMA 对内部寄存器进行读写
- 16 bit 定时器，3 个外部中断
- 50MHz 时钟（周期=时钟/4）
- 单周期指令
- 单周期的 MPY，MAC
- 支持数据块移动
- 零开销的指令重复
- 位反序寻址
- 硬件等待状态产生器
- 并行 MPY、ALU 操作
- 空闲模式下进入省电模式
- 中断响应时间最多 8 周期

C2X 增加了 24 条新指令，包括循环指令（对于 MAC 操作有自动数据地址增加功能）和扩展的片内 RAM 、2×512 字节数据块。程序 ROM 是 16K 字节，且 DMA 简化了外部对内部存储器的读写。带有 8 个辅助寄存器的 16 bit ALU 与 MAC 并行操作。TMS320C2X 的源代码与以前的 16-bit 定点 C1X 芯片兼容。

三、TMS320C3X

TMS320C3X 采用改进的哈佛结构，其主要性能如下：

- 指令周期 33 ns（60MHz）, 60 MFLOPS，33MIPS；
- 主总线 24 bit 地址，32 bit 数据／程序，扩展总线 14 bit 地址，32 bit 地址，32 bit 数据／程序；
- 64×32 bit 指令 Cache；
- 16M 片外存储空间：数据程序混放，读单周期，写双周期；
- 片内 2K×32 bit 双口 RAM，可分为两组分别访问；
- 非标准 32 bit／40 bit 浮点格式；
- 32／40 bit 浮点乘法器及 ALU，32 bit 移位器；
- 并行乘／累加操作；
- 8 个 40 bit 数据寄存器，8 个 32 bit 辅助（寻址）寄存器；
- 片内 DMA 控制器；
- 寻址：循环、位反序；
- 单指令循环、程序块循环；
- 条件调用／返回；
- 互锁操作；
- 2 个串口；
- 2 个定时器；
- 加载方式 32 bit；
- TI 仿真接口（非 JTAG）；
- 软／硬件等待状态；

- 外部中断 4 个；
- 1024 点复数 FFT：1.67 ms；
- 浮点求倒数 1155 ns；
- 浮点求平方根倒数 1287 ns；
- 封装 181 PGA。

TMS320C31 是 C30 的简易型，区别在于 C31 没有扩展总线，仅有一个串口，QFP132 封装，可以用多种模式（8 / 16 / 32 bit / 串口）加载且可重定位中断矢量表，而 C30 必须用 32 bit 存储器从 0 地址装入初始化程序代码。

TMS320C32 在 C31 的基础上对结构进一步简化，即将片内 RAM 从 2K×32 bit 减少为 512×32 bit，同样分成两个 256 字存储块，也具有像 C31 一样的多模式程序加载方法。

为了与 AD 公司的大容量片内 RAM DSP 相抗衡，TI 公司后来又发表了 TMS320VC33 规划书，VC33 采用高达 120 MHz 或 150 MHz 的主频，有 120 / 150MFLOPS 的峰值运算能力，片内 1 Mbit RAM，程序代码与先前的 C3X 完全兼容，VC33 本身结构功能也与 C31 兼容，而且采用了 3.3V I/O 和 1.8V 处理器核使功耗降低到 220 mW，而 C30 / C31/C32 的功耗在 1.5~3W 之间。

TMS320C3X 可以采用与浮点乘 / 加相同的速度完成 32 bit 定点乘 / 加，要注意的是 32 bit 定点乘限制输入数据 16 bit（C32）或 24 bit（C30），结果取 32 bit。

四、TMS320C4X

TI 公司于 1991 年推出了 40MHz 主频的并行浮点 DSP 芯片 TMS320C40，以后发展了 50 MHz、60 MHz 的 TMS320C40/C44 产品。C40 的主要性能如下：

- 指令周期为 33ns（60 MHz）；
- 片内 3 条 32 bit 总线，供程序、数据和 DMA 使用，片内 2 个 1K×32 bit 存储器，每周期内分别可以存取 2 次；
- 片外两条独立 32 bit 总线，片外寻址空间 2×2G，零等待访问时为单周期读或双周期写；
- 指令 Cache 128×32 bit；
- 非标准浮点格式 32/40 bit；
- 乘法器 32 bit 浮点输入，40 bit 结果，或 32 bit 定点输入，32 bit 结果；
- ALU 工作于 40 bit 浮点或 32 bit 定点；
- 32 bit 移位器
- 单周期乘、累加并行指令；
- 12 个 40 bit 运算寄存器；
- 零开销单指令或块循环；
- 迟延、条件跳转/调用；
- 寻址：循环，位反序；
- 2 个定时器；
- 软/硬件等待状态（1~7 周期软等待）；
- 外部中断 4 个；
- JTAG 仿真接口，允许多 DSP JTAG 仿真；

- 加载方式：8/16/32 bit 或通信口；
- 1024 点 FFT 1.3ms；
- 浮点倒数 264ns；
- 浮点平方根倒数 363ns；
- 6 个 8 bit 通信口（各带 4 个控制信号线），每个通信口速度为 20MB/s；
- DMA 控制器可控制 6 个 DMA 通道；
- 封装 325PGA。

C44 作为 C40 的简易型，通信口减少到 4 个，寻址空间减少到 16M。

五、TMS320C5X

TMS320C5X 与 TMS320C1X / C2X 源代码兼容，有多个品种，下面以较新的 80 MHz TMS320C50 为例作一介绍。其性能如下：

- 指令周期 25 ns；
- 16 bit 定点处理器，数据 16 bit，指令 16 bit，累加器 32 bit，ALU 32 bit ，16 bit×16 bit 乘法器；
- 寻址空间 64K×16 bit；
- 单周期乘 / 累加器；
- 224K 外部寻址空间（64K 程序，64K 数据，64K I/O，32K 全局存储器）；
- 8 个辅助 / 寻址寄存器，11 个上下文切换寄存器；
- 8 级硬堆栈；
- 两个循环寻址缓冲，一个位反序寻址；
- 可单指令重复、块指令重复操作；
- 程序 / 数据块传送指令；
- 全双工异步通信口；
- TDM（时分多路访问）串口；
- 1 个定时器；
- 无 DMA；
- 4 级流水；
- 迟延跳转 / 调用 / 返回；
- 有总线请求 HOLD 和 BR 信号支持多片并行工作，并可互相访问片内 RAM；
- 132QFP 封装。

六、TMS320C6X

TMS320C6X 是 TI 公司推出的一种高性能并行 DSP 芯片。下面我们分别以 TMS320C6201 和 TMS320C6701 为例，对其进行介绍。

1. TMS320C6201

TMS320C6201 是主频 200MHz 的定点 DSP，8×32 bit=256 bit 的超长指令字（VLIW）使其每周期能命令 8 个处理单元执行 8 条指令，即 1600MIPS 或 400MMAC（MMAC 为每秒百万次乘加），其主要性能如下：

- 8 个处理单元包括 6 个 32 bit ALU 和 2 个 16 bit 乘法器；
- 片内 1Mbit SRAM，可装 512K bit 指令和 512K bit 数据；

- 指令 Cache 16K×32 bit =2K×256 bit;
- 32 位外部总线，4G 寻址空间，可按 8/16/32 bit 寻址；
- 32 个 32 bit 运算寄存器；
- 2 个定时器；
- 2 个串口；
- 1 个 16 位主机接口；
- JTAG 仿真口；
- 外部中断 4 个；
- 4 个 DMA 通道；
- 1024 点复数据 FFT：70 μs（16 bit，基 4），704 μs（32 bit，基 4）；
- 封装 352BGA。

TMS320C6201 虽然是 16 bit 定点 DSP，但也可以进行 32 bit 数据格式的运算，因为其乘法器是 16 bit，因此作 32 bit 运算速度要下降很多。

2. TMS320C6701

在 1998 年年底才推出样片的 TMS320C6701 是一种浮点超高速 DSP。它与 TMS320C6201 管脚兼容，主频 167MHz，也有 8 个运算单元，其中 6 个为浮点单元，同样采用 8×32 bit =256 bit 的超长指令字，峰值运算能力为 1 GFLOPS 或 1336 MIPS。结构上与 C6201 类似，不同之处有：

- 4 个浮点/定点 ALU；
- 2 个定点 ALU；
- 2 个浮点/定点乘法器，乘法器接收 32 bit 定点数，产生 64 bit 结果；
- 支持 32/64 bit 的 IEEE 浮点格式。

其峰值运算能力为：

- 1GFLOPS（32 bit 单精度）；
- 256MFLOPS（64 bit 双精度）；
- 乘加并行操作 688MFLOPS。

七、TMS320C8X

TMS320C80 是 TI 在 1995 年推 出的一种针对图像、视频处理的高性能 DSP，包括内部 1 个 32 bit RISC 主处理器和 4 个 32 位定点处理器，单周期指令，因此总处理速度为 250MIPS（50MHz），其主要性能如下；

- 片内 2×25K×8 bit 存储器，支持 8/16/32/64 bit 格式，每周期可接受 15 次访问；
- RISC 主处理器，支持 32 bit IEEE 浮点格式，有 32/64 浮点单元，运算速度 100MFLOPS，带有 4KB 指令 Cache；
- 每个 32 位定点处理器带有 16 bit 乘法器和 32 bit 累加器，2KB 指令 Cache；
- 4GB 存储空间，具有 8/16/32/64 bit 动态总线宽度；
- 4 个外部中断；
- JTAG 仿真口；
- 封装 305PGA。

TMS320C82 是 C80 的低成本型，减少了 2 个 32 位定点处理器，片内存储器为 44K×8

bit ，采用 240QFP 封装。

1.3 DSP 系统设计概要

1.3.1 典型的 DSP 系统构成

典型的 DSP 系统构成如图 1-1 所示。其中输入信号可以是语音信号、传真信号，也可以是视频信号，还可以是传感器（如温度传感器）的输出信号。输入信号经过带限滤波后，通过 A/D 变换将模拟信号转换成数字信号。根据奈奎斯特采样定理，采样频率至少是输入带限信号最高频率的 2 倍，在实际应用中，一般为 4 倍以上。数字信号处理一般是用 DSP 芯片和在其上运行的实时处理软件对 A/D 转换后的数字信号按照一定的算法进行处理，然后将处理后的信号输出给 D/A 转换器，经 D/A 转换、内插和平滑滤波得到连续的模拟信号。当然，并非所有的 DSP 系统都具有图 1-1 所示的所有部件，例如，频谱分析仪中输出的不是连续波形而是离散波形，而 CD 唱机中的输入信号本身就是数字信号，等等。

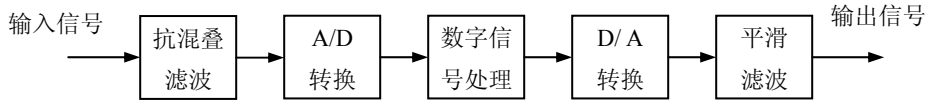


图 1-1 典型的 DSP 系统构成

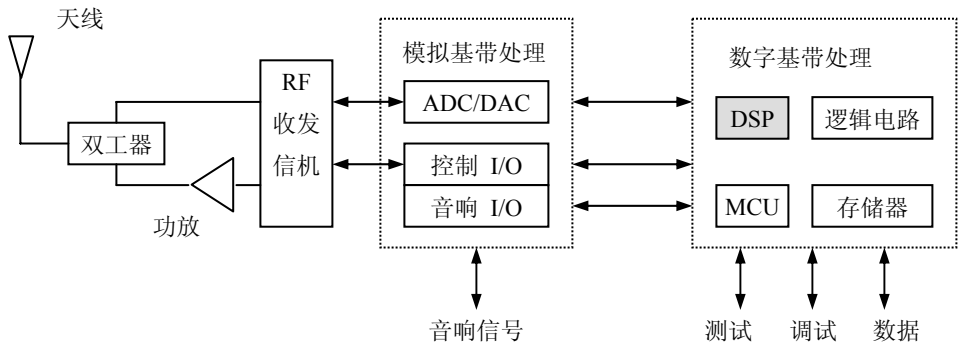


图 1-2 一种典型移动通信终端的原理框图

DSP 系统可能有一个 DSP 芯片及外围电路组成，也可能由多个 DSP 芯片及外围电路组成，这完全取决于处理的要求。对于无线通信，其信号处理一般包括信源编/解码、信道编/解码、交织/解交织、加密/解密、调制/解调、均衡、分集接收等。就一个终端设备而言，往往要完成由应用层、网络层、数据链路层和物理层组成的通信协议处理，其中物理层主

要完成无线通信中的信号处理。图 1-2 给出了一种移动通信终端的原理框图，其中，RF 收发信机负责无线信号的收发；模拟基带处理负责 A/D 和 D/A 转换及控制接口等；数字基带处理完成通信协议处理。由图 1-2 可见，数字基带处理包括 DSP 芯片、微处理器（MCU）、存储器和硬件逻辑等部分，它们所完成的任务不同，MCU 用于系统控制，侧重于应用层、网络层、数据链路层的处理；DSP 用于完成物理层的处理，倾向于数字基带信号处理，它包括通用的信号处理（如 FIR 滤波、FFT 等）和移动通信信号处理（如 CRC 校验、纠错编码、数据调制、分集接收、同步、均衡等）。图 1-3 给出了一种基于多个 DSP 芯片的软件无线电台硬件结构图，它采用支持多处理器的 VME 总线，使用了 4 片 DSP 芯片，能够很好地满足软件无线电的开放性的要求，已在美军的三军通用软件无线电台研究开发项目中使用。

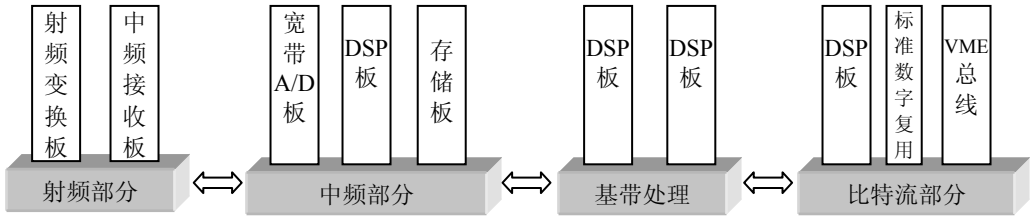


图 1-3 基于 VME 总线的软件无线电台的硬件结构框图

不同的无线通信系统的 DSP 实现结构可能与图 1-2 和图 1-3 所示的结构有所不同，但一般都含有许多类似的功能模块，在本书的后续章节将详细介绍。

1.3.2 DSP 系统设计过程

与其它系统设计工作一样，在进行 DSP 系统设计之前，设计者首先要明确自己所设计的系统用于什么目的，应具有什么样的技术指标。对于一个实际的 DSP 系统，设计者应考虑的技术指标主要包括：

① 由信号的频率范围确定系统的最高采样频率；
② 由采样频率及所要进行的最复杂算法所需的最大时间来判断系统能否实时工作；
③ 由以上因素确定何种类型的 DSP 芯片的指令周期可满足需求；
④ 由数据量的大小确定所使用的片内 RAM 及需要扩展的 RAM 的大小；
⑤ 由系统所需要的精度确定是采用定点运算还是浮点运算；
⑥ 根据系统是计算用还是控制用来确定输入输出端口的需求。在一些特殊的控制场合有一些专门的芯片可供选用，如 TMS320C2XX 系列自身带有 2 路 A/D 输入、6 路 PWM 输出及强大的人机接口，特别适合于电机控制场合。

由以上因素可大体上确定应该选有的 DSP 芯片的型号。根据选用的 DSP 芯片及上述技术指标，还可以初步确定 A/D、D/A、RAM 的性能指标及可供选择的产品。当然，在产

品选型时，还须考虑成本、供货能力、技术支持、开发系统、体积、功耗、工作环境温度等。

具体进行 DSP 系统设计时，其一般设计流程图如图 1-4 所示，设计步骤可大致分为如下几个阶段：

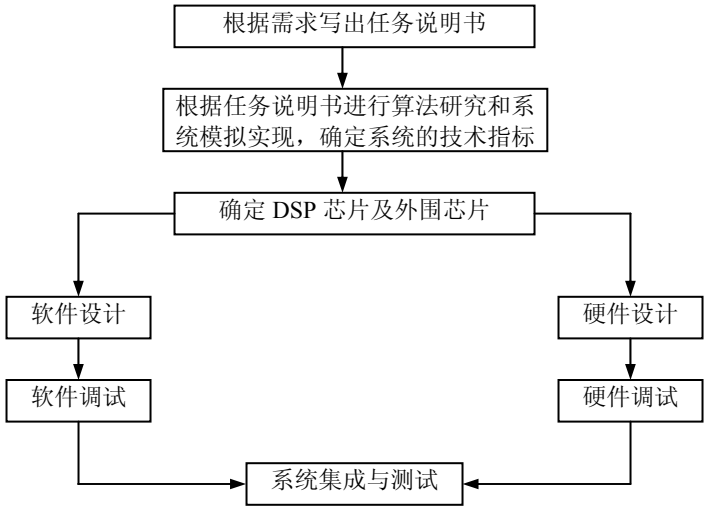


图 1-4 DSP 系统设计流程图

（1）算法模拟阶段。

在这一阶段主要是根据设计任务确定系统的技术指标。首先应根据系统需求进行算法仿真和高级语言（如 MATLAB）模拟实现，以确定最佳算法，并初步确定相应的参数。

（2）DSP 芯片及外围芯片的确定阶段

根据算法的运算速度、运算精度和存储要求等参数选择 DSP 芯片及外围芯片。

（3）软硬件设计阶段

首先按照选定的算法和 DSP 芯片，对系统的哪些功能用软件实现，哪些功能用硬件实现进行初步分工，如 FFT、数字上/下变频器、RAKE 分集接收是否需要专门芯片或 FPGA 芯片实现等，译码判决算法是用软件判决还是硬件判决，等等。然后，根据系统技术指标要求着手进行硬件设计，完成 DSP 芯片外围电路和其它电路（如转换、控制、存储、输出、输入等电路）的设计；根据系统技术指标要求和所确定的硬件编写相应的 DSP 汇编程序，完成软件设计。当然，软件设计时也可采用高级语言进行，如 TI 公司提供了最佳的 ANSI C 语言编译软件，该编译器可将 C 语言编写的信号处理软件变换成 TMS320 系列的汇编语言。

（4）硬件和软件调试阶段

硬件调试一般采用硬件仿真器进行。软件调试一般借助 DSP 开发工具如软件模拟器、DSP 开发系统或仿真器进行。通过比较在 DSP 上执行的实时程序和模拟程序执行情况来判断软件设计是否正确。

（5）系统集成和测试阶段

硬件和软件调试分别调试完成后，将软件脱离开发系统，装入所设计的系统，形成所谓的样机，并在实际系统中运行，以评估样机是否达到了所要求的技术指标。若系统测试符合指标，则样机的设计完毕。但这种情况并不常见，实际上由于软硬件调试阶段的环境是模拟的，因此在系统测试中往往可能会出现诸如精度、稳定性不好等问题。当出现这类问题时，一般采用修改软件加以解决，若软件修改仍无法解决，则必须调整硬件，此时的问题可能就比较严重了。

第二章 TMS320C5000 系列

2.1 引言

TI 公司的 TMS320 系列 DSP 芯片的发展经历了 C1x、C24x、C28x、C3x、C4x、C54x、C55x、C62x、C64x、C67x 和 C8x 等，其中的 C3x、C4x、C67x 属于浮点 DSP，C8x 为多处理结构，余下为定点 DSP，为方便起见，通常将 C24x、C28x 称为 C2000 系列，此类芯片主要用于数字控制系统；而将 C54x 和 C55x 称为 C5000 系列，C5000 系列芯片因其具有强大的运算能力，低功耗等特点，主要用于功耗低、便于携带的无线通信终端产品；将 C62x、C64x、C67x 等称为 C6000 系列，主要用于高性能的复杂的通信系统，如移动通信基站。

C5000 系列的 DSP 芯片在无线通信终端和其它的通信主品中应用广泛，其中 C54x 最为成熟，它集成了丰富的硬件逻辑和外部接口资源，在提高性能的同时，降低了成本和体积，已经获得了广泛的应用。C55x 是在 C54x 的基础上发展起来的，是目前 TMS320 系列中功耗最低的主品，在移动通信的 2.5G 和 3G 移动通信终端产品中将有广泛的应用。

本章将主要以 C54x 芯片为主介绍 C5000 系列芯片，并在 C54x 的基础上介绍 C55x。

TMS320C5000 系列芯片在通信、自动控制、医疗设备、精密仪器等方面得到广泛的应用，其主要的应用场合为：

- 数字蜂窝移动通信，如低功耗的手机；
- 个人通信系统（PCS）；
- 高速数字寻呼机；
- 个人数字助理（PDA）；
- 数字无绳电话；
- 无线数据通信；
- 医用精密仪器等。

2.2 TMS320C54x 的基本结构

TMS320C54x 系列芯片同其它 TI 公司 DSP 定点数字信号处理器一样，为典型的改进型哈佛结构，图 2-1 为其结构图。

TMS320C54x 芯片的主要特点：

- 操作性能高达 100MIPS（每秒执行百万条指令），其中 TMS320VC5416 高达 160MIPS；
- 25/20/15/12.5/10/6.25ns 机器指令周期，

- 整合维特比操作；
- 备有三个掉电模式；
- 整合随机存储器及只读存储器配置；
- 自动缓冲串口；
- 主端口接口；
- 超薄包装（100 脚、128 脚、144 脚 TQFP 及 144 脚 BGA 包装）。
- 8V、3.3V 及 5V 操作；
- 40 位加法器及两个 40 位加速器，用于支持并行指令；
- 40 位算术逻辑部件连双 16 位配置性能，用于双重单周期操作；
- 17_17 乘法器，允许 16 位带符号或不带符号的乘法；
- 四个内部总线及双地址生成器，进行多重操作数运算，并降低存储器瓶颈现象；
- 单周期正规化及指数译码；
- 八个辅助寄存器及一个软件栈，允许使用定点 DSP C 语言译器；
- 掉电模式，宜于电池供电应用。

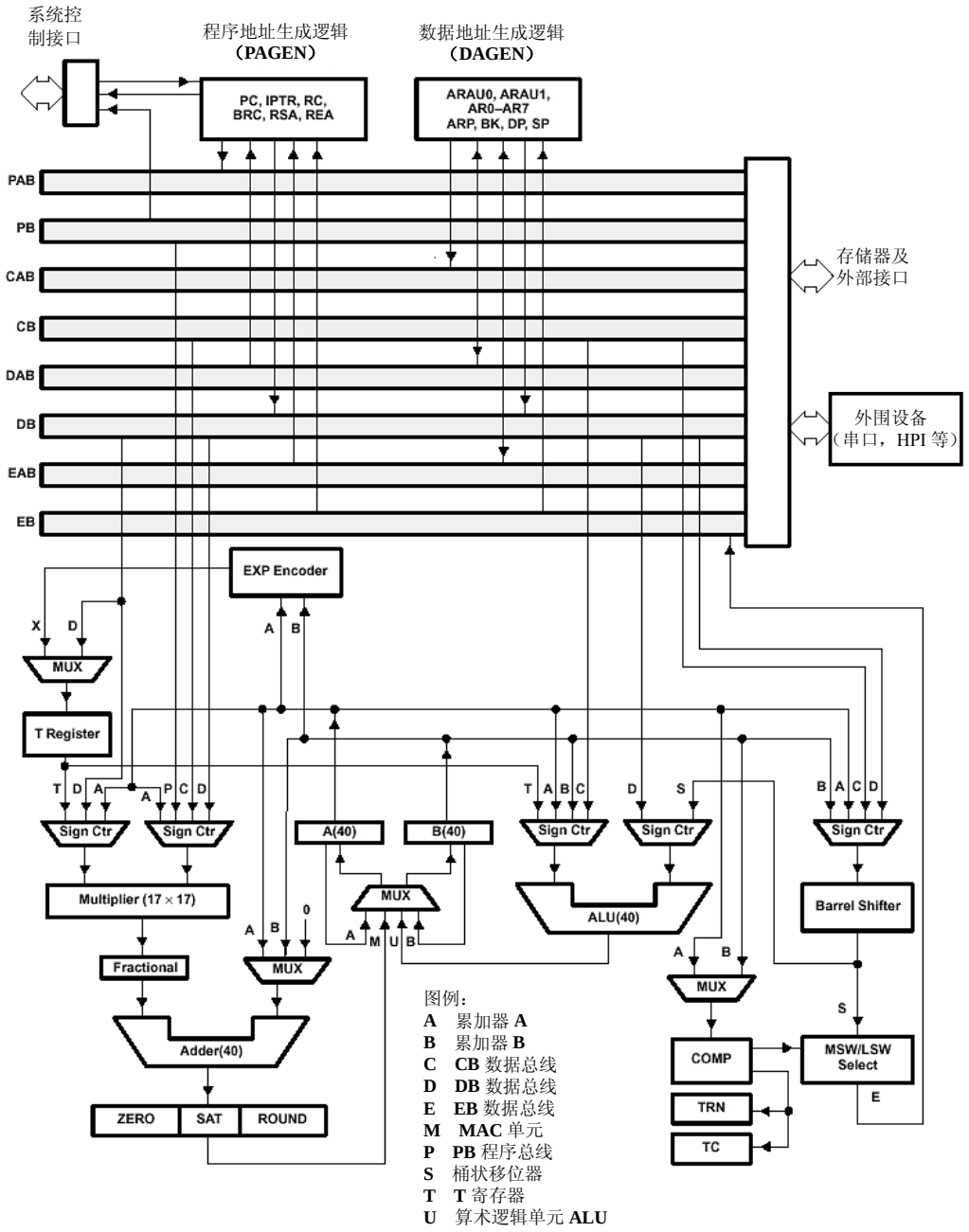


图 2-1 TMS320C54x 的结构图

2.2.1 TMS320C54x 的总线结构

C54x 的总线结构由八组 16-bit 总线（四组程序/数据总线，四组地址总线）构成的。

- 程序总线（PB）传送从程序存储器来的指令代码和立即数。
- 三组数据总线（CB、DB 及 EB），CB 和 DB 总线传送从数据存储器读出的操作

数，EB 总线传送写入到存储器中的数据。

- 四组数据总线（PAB、CAB、DAB 和 EAB）传送执行指令所需的地址。

2.2.2 中央处理单元

C54x 芯片的 CPU 包括：

- 一个 40-bit 的算术逻辑单元（ALU）。

TMS320C54x 使用了 40-bit 的算术逻辑单元（ALU）和两个 40-bit 的累加器（ACCA 和 ACCB）来完成二进制补码的运算，同时 ALU 可完成布尔运算。在状态寄存器 ST1 中的 C16 位置 1 时，可同时完成两个 16-bit 的运算。

- 两个 40-bit 的累加器（ACCA 和 ACCB）

如图 2-2 所示，累加器 ACCA 和 ACCB 存放从 ALU 或乘法器/加法器单元输出的数据，；累加器也能输出到 ALU 或乘法器/加法器中，累加器可分为三个部分：

1) 保护位（bit39-32）
2) 高位字（bit31-16）
3) 低位字（bit15-0）

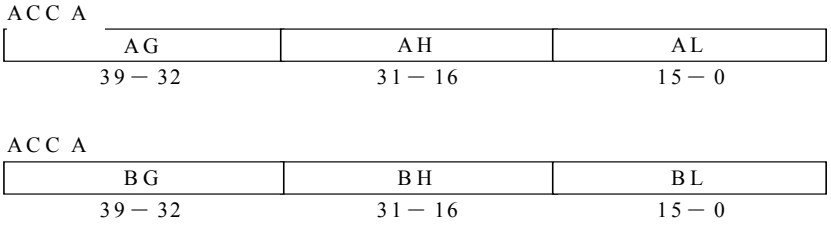


图 2-2 累加器

保护位用来作为计算的前部留空，防止在迭代运算中产生溢出，

- 一个桶形移位器

C54x 的桶形移位器有一个与累加器或数据存储器（CB、DB）相连接的 40-bit 输入，和一个与 ALU 或数据存储器（EB）相连接的 40-bit 输出，能把输入的数据进行 0-31bit 的左移和 0-16bit 右移，所移位数由 ST1 中的移位数域（ASM）或被指定作为移位寄存器的暂存器（TREG）决定。桶形移位器和指数译码器可把累加器中的值在一个周期中进行归一化。移位输出的最低位填 0；最高位可以填 0 或进行符号扩展，由 ST1 中的符合扩展方式位（SXM）决定。

- 17_17-bit 乘法器，40bit 加法器

乘法器/加法器与一个 40-bit 的累加器在一个单指令周期内完成 17_17-bit 的二进制补码运算，乘法器/加法器由乘法器、加法器、带符号/列符号输入控制、小数控制、零检测器、舍入器、溢出/饱逻辑、暂存器（TREG）组成，乘法器有两个输入：一个从 TREG，数据存储器操作数，或一个累加器中选择；另一个则从程序存储器。数据存储器，一个累加器中选择。另外，乘法器和 ALU 在一个指令周期里共同执行乘/累加（MAC）运算且并

行 ALU 运算。

- 比较、选择和存储单元（CSSU）

该单元完成累加器的高位字和低位字之间的最大值比较，即选择累加器中较大的字并存储在数据存储器中，不改变状态寄存器 ST0 中的测试/控制位和传送寄存器（TRN）的值，同时 CSSU 利用优化的片内硬件促进 Viterbi 型蝶形运算。

- 指数编码器

指数编码器是用于支持单周期指令 EXP 的专用硬件，在 EXP 指令中，累加器的指数值能以二进制补码的形式存储于 T 寄存器中，范围为 bit8-31。指数定义为前面的冗余位数减 8 的差值，即累加器中消除非有效符号位所需移动的位数，当累加器中的值超过 32bits 时将产生负值。

- 各种 CPU 寄存器（CPU 寄存器是存储器映射的，能快速恢复和保存）

C54x 有三个状态和控制寄存器，分别为：状态寄存器 ST0 和 ST1，处理器方式状态寄存器 PMST，其中 ST0 和 ST1 包括了各种条件和方式的状态，PMST 包括了存储器配置状态和控制信息。图 2-3 为 ST0、ST1 和 PMST 的详细内容。

ST0

15－13	12	11	10	9	8－0
ARP	TC	C	OVA	OVB	DP

ST1

15	14	13	12	11	10	9	8	7	6	5	4－0
BRAF	CPL	XF	HM	INTM	0	OVM	SXM	C16	FRCT	CMPT	ASM

PMST

15－7	6	5	4	3	2	1	0
IPTR	MP/MC	OVLY	AVIS	DROM	CLKOFF	SMUL+	SST+

图 2-3 状态寄存器 ST0、ST1、PMST

2.2.3 C54x 的存储器组织

C54x 存储器由三个独立的可选择空间组成：程序、数据和 I/O 空间。所有芯片均有随机访问存储器（RAM）和只读存储器（ROM），RAM 分为双访问 RAM 和单访问 RAM，ROM 是用来存放固化的程序的。

- 片内 ROM

标准的 C54x 片内 ROM 中有一个引导程序，它可以将用户代码调入到程序存储器的任何一个位置，若 MP/$\overline{\text{MC}}$ 在硬件复位时低电平，执行从单元 FF80H 开始，此单元存有转移到引导程序开始处的转移指令。

- 片内 RAM

片内分为双访问 RAM（DARAM）和单访问的 RAM（SARAM）。DARAM 存储器在一个周期里能被访问两次，在复位时，DARAM 被映射到数据存储器空间，DARAM 也可以通过设置 PMST 中的 OVLY 位映射到程序/数据空间。片内的 SARAM 由几块组成，每块在一个机器周期里只能访问一次，SARAM 优先存储数据，也可以映射到程序空间来存储程序代码。

2.2.4 C54x 的寄存器

C54x 拥有众多的寄存器，有辅助寄存器、暂存器、过渡寄存器、堆栈寄存器、循环缓冲寄存器、块循环寄存器、中断寄存器。

- 辅助寄存器（AR0~AR7）

8 个 16-bit 的辅助寄存器（AR0~AR7）能被 CALU 访问，也能被辅助寄存器算术单元（ARAUs）修改，其主要功能是产生 16-bit 的数据空间，但也能用来作为通用寄存器和计数器。

- 暂存器（TREG）

TREG 为乘法指令和乘/累加指令存放一个乘数，带移位操作的指令，如 ADD，LD 的 SUB 存放一个动态的移位计数，也可作为 BITT 指令存放一个动态位地址。EXP 指令把计算出的数值存入 TREG，而 NORM 指令将 TREG 的值归一化。

- 过渡寄存器（TRN）

TRN 是一个 16-bit 的寄存器，用来得到新的度量值存放中间结果，以完成 Viterbi 算法，CMPS 指令（实现比较、选择并存储最大值功能）在累加器高位字和低位字进行比较的基础上修改 TRN 的内容。

- 堆栈指针寄存器（SP）

SP 是存放栈顶地址的 16-bit 寄存器，SP 总是指向压入堆栈的最后一个数据，中断、陷阱、调用、返回，PUSHED、PUSHM、POPD、POPM 等指令都要进行堆栈处理。

- 循环缓冲大小寄存器（BK）

由 ARAUs 用来在循环寻址中确定数据的大小。

- 块循环寄存器（BRC、RSA、REA）

块循环寄存器（BRC）在块循环时确定一代码所需循环的次数，块循环开始地址（RSA）是需要循环的程序的开始地址，块循环尾地址（REA）是循环程序块的结束地址。

- 中断寄存器（IMR、IFR）

中断寄存器（IMR）在需要的时候独立地屏蔽特定的中断，中断标志寄存器（IFR）用来指明各个中断的当前状态。

2.2.5 C54x 的片内外设

C54x 芯片有以下外设：通用 I/O 脚（$\overline{\text{BIO}}$ 和 XF），软件可编程等待状态发生器，可

编程的块切换逻辑，主机接口（HPI），硬件定时器，时针发生器和串口。

- 通用 I/O 引脚

每一种 C54x 芯片都有两个通用 I/O 引脚，$\overline{\text{BIO}}$ 和 XF。$\overline{\text{BIO}}$ 是用来监测外部设备状态的输入管脚。在对时间要求很严格的循环不能被外部中断所打断的时候，可以用 $\overline{\text{BIO}}$ 脚来代替中断与外设相连，根据 $\overline{\text{BIO}}$ 输入的状态来执行一个转移。XF 用于发信号给外部设备，可以通过编程控制。

- 软件可编程等待状态发生器

软件可编程等待状态发生器可以把外部总线周期扩展到 7 个机器周期，从速度上和较慢的片外存储器和 I/O 设备相匹配，它不需要任何的外部硬件，只由软件控制。

- 可编程块切换逻辑

可编程切换逻辑在访问越过存储器块边界，或从程序存储器跨越到数据存储器时，能自动插入一个周期，此额外的周期允许存储器器件在其它器件开始驱动总结之前释放总线，以此来防止总线竞争。用于存储器块切换的块大小由切换控制寄存器（BSRC）确定。

- 主机接口

主机接口（HPI）是一个 8-bit 的并口，提供 C54x 与主处理器的接口，C54x 和主处理机都可以访问 C54x 的片内存储器，并且通过它进行信息交换。

- 硬件定时器

C54x 有一个带有 4-bit 预分频器的 16-bit 的定时电路，定时计数器在每一个时钟周期中减 1，每当计数器减至 0 时产生一个定时中断，可以通过设置特定的状态来使定时器停止，恢复运行、复位或禁止。

- 时钟发生器

时钟发生器由一个内部振荡器和一个锁相环电路组成，可以通过晶振或外部的时钟驱动，锁相环电路能使时钟源乘上一个特定的系数，得到一个内部 CPU 时钟，故可以选择一个频率比 CPU 时钟低的时钟源。

- 串口

各种 C54x 的芯片配有不同的串口，但不外乎以下三种：同步的，缓冲的时分多路。

1) 同步串行 I/O 口：是高速、全双工串口，提供与编码器、A/D 转换器等串行设备之间的通信。当一个 C54x 芯片多个同步串口时，它们是相互独立的，每个串口都能工作到 1/4 的机器工作频率（CLKOUT）。同步串口为收发双向缓冲，单独由可屏蔽的外部中断信号控制，数据可以字节或字传送。
2) 缓冲串口（BSP）：是在同步串口的基础上增加了一个自动缓冲单元，并以整 CLKOUT 频率计时。它也是全双工和双缓冲的，能提供灵活的数据串长度，自动缓冲单元支持高速传送并降低服务中断开销。
3) 时分多路串口（TDM）：是一个允许数据时分多路的同步串口，既能工作于同步方式，也能工作于 TDM 方式下，在多处理器得到广泛的应用。

2.2.6 外部总线接口

C54x 有 64K 的 16-bit 并行 I/O 寻址空间，对外部存储器或 I/O 的访问则通过外部总线

进行，独立的空间选择信号 DS、PS 和 IS 允许进行物理上分开的空间选择。

接口的外部 ready 输入信号和软件产生的等待状态允许处理器与各种不同速度的存储器和 I/O 设备相连，接口的保持模式使外部设备能控制 C54x 的总线，这样的外部设备就能访问程序、数据和空间的资源。

2.2.7 IEEE 1149.1 标准扫描逻辑

IEEE 1149.1 标准扫描逻辑电路用于仿真和测试，它所对所连设备的边界扫描，同时也能用于测试引脚到引脚的连续性，完成 C54x 芯片的外围器件物操作测试。IEEE 1149.1 标准扫描逻辑与能访问片内所有资源的内部扫描逻辑电路相连，故 C54x 能使用 IEEE1149.1 标准串行扫描引脚和专用仿真引脚来完成在线仿真。

2.3 TMS320C55x 信号处理器

2.3.1 TMS320C55x 的描述

TMS320C55x 是在 C54x 的基础上经过改进发展起来的，它完全兼容 C54x 的指令，和 C54x 芯片相比，具有更强的运算能力，更低的功耗。它非常适合于数据速率高、运算量大、而功耗要求很低的的应用场合。

C55x 和 C54x 相比，在 C55x 的功能单元方面作了如下的扩展：

- 增加了两条总线，一条读操作数总线（BB），一条写操作总线（FB）；
- 增加了一个乘加单元（MAC）；
- 增加了一个 16bit 的 ALU；
- 将累加器增至 4 个，即 AC0、AC1、AC2、AC3；
- 临时寄存器增至 4 个，即 T0、T1、T2、T3；

与 C54x 相比，C55x 不仅增加了硬件资源，而且优化了资源的管理，故其性能大为提高，就以常用的 C5510 和 C5410 相比，前者运算能力最高为后者的 4 倍，而功耗仅为后者的 1/6。下面以 C55x 处理器最先推出的 C5510 来说明其内部及外部电路。

2.3.2 TMS320C55x 的结构

C55x 处理上较 C54x 具有更丰富的硬件资源，更快的运算速度，为了更好地管理这些硬件资源，依据功能是同将 CPU 分为 4 个主要单元：指令缓冲单元（Instruction buffer Unit）、程序流程单元（Program flow Unit）、地址数据流程单元（Address-data flow Unit）和数据计算单元（Data computation Unit），图 2-4 为 TMS320C55x 芯片的结构示意图。

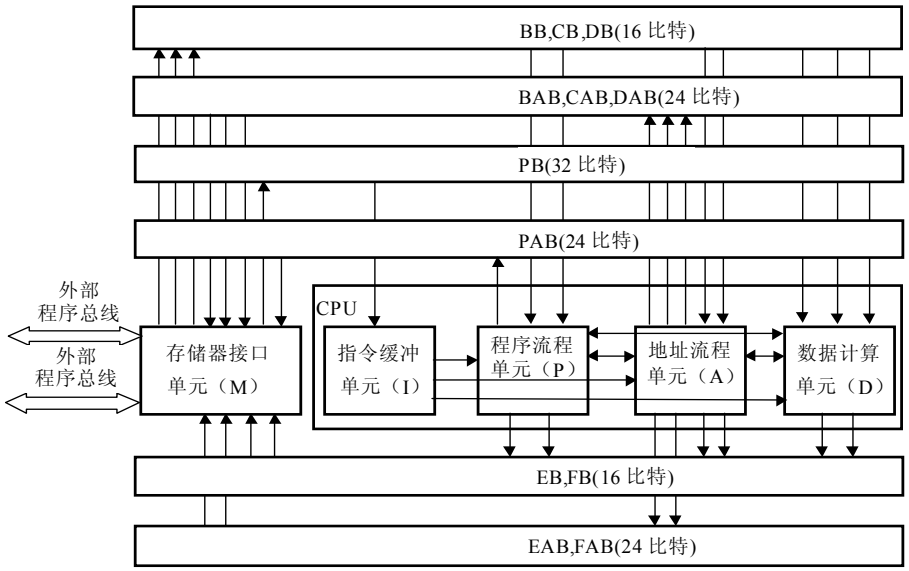


图 2-4 TMS320C55X 的 CPU 结构示意图

2.3.3 TMS320C55x 的 CPU 内部总线结构

由 C55x 的结构示意图可看出，C55x 有 1 条 32-bit 的程序数据总线（PB），5 条 16-bit 的数据总线（BB、CB、DB、EB、FB）和 6 条 24-bit 的程序及数据地址总线分别与 CPU 相连。所有的总线除了 B 总结，均可通过存储器接口单元同 DSP 外部程序数据总线相连，实现 CPU 对外部存储器的访问，这种并行的总线结构使 C55x 能在一个周期内完成一个 32-bit 程序代码的读、3 个 16-bit 数据的读和 2 个 16-bit 数据的写。

这些总线中，32 位的读程序数据总线（PB）和 24 位的读程序地址总线（PAB）配对使用，CPU 通过它们读取指定存储器单元内的程序代码，并送至 CPU 的指令缓冲单元 1。

3 条 24 位的读数据地址总线 BB、CAB 对应 CB 和 DAB 对应 DB。地址总线指定数据空间或 I/O 空间的地址，然后将该地址的数据通过数据总线传到 CPU 的各功能单元。其中，C 总线和 D 总线都与 P 单元、A 单元和 D 单元相连，而 B 总线只与 D 单元相连，用于完成从片内存储器到 D 单元 2 个乘累加器（MAC）的数据传送，另外，某些特殊指令也会在同一时间使用 CB、DB 和 BB 来读取 3 上操作数。

2 条 24 位的写地址总线（EAB、FAB）与 16 位的写数据总线（EB、FB）配对使用，即 EAB 对应 EB、FAB 对应 FB。CPU 的功能单元通过数据总线，将数据传到地址总线指定的数据空间或 I/O 空间。

2.3.4 TMS320C55x 的中央处理单元

TMS320C55x 的中央处理单元具有和 TMS320C54x 芯片的 CPU 相同的 40-bit 的算术逻辑单元（ALU），一个桶移位器，比较、选择和存储单元（CSSU），指数编码器等。

不同之处在于，TMS320C55x 有两个乘累加器（MAC）而 TMS320C54x 只有一个，双乘累加器支持乘、乘加和乘减操作，在一个周期内每一个乘累加器能完成 17 比特乘 17 比特的乘法和 40 比特加法或减法运算。

C55x 芯片有 4 个累加器，而 C54x 只有两个累加器。另外临时寄存器增至 4 个：ST0、ST1、ST2、ST3。

2.3.5 C55x 的存储器组织

- 片内存储器

以 C5510 为例，片内具有 256KB 的 SARAM、64KB 的 DARAM 和 32KB 的 ROM，较之 C54x 芯片具有更大的存储器。

- 外部存储器接口

仍以 C5510 为例，其 CPU 地址总线是 24 位的，能够寻址 16MB 的存储空间，除了片内的存储器外，其余存储空间是通过外部存储接口（EMIF，External Memory Interface）进行访问的。外部空间分为 4 个较小的空间，每个空间用一个片选信号来指定。每个片选空间相对独立，各占一段地址，可以接不同类型的存储器件，如异步 SRAM、ROM 或同步 SDRAM 等。每个片选空间可以单独编程设置，能适应不同速率的存储器件，故利用 EMIF 能够非常灵活方便地与多种存储器件相连，使硬件设计相对变得容易。

EMIF 分别和 CPU 总线、外设总线和 DMA 总线相连，使外设、增强型主机接口（EHPI）和片内存储器都能与外部存储器进行交换，也可利用 DMA 实现高速数据传输。

2.3.6 C55x 的寄存器

C55x 包含如下一些寄存器：

- 数据页寄存器，包括数据页寄存器（DPH，DP）和接口页寄存器（PDP）；
- 指针，包括系数馈针寄存器（CDPH，CDP）、堆栈指针寄存器（SPH，SP，SSP）和辅助寄存器（AR0-AR7）；
- 循环缓冲寄存器，包括循环缓冲大小寄存器和循环缓冲起始地址寄存器；
- 临时寄存器，包括 ST0、ST1、ST2、ST3。

2.3.7 C55x 的片内外设

C55X 有以下片内外设：

- 时钟发生器

时钟发生器由一个数字锁相环（DPLL）和一个控制寄存器组成，能对输入时钟进行1-4 分频和 1-31 的倍频。

- 定时器

片内有两个独立的可编程定时器（Timer0、Timer1），每个定时器都由 20 比特定时电路组成，计时器减到 0 时可向 CPU 发定时中断或启动 DMA 传输。

- 通用输入输出口

C5510 有一个专门的通用输入输出口（GPIO，Genernal Purpose Input/Output），它由 8 个相互独立的且与内部外设总线相连的可编程管脚构成。DSP 能设置每一个 GIPO 管脚输出的电平，用于对外部器件的控制或连络。

- 多通道缓冲串口

与 C54x 芯片相似，C5510 有三个高速全双工多通道缓冲串口（McBSP），能够连续收发数据，速率为最高时钟频率值的一半，可以灵活设置数据传输参数，包括帧、通道数量、每个数据单元的长度、发送时钟速率和是否进行 μ 率或 A 率压扩等。

每个 McBSP 串口的接收部分和发送部分是相互独立的，可以单独使用，无论接收通道还发送通道都具有多级缓冲能力，使用户可以进行连续的数据收发，而且在收到或发出一个数据时能够设置相应的标志，并向 CPU 发中断或向 DMA 发启动信号，使用户非常方便地对数据进行处理。

- 增强型主机接口

和 C54x 芯片一样，C55x 芯片也有 HPI 接口，专门用于 DSP 同外部主机相连，使外部主机能访问 DSP 的存储空间，和 C54x 芯片的 HPI 接口的不同之处是其接口是并行 16-bit 接口，可以按 8 位或 16 位方式进行访问，而一般的都是 8 位的并行接口。另外外部主机能够通过 HPI 口控制 C55x 的复位。在主机和 C55x 之间有专门的中断通道，能够通过 HPI 接口相互中断，实现快速握手，这种方式特别适合于构成主从处理的模式。

上面简单地描述了 C54x 和 C55x 芯片的基本情况，后面章节将以 C54x 芯片为例来介绍 C5000 系列芯片的应用与开发。

2.4 TMS320C54x 系列芯片的硬件设计

单有一个 DSP 芯片是无法使用的，它必须和其它相应的外围器件才能构成一个完整的系统。一个 DSP 硬件系统包括电源电路、复位电路、电平匹配电路、信号输入与输出电路等。

2.4.1 TMS320C54x 芯片的电源设计

TMS320C54x 系列芯片大部分采用低电压设计，这样可以大大地节约系统的功耗，该系列芯片的电源分为两种，即内核电源与 I/O 电源，其中 I/O 电源一般采用 3.3V 设计，而内核电源采用 3.3V，2.5V，或更低的 1.8V 电源，降低内核电源的主要目的是为了降低功

耗。下面我们以目前使用最流行的 TMS320VC5410 为例来介绍 TMS320C54x 的电源部分的设计。

一、电源电压及电流要求

TMS320VC5410 芯片是目前该系列芯片中用得较多的一种，因为它的最高运算能力可达 100 MIPS，除具有 TMS320C54x 芯片的一般功能外，其内部有 64k_16 bit 的内存，可以满足大部分的应用场合对快速 RAM 的需要，它的电源电压有 3.3V 与 2.5V 两种，其中 3.3V 电压供 I/O 接口用，2.5V 电源主要供器件的内部，包括 CPU 和其它所有的外设逻辑。

TMS320VC5410 的电流消耗主要取决于器件的运算的负载能力，就是说如果器件处于全速运行的时候，那么其电流就达到该芯片的电流消耗的极大值，当 CPU 处于等待状态时，其电流消耗就很小，因此我们在编程的时候,当 CPU 没事可做的时候，尽量使其处于等待状态或者使 CPU 处于休眠状态，这样可以降低系统的功耗；相对于 CPU 来说,外设消耗的电流是比较小的，它由 I/O 电源提供，消耗的电流主要取决于外部输出总线的速度及负载电容。

二、TMS320VC5410 的电源产生

电源的产生一般由 5V 电源产生 3.3V、2.5V 或 1.8V，产生电源的芯片比较多，如 Maxim 公司的 MAX604，MAX748，TI 公司的 TPS72XX 系列，这些芯片又可以分为线性和开关两种，在设计的时候应根据实际的需要，如果系统对功耗要求不是很高的情况下，可以使用性线稳压器，它的使用方法较为简单，而且电源的纹波电压较小，对系统的干扰较小；而在系统对功耗要求比较苛刻的情况下，应使用开关电源芯片，一般的开关电源芯片的效率可以达到 90%以上，但是一般而言，开关电源产生的纹波电压要比线性电源产生的大，而且开关电源的振荡频率在几千赫兹到几百千赫兹的范围内，会对 DSP 系统产生干扰。特是开关电源产生的电压用于 A/D/A 电路时应加滤波电路，以减小电源噪声对模拟电路的影响。

下面的使用 MAX748 产生 3.3V 电路，该电源可以产生最大 2A 的电流。

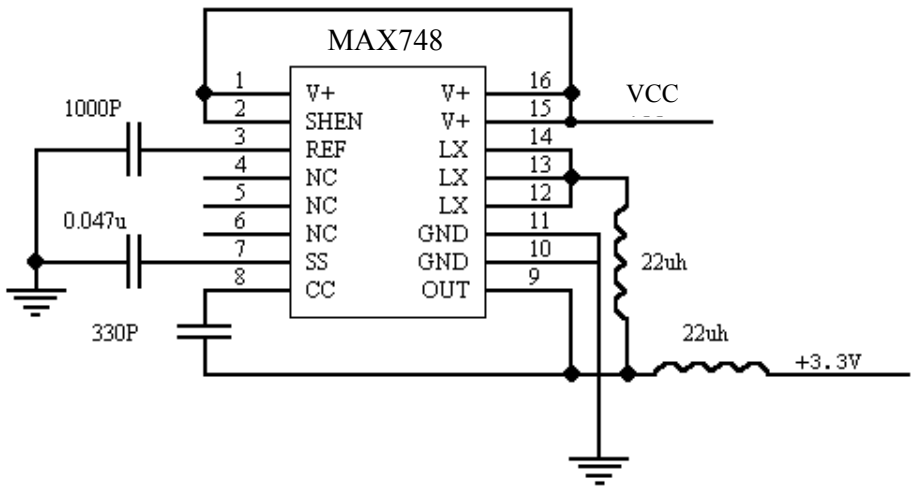


图 2-5 使用 MAX748 芯片产生 3.3V 电源的原理图

下面是利用 TPS72XX 系列芯片产生线性电源的连接方法，它可以根据所选择的芯片不同可以产生 3.3V，3.0V，2.5V 等电压。

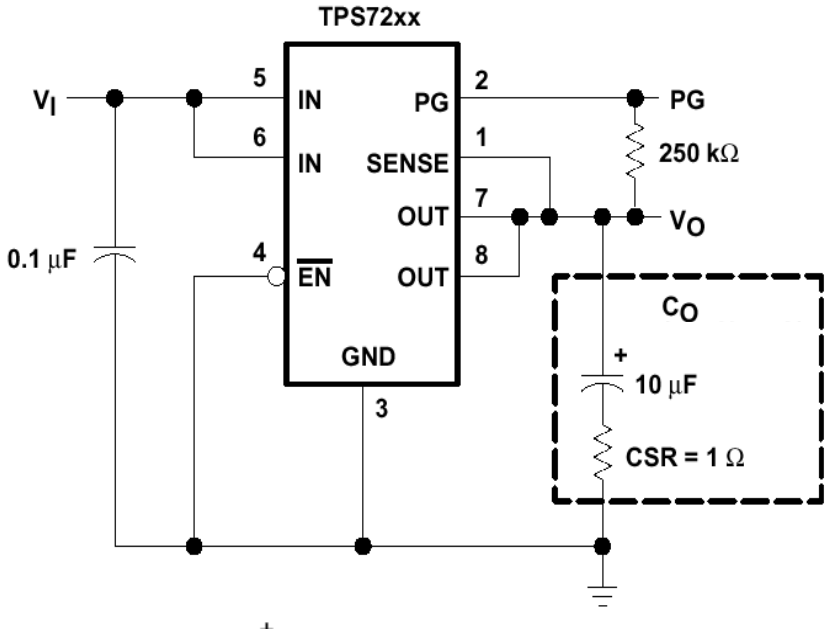


图 2-6 利用 TPS72XX 系列芯片产生 DSP 所需电源的电原理图

2.4.2 DSP 系统的硬件复位和看门狗电路

对于一个 DSP 系统而言，上电复位电路虽然只占很小的一部分，但它的好坏将直接影响系统的稳定性。

早先一般使用的是 RC 设计的简单的复位电路，但在实际使用中存在某些不稳定的情况，在某些情况下会导致上电复位的失败。

另外在系统的运行中，由于干扰等原因导致系统崩溃，此时如果有看门狗（Watch dog）电路可以使系统重新复位，使之恢复正常的工作。

下面介绍一种简单的复位电路。

Maxim 公司的 MAX706 芯片是一个非常好的上电复位及看门狗电路，为了更好的和 TMS32054X 芯片的 3.3V 系列相配合，最好选用 MAX706R，该芯片具体接法如下图：

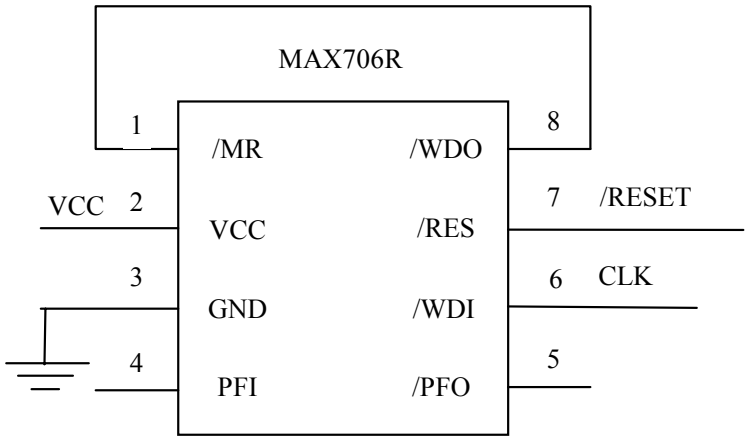


图 2-7 使用 MAX706R 作为 DSP 系统的复位及看门狗的电原理图

MAX706R 的 6 脚需要输入一个周期不小于 10Hz 的脉冲信号，该信号将由 DSP 通过程序产生。该电路不但在系统上电时能产生一个标准的复位脉冲，而且当 DSP 处于不正常工作状态时，通过程序产生的周期性脉冲消失，此时将在 7 脚上产生一个不小于 1.6 秒的复位脉冲信号，确保系统重新复位，程序从新开始运行，强行使系统恢复正常。

2.4.3 3.3V 和 5V 混合逻辑设计

在设计 DSP 系统时，如果都能采用 3.3V 芯片设计当然最好，这样其接口电平相匹配，不存在电平转换的问题。但在实际上往往还不能避免混合设计，即在一个系统中同时存在 3.3V 和 5V 系列芯片，让两种电压芯片的输入输出直接连接是不行的，因为 5V 的芯片可以承受 3.3V 的电压，但是 3.3V 不能承受 5V 的电压。所以在有 5V 和 3.3V 芯片共存的电路中就存在一个混合逻辑设计的问题。下表是各种电平的数据。

表 2-1 5V TTL、CMOS 和 3.3V 逻辑电平比较

	5V TTL 电平	5V CMOS 电平	3.3V 逻辑电平
V_{OH}	4.4V	2.4V	2.4V
V_{OL}	0.5V	0.4V	0.4V
V_{IH}	3.5V	2.0V	2.0V
V_{IL}	1.5V	0.8V	0.8V
V_t	2.5V	1.5V	1.5V

表中：V_{OH} 为输出高电平的最低值，

V_{OL} 为输出低电平的最高值，

V_{IH} 为输入高电平的最低值，

V_{IL} 为输入低电平的最高值，

V_t 为“0”、“1”电平的分界值。

从表中可以看出，在 5V CMOS 电压和 3.3V 电平转换时就存在电平匹配问题，例如在程序载体 29F010 或 29F020 与 TMS320VC5410 接口的时候就必须有电平转换，电平转换的芯片有 AN74ALVC16425 和 AN74LCX245 等，AN74ALVC16425 是一个 16 位的收发器，

可以用在需要转换的比较多的场合，如用于 16 位数据线转换最合适，而 AN74LCX245 是一个 8 位的收发器，可以用于 8 路以下的转换。

从目前的趋势来看，使用低电压的 3V 系列的芯片已经是发展方向，所以我们在设计的时候应尽量地使用 3V 芯片，一是可以设计成一个低功耗的系统，另一个方面可以避免混合系统设计中的电平变换问题。

2.4.4 TMS320VC5410 系统的信号输入输出

利用 TMS320VC5410 芯片构成一个最简单的系统，它需要信号的输入与输出通道，在这里仅讨论最简单的音频信号的输入输出的处理。

音频信号的输入与输出是大部分的 DSP 平台不可缺少的一个部分，如立体声音响处理系统，语音编码处理系统等均需要对声音信号进行采样处理，并将接收的信号还原成声音。

以前对语音的处理是用一般的 A/D 和 D/A 芯片，如 AD574（A/D）和 DAC1210（D/A）等，这些芯片一般都是并口操作，连线比较复杂，并且这些芯片没有采样保持和抗混迭滤波，因此使用这些片子还需复杂的外围电路。随着技术的发展，一种Σ–Δ调制技术使得音频的输入输出技术取得比较大的进步，它可以将音频采样和音频输出集成在一个芯片上，另外还能将抗混迭滤波等做在同一个芯片上，典型的芯片如 AD 公司的 AD73311、AD73322 以及 TI 公司的 TLV320AIC10 等，下面以 TI 公司的 TLV320AIC10 为例作介绍。

TMS320AIC10 的框图如图 2-8 所示，

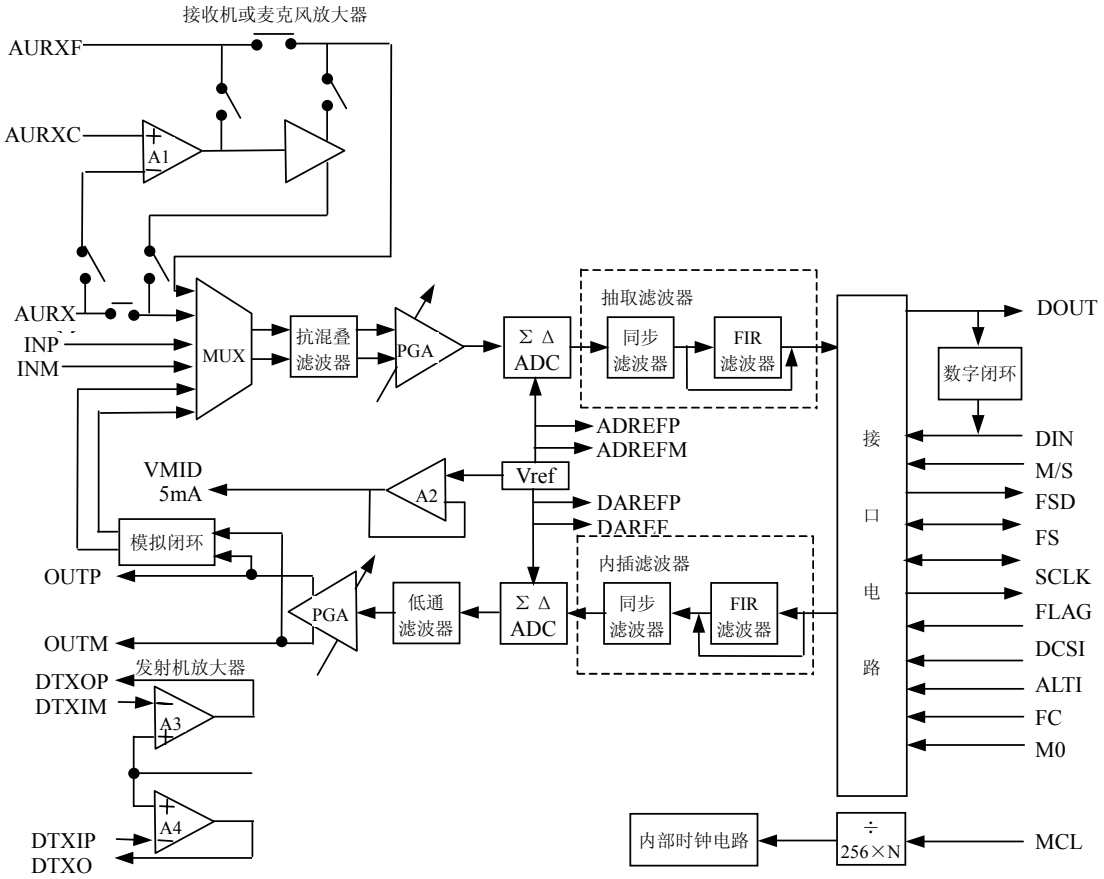


图 2-8 TLV320AIC10 的结构图

其基本的性能如下：

- 16bit 的Σ–ΔA/D 转换器；
- 16bit 的Σ–ΔD/A 转换器；
- 最大的转换速率：
  1) 带在片的 FIR 滤波器时为 22Ksps；
  2) 跳过 FIR 滤波器时为 88Ksps；
- 在跳过 FIR 滤波器模式且最终采样率为 8ksps 时，音频带宽内的信噪比：
  1) 5V 供电 88ksps 时可获得 90dB 的 SNR/ADC 及 87dB 的 SNR/DAC；
  2) 3V 供电 88ksps 时可获得 87dB 的 ANR/ADC 及 85dB 的 SNR/DAC；
- 在 11kHz 的带宽内用片内的 FIR 滤波器可获得 84dB 的 SNR/ADC 及 85dB 的 SNR/DAC；
- 提供片内的抗混迭模拟滤波器及片内的运算放大器；
- 可编程的 ADC 及 DAC 速率及可编程的输入输出增益控制；
- 可通过软件分别控制 ADC 或 DAC 的进入 Power-down(省电)模式；
- 提供标准的与 DSP 相连的串行接口，可连接 DSP 的 SPI 端或串行口；

- 自动级连检测与级联编程，最多可以有 8 片级联；
- 在 3.3V 工作电压下 8k 采样率典型的功耗为 39mW 等。

图 2-9 为典型的 DSP 与 TLV320AIC10 的连接方法（级联）。

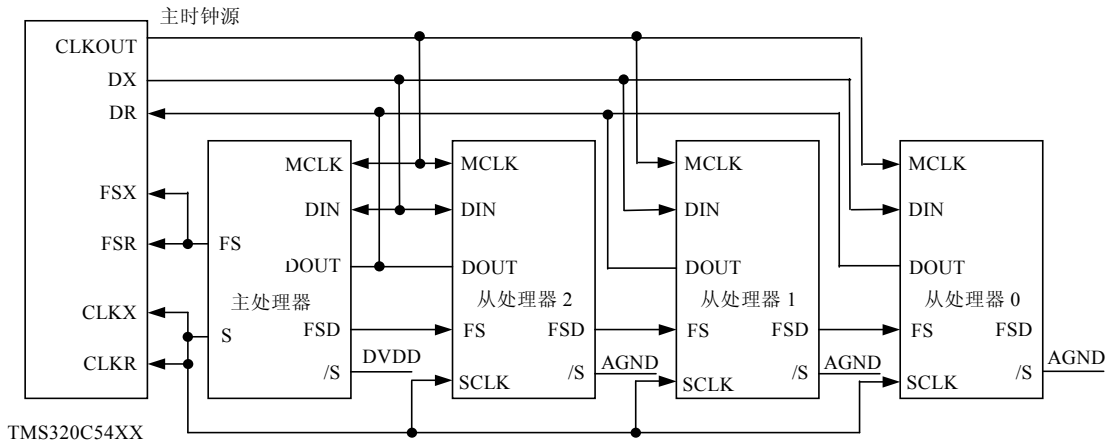


图 2-9 DSP 与 TLV320AIC10 的主从级联方式

2.5 TMS320VC54x 的软件设计

2.5.1 编程时需注意时序问题

一、系统上电时要注意的问题

在一个 DSP 系统中会不可避免地存在时序的问题，在系统中各种速度不一样的电路对时序会有不同的要求，在某些情况下程序本身并没有问题，但运行的结果就是不对。举一个很简单的例子，系统刚上电时，如果 DSP 立即对外设进行操作，在很多情况下会失败，因为一般情况外设的上电复位的速度要比 DSP 慢得多，如 A/D，D/A 芯片等，在 DSP 可以工作的时候，它还没有完成上电复位，此时如果对其进行操作将会失败，所以在上电的时候，DSP 必须等所有的其它电路准备好之后才可以运行程序，对其它电路进行操作。所以在 DSP 系统上电的时候，一般在程序加载完成之后需要插入一段等待时间，可以使用循环空操作指令完成。待系统中其它的芯片都完成上电复位的过程之后再对它操作，就可以解决这个问题。

二、流水线冲突

在 TMS32054X 中采用深度为 6 级的流水线操作，因此流水线冲突将是不可避免的。一般情况下，发生流水线冲突时，由 DSP 自动插入延时以解决冲突问题。但在某些情况下 DSP 无法自动解决冲突问题，这就需要通过调整程序次序或人为改变时序，即在合适的地方加入一个或数个空操作指令，这种情况对于初次编程的人员来说是很不习惯的。在某些情况下，程序也看不出有任何的问题，但运行就是不正确的情况，就要考虑是否存在流水

线冲突的问题。

三、编译模式的选择

在 ST1 状态寄存器中，有 1 位编译器模式控制位 CPL。CPL 用于指示相对直接寻址中采用哪种指针。当 CPL=0 时，使用页指针，使用页指针 DP，当 CPL=1 时，采用堆栈指针 SP，要注意在切换模式时可能引起的流水冲突。

另外的 ST1 中的符号扩展的使用也应注意，SSXM 及 RSXM 的使用必须及时切换，否则就会出现溢出的情况。

四、某些指令对存储器的要求

在 TMS32054X 指令系统中的某些指令对存储器是有特殊要求的，如 MAC 和 MACD 指令，

```
MACD      Amem,Bmem,scr,dst
```

在这条指令中的 Amem，Bmem 存储器是双寻址的，要求前两个操作数位于片内的 DRAM 中，否则就会出错。

2.5.2 软件编程中的一些技巧的运用

一、合理使用存储器

TMS32054X 系列芯片内部集成了不同容量的 RAM 和 ROM，内部的 RAM 和 ROM 在运行的时候不需要插入等待，便于程序的全速运行。

片内的 ROM 对于一般的用户是无法使用的，只是在产品成熟之后需要将某些固化在片内的时候可以到 DSP 的生产厂家做掩膜（厂家要求芯片数量一般在 10K 以上）。

目前所用的 TMS32054X 系列芯片的内存越来越大，如 TMS320VC5410 具有 64K×16 的片内 RAM，TMS320VC5416 有高达 128K×16 的片内 RAM，如此大的片内 RAM 对于一般的应用场合已经足够。

但如果在某些场合，如为了节约成本选用片内 RAM 较小的 CPU 或程序太大，除了片内 RAM 外，还需要片外的 RAM，在这种情况下就要合理地分配片内 RAM 的使用。如果某些程序需要大量的运算，如卷积运算和其它的滤波运算的时候，留一部分片内单元给这部分程序，这样可以提高系统的运算速度和效率。而对于一般对 RAM 操作较少的程序就不要分配固定的片内的 RAM。

另外可以留出一部分公用的片内 RAM，程序用过之后立即清除，以便于其它的程序重新使用这部分 RAM 空间，当然在使用的时候要注意不要重复使用，尤其在中断使用公用的 RAM 时往往会出错，需要特别注意。

二、程序的模块化、程式化设计

在编制程序时往往会用到一些相同的程序，或者是程序大致相同，仅需修改个别参数，这种情况在编程序的时候尽量使用公用的程序，在调用前给出相应的参数即可。模块化的设计可以减少程序量，减少编程的错误。如常用的延时程序，FFT 变换、IFFT 变换程序，FIR 滤波器程序等，这些程序一般都具有固定的格式，有成熟的程序段直接可以利用，这样可以为我们节省大量的编程时间。

三、使用 C 语言还是汇编语言的选择

C 语言编制的程序通用性好，易修改、易移植，便于系统的升级。但是 C 语言转换成汇编语言的效率只有 50%左右，在某些复杂操作情况下的效率更低，所以对于系统在运算量巨大，程序空间和数据空间有限的情况就不适合使用 C 语言。

汇编语言具有直观性好，直接面向对象操作，效率很高，可以最大限度地利用 DSP 的运算能力，节省程序和数据空间，但是它存在可移植性差、修改困难等缺点。

所以在实际的运算中要根据情况选用合适的语言。但是对于初学者，使用汇编语言可以更好地理解 DSP 的操作过程，在调试过程中也更能直观地看到运行的结果。

2.5.3 TMS320VC5410 的 BOOT 设计

TMS32054X DSP 的芯片一般都在片内设置了 BOOT 程序，BOOT 程序的主要作用是在开机时将用户的程序从外部程序存储器装载到片内或片外的随机程序存储器中，本节以 TMS320C5410 为例说明 BOOT 程序的使用方法。

TMS320VC5410 提供了多种 BOOT 方法，包括并行 I/O 口 BOOT、串行口（标准/TDM/BSP）BOOT、HPI BOOT、外部并行 BOOT、WARM BOOT，并支持 8 位/16 位及多块程序的 BOOT，这些不同的方式可以满足不同用户的需求，在 MP/MC 管脚置低，这样上电复位后，程序自动从内部的 FF80H 地址开始运行。在 FF80H 处，有一条跳转到 BOOT 程序的指令，开始运行内部的 BOOT 程序。

在 BOOT 在运行搬移程序前，首先进行初始工作：将中断置为无效（INTM=1），内部 RAM 映射到程序/数据区（OVLY=1），对程序和数据区均设置 14 个等待。

BOOT 程序选择何种 BOOT 方式是根据外部设置的不同条件实现的，并且判断条件是有先后次序的，用户想使用特定的 BOOT 方式，必须了解相应的 BOOT 程序工作流程。

图 2-10 是 TMS320VC5410 和 TMS320VC5416 的上电加载的框图，从图上可以很清楚地看出加载种类的判断和加载的过程。

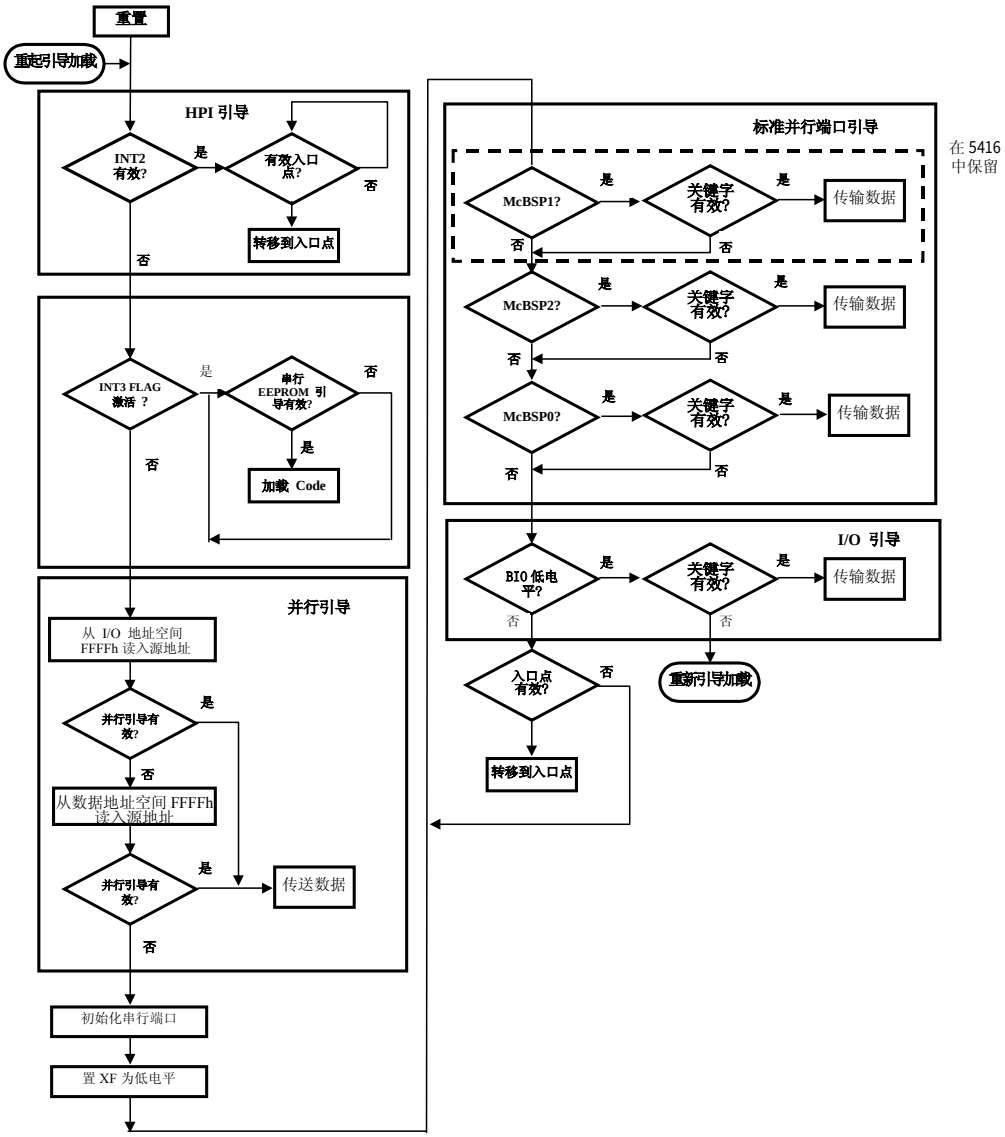


图 2-10 TMS320VC5410 和 TMS320VC5416 加载顺序图

利用并口加载是常用的一种方法，在加载过程中，根据外部程序存储器的数据位数的不同又可以分为 8 位或 16 位数据方式，其加载的流程图如图 2-11 所示。

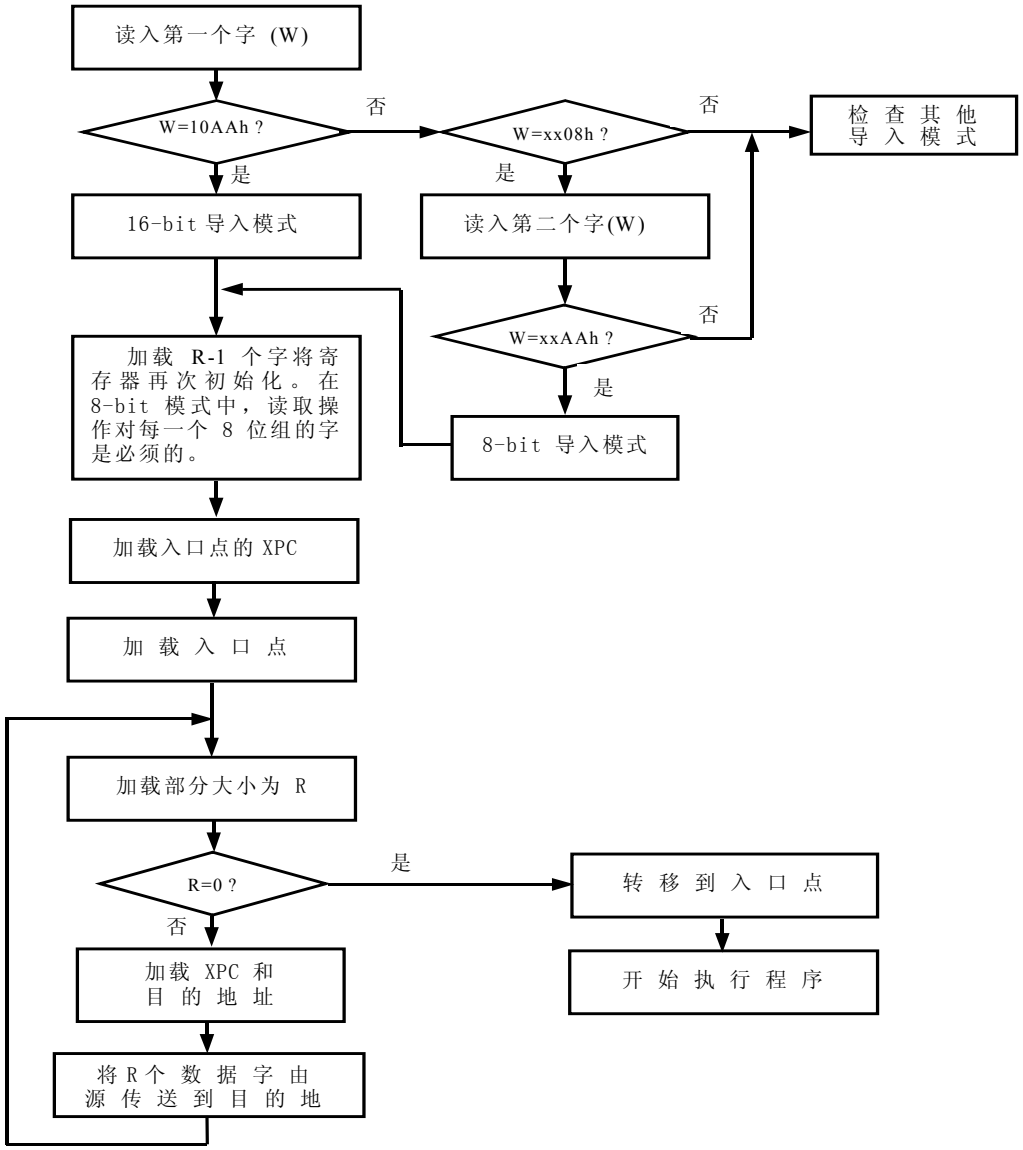


图 2-11 TMS320C54x 的外部并口 8 位或 16 位加载顺序图

2.6 DSP 芯片的开发工具

2.6.1 引言

可编程的 DSP 芯片开发需要一整套的硬件和软件开发工具，通常又可以分为代码生成工具和代码调试工具两类。代码生成工具是将用汇编语言或用 C 语言编写的 DSP 程序编译并链接成可执行的 DSP 程序；代码调试工具作用是对 DSP 程序及系统进行调试。TI 公

司及许多第三方公司为 DSP 的系统集成和软硬件的开发提供了许多适应的工具。

图 2-12 为 TMS320C54x 典型的软件开发流程。

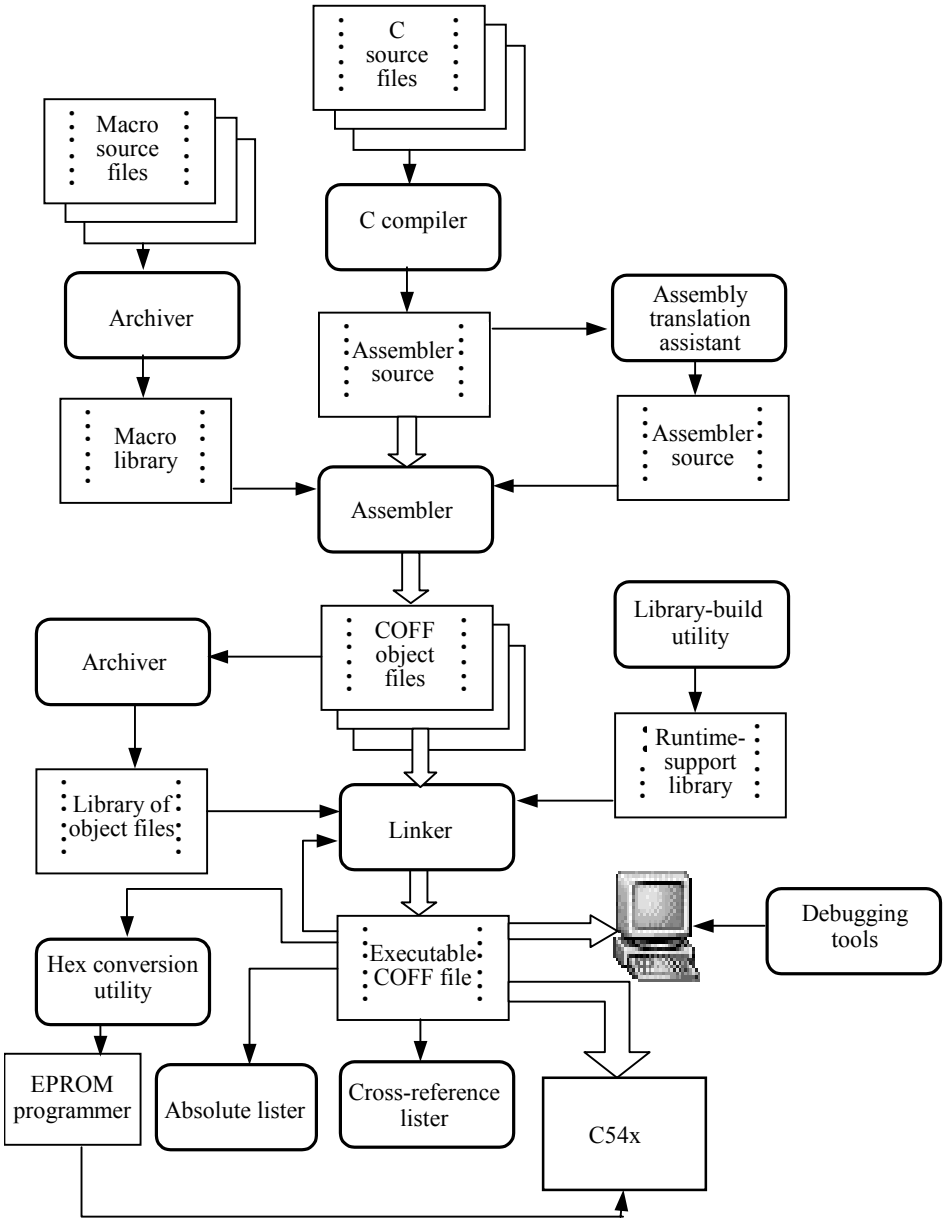


图 2-12 TMS320C54x 软件开发流程

2.6.2 代码生成工具

代码生成工具主要有宏汇编器/链接器（Micro Assempler/Linker）,C 语言编译器以及一些辅助的工具，如文件格式转换、库生成及文件管理程序等。

一、 汇编器（Assembler）

TMS320 的汇编器将汇编语言的源程序文件汇编成为机器语言的目标程序文件，其格式为 COFF（公用目标文件格式）。汇编语言源程序可以包括汇编语言指令、汇编指令和宏指令。汇编器的输入文件为汇编语言源文件，其缺省的文件扩展名为“.asm”。由汇编器所建立的目标文件其缺省文件扩展名为“.obj”，用汇编器建立列表文件，其缺省的文件扩展名是“.lst”。

汇编器所作的工作包括：

- 处理汇编语言源文件中的源语句，产生一个可重新定位的目标文件。
- 根据要求，产生源程序列表文件。
- 根据要求，将交叉引用列表加到源程序列表中。
- 将代码分为段（section），并为每个目标代码段设置一个段程序计数器（SPC,section program counter）。
- 定义和引用全局符号。
- 汇编条件块。
- 支持宏调用，允许在程序中或在库中定义宏。

二、连接器（Linker）

连接器的基本任务是将目标文件连接在一起，产生可执行模块。连接器可以接受的输入文件包括汇编器产生的 COFF 目标文件、命令文件、库文件以及已部分连接好了的文件。它所产生的可执行 COFF 目标模块可以装入各种开发工具，或由 TMS320 器件来执行。

连接器的功能简列如下：

- 定义一个与目标系统存储器一致的存储器模块。
- 将目标文件段组合起来。
- 将程序段定位到目标系统存储器的特定区域，赋予它们最后的地址。
- 定义或重新定义全局符号，以赋予它们特定的值。
- 重新解决输入文件之间的没有定义的外部引用。

根据用户的要求，该连接器还可以建立一个连接映射列表，用来描述存储器的分配、输入和输出程序段的位置以及外部符号重新定位后的地址。

三、归档器(Archiver)

归档器允许用户将一组文件归入一个档案文件（库）。例如，将若干个宏归入一个宏库，汇编器将搜索这个库，并调用源文件中使用的宏。也可以用归档器将一组目标文件收入一个目标文件库，连接器将连接库内的成员，并解决外部引用。

四、交叉引用列表器（Cross-reference Lister）

交叉引用列表器是一个查错的工具。它接受已经连接好的目标文件作为输入，产生一个义叉引用列表作为输出。它列出符号、符号的定义以及它们在已经连接的源文件中的引用。

为了使用这个交叉引用列表器，在汇编源程序时，汇编命令中加入一个适当的选 项，在列表文件中产生一个交叉引用表，并在目标文件中加入交叉引用的信息。连接目标文件得到可执行文件。再使用交叉引用列表器，就可以得到希望的交叉引用列表。

五、C 编译器

C 编译器是一种将 C 语言自动编译成为 DSP 汇编程序的代码生成工具。TI 对定点 DSP

芯片 TMS320C2x/C5x/C54x 和浮点 TMS320C3x/C4x 等提供了相应的 C 编译软件工具。为 DSP 的开发提供了一种极为便利的开发手段。仍以 TMS320C54x 为例来说明。

1. ANSI C 标准语言

C54x 编译器完全兼容 K&R C 语言的第二版的标准，包含扩展 C 的 ANSI C 标准提供最大限度的简洁和兼容能力。

编译器工具所有的库函数均包含在标准的 ANSI C 函数库中，这些库包括标准的输入和输出、串操作、动态存储器重新定位、三角、指数、双曲线等函数，但基于实际目标系统的信号处理函数则不包括在内。

2. 各种输出文件

汇编程序的输出。

C 编译器产生汇编语言程序，这些程序使你更简单地检查，也能使你看到如何由 C 源文件产生汇编代码。

公用目标文件（COFF）

公用目标文件（Common Object File Format）使用户能够在连接时定义自己系统存储器映射，将 C 代码和数据连接到指定的存储器，从而最大限度地改善其性能，COFF 还可以支持源程序级的调试。

代码和初始化到 ROM

对于独立的嵌入式应用，该编编译器允许 C 代码和初始化数据连接入 ROM 中，并允许在复位后立即运行 C 代码。

3. 编译码的接口

编译器外壳程序

C 编译器工具包括了一个程序组，如经常用到的编译、汇编和单步连接等程序。

灵活的汇编语言接口

该编译器具有灵活的汇编语言接口，使用直接调用协议，可以方便地编写能互相调用的汇编和 C 函数。

4. 编译器的操作

C 预先处理器

C 预先处理器（C preprocessor）是和分析器（paser）集成在一起的，作快速编译，即作第一遍编译，其处理内容包括：宏定义和扩展、#incloude 文件、条件编译以及各种预先处理指令等。

5. 优化处理

源内表程序

该编译器能作复杂的优化，使用多种街技术来从 C 源代码产生高效的汇编代码，通用优化可用于任何 C 代码，而对各 C54x 的优化更适合于 C54x 的特定的结构。它通过简化循环、重新安排语句和表达式，将变量安排入寄存器等方法来改善运行速度，减小 C 程序的大小。

建库公用程序

用户可以使用 C 编译器软件所提供的建库公用程序（Library-Build Utility）,在该软件中包括了所有的运行支持函数，用户可通过不同的选择来建立自己的运行支持库。

2.6.3 系统集成与调试工具

TI 公司为 TMS320 系统的集成与调试所提供的工具包括调试器接口（C/Assembly source debugger）、软件仿真（Simulation）、DSP 入门套件（DSK，DSP Starter Kit）、标准评估模块（EVM）、以及开发系统 XDS（extended developmemnt system）等。

一、调试接口（C/Assembly source debugger）

TMS320 的调试器接口为嵌入式系统的开发提供了丰富的功能与灵活性。该调试器是接着要讨论的软件仿真器、评估模块、在线仿真器等的标准接口。

该调试器可以运行一般的 PC 平台上，对用 C 或汇编语言写的程序提供完全的控制。其代码分析功能通过快速确认最费时的程序段，提示应该将开发时间集中在哪里，图 2-13 为 TMS320C54x 的调试器的屏幕显示。

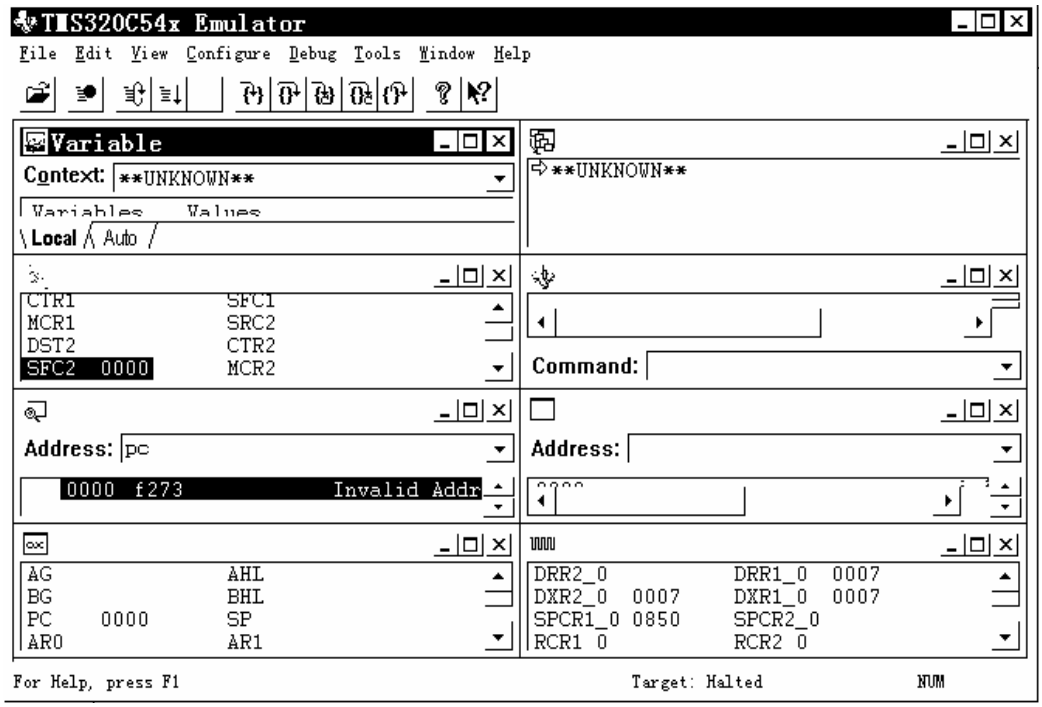


图 2-13　TMS320C54x 调试器界面

1. 调试器的性能

条件执行和单步执行使用户可以完成控制程序的运行。可以用鼠标或键入命令的方式设置或取消断点。存储器的分布与目标系统一致，以便于调试器访问和定义。调试器可以接受从批处理文件来的命令，从而容易进入经常使用的命令序列。

该调试器的主要功能包括：

- 支持多操作

对于 C2xx、C4x、C5x、C54x，C/汇编调试器增强了并行处理的能力（多处理 debug、断点、单步）。

- 多层调试

允许对 C 和汇编语言作调试。在调试 C 程序时，可以选择查看 C 源代码，汇编的目标代码，或者同时查看。

- 可灵活配置的窗口式界面

该调试器将代码、数据、以及命令分为可以管理的信息。用户可以从多个显示中选择，也可以根据用户的使用习惯建立最适合于应用的显示。

- 灵活的命令输入方式

命令的输入可以用鼠标、功能键、下拉菜单。过去的命令序列可以作成批处理文件重新运行。支持符号调试，故结构和变量名可以用来取代地址。

- 全屏编辑

在任何窗口显示的任何值都可以很方便地用鼠标定位，键入新的值来修改。

- 连续更新

屏幕上的信息连续更新，并将改变了值增高。

- 数据显示

可以方便地建立窗口来观察、显示和编辑变量、数组、结构、指针的值。任何类型的数据都以其自然格式显示（浮点数、整数、字符、指针）。

- 存储器窗口

存储器的内容可以显示和编辑。用户可以时刻的数据，并将期望的数值与实际显示的数值相比较。

2. 调试器和调试模式

调试器提供三种调试模式：自动模式、汇编模式、混合模式。

- 自动模式：在自动模式中，调试器自动显示当前正在运行的代码，或者是汇编语言，或者是 C 语言。这也是缺省的调试模式；
- 汇编模式：在这种模式下，显示的是汇编语言，而不管运行的是 C 语言还是汇编语言；
- 混合模式：在这种模式下，可以同时显示 C 语言程序和汇编程序。

3. 调试器窗口

调试器能显示多种不同类型的窗口。窗口的名称显示在窗口的项行。调试器共有八种类型的窗口，可以分为三大类。

1) 命令显示窗：命令窗口提供一个可以键入各种命令的区域，并显示各种信息，如进程信息、错误住处或命令输出。
2) 代码显示窗：显示汇编语言代码或 C 语言代码。有 3 种代码显示窗口：
   (1) 反汇编窗口——显示内存的反汇编码
   (2) 文件窗口——显示任何文本文件，主要是 C 语言源程序
   (3) 进程调用窗口——运行 C 代码时显示当前跟踪运行的函数进程
3) 数据显示窗：观察和修改各种类型的数据。有 4 种数据显示窗口：
   (1) 内存显示窗口——显示一定范围的内存内容
   (2) CPU 窗口——显示处理器各个寄存器的内容
   (3) 数据显示窗口——显示处理器各个寄存器的内容
   (4) 数据显示窗口——显示一个集合的数据类型，如数组、结构等

(5) 观察窗口——显示已选定的数据，如变量、寄存器或内存的某个单元

调试器的窗口可任意移动或改变大小，在数据显示窗口中可以编辑其中的任一个值。当然，在作这些工作之前，必须首先选择好窗口并使之成为活动窗口。

4．调试器的命令输入

调试器提供了非常灵活方便的命令输入方法。调试器提供的命令输入方法主要有以下几种：

1) 命令行输入

命令行输入就是在调试器的命令窗口中键入命令。如：

go main ：执行到程序的 main 处

win CPU：激活 CPU 窗口

利用调试器的记忆功能可以用简便的方法输入以前键入过的命令。如用 TAB 键可以向后逐条移动执行过的命令，用 TAB+SHIFT 键可以向前逐条移动执行过的命令，选择好以后按回车键即可。

2) 菜单输入

菜单输入是一种常用的输入方法。选择菜单可以采用三种不同的方法：鼠标输入、按键输入和热键输入。热键输入主要用于没有下拉菜单的主菜单，如 RUN、STEP 和 NEXT。鼠标输入比较方便，用鼠标左键点中相应的主菜单和下拉菜单即可。按键输入时首先同时按下 ALT 键和代表主菜单的高亮字母以选择主菜单，然后按下代表下拉菜单的高亮字母即可。当然也可以用方向键选择好下拉菜单再按回车键。

3) 批文件输入

调试器提供的 take 命令可以使调试器执行一个批处理文件。设有一个批处理文件 batcom.fil 的内容为：

```
reset
load sample.out
go main
```

则调试器的命令行中键入 take batcom.fil 就使调试器在复位系统之后，装入 sample.out 并执行至程序 main 处。

5. 调试器使用方法简介

使用调试器调试程序主要有以下几个步骤：（1）调试器配置；（2）装入程序；（3）运行程序：（4）观察运行结果；（5）编辑更改数据。

1) 调试器配置

在使用调试器之前，首先必须根据需要配置调试器。调试器配置包括内存配置和显示配置，其中内存配置尤为重要。

内存配置用于告诉调试器哪些内存有效，哪些内存不能进行存取。因此提供给调试器的内存映象应与实际目标系统的内存配置相一致，当然内存映像也应与链接命令文件中的 MEMORY 定义相匹配。当内存映像配置好以后，启动调试器运行程序时，如果访问一个没有定义的存储区，则调试器将显示一个错误信息。

一般的配置内存可以启动调试器之前进行。方法就是在调试器能够自动读到的批文件 init.cmd 中放入内存映像命令。启动调试器，在初始化过程中读入 init.cmd 的命令，从而完

成内存配置。需要注意的是，对于不同的调试环境，init.cmd 的文件名有所不同，如模拟器的文件名为 siminit.cmd，仿真器中的文件名为 emuinit.cmd 等。

2) 程序的装入

将源程序经汇编器汇编产生的.out 文件装入调试器即可运行。方法是在主菜单的 File 命令下选择 LOAD 命令，键入文件名即可。

3) 运行程序

运行程序的方法主要有全部运行、断点运行和单步运行等，具体使用如下：

- run 为基本的运行命令，执行该命令后，程序开始运行直到遇到断点或人为中断。
- runb 用来运行并计算一段程序的时钟周期数，若要计算某上段程序的周期数，在该段程序的首尾各设置一个断点，先用 run 运行至前面的断点，然后再用 runb 运行至后面的断点，再在命令窗口输入？clk 命令就可知道该段程序运行所用的周期数。
- Go 命令用来执行程序到某一点。
- Step 和 next 都是单步招执行的命令，在同之处在于，step 是顺序执行指令，而 next 在遇到函数调用时不进入内部，而直接执行到调用语句的下一条指令。

4) 观察运行结果

调试器提供了灵活的观察运行的手段，观察数据的方法有以下三种方法：

- 直接在已有的显示窗口中观察。
- 在命令窗口中用“？”命令观察。
- 增加显示窗口跟踪变量。窗口有两种，一种是 WATCH 窗，可以观察单个变量、寄存器或指定的单元；另一种是 DISP 窗，可以观察集合数据类型，如数组、结构等。

5) 编辑和更改数据

可以直接在数据显示窗口中更改，如在 CPU 的窗口直接更改寄存器的值，存储器窗口可更改内存的内容；另外一种方法用表达式的附加作用更改数据，如？AR3=5，即付 AR3 的值为 5。

二、代码分析器（Code Profiler）

代码分析的功能可以提高调试器的灵活性。在熟悉的调试器界面上，代码分析器通过快速确认时间花费最多有程序段，提示开发应集中在什么地方。通过消除瓶颈问题，优化后的代码可以极大地缩短执行时间。一个功能强大的代码分析命令集将代码优化的过程简化。

代码优化器的主要性能包括：

与 C 调试器使用同样的界面，学习和使用都方便。

多层分析。显示汇编窗口和 C 窗口，因此可以分析 C 代码、汇编代码，或同时分析。

具有一个丰富的命令集，对全局变量、模块、函数以及在各种层次上选择和建立代码分析区域，从而可以对复杂的应用作有效的代码分析。

具有广泛的统计功能，可以向用户提供发现程序代码中的瓶颈问题所需要的各种信息。

用户可以选择分析区域、统计数据的类型、以及排序标准，以保证用户指定的高效的统计显示。还可以用直方图来显示分析区域之间的数据的统计关系。

可以禁上分析区域的某些部分，使用其不参与统计。这有助于避免对标准库函数或代码中已充分优化过的部分再去作优化。

三、软件仿真器(Simulation)

TMS320 软件仿真器是一个利用 PC 的资源来仿真 DSP 资源的软件程序。它使用主机的处理器和存储器来仿真 TMS320 DSP 的微处理器和微计算机模式，从而进行软件开发和非实时的程序验证。

这是一种廉价的软仿真器，它可以在没有 DSP 硬件的情况下作 DSP 软件的开发和调试。它使用由 TMS320 宏汇编器/连接器或 ANSI C 编译产生的目标代码，由 I/O 指令的口地址所指定的输入和输出文件来仿真与处理器相连的 I/O 器件，可以按用户定义的时间间隔周期性地设置中断标志，仿真中断信号。在程序运行之前作初始化，设置断点及跟踪模式。程序一旦终止，则可对内部寄存器、程序和数据存储器作检查和修改。

软件仿真器的主要性能：

- 在 PC 或其它兼容机可执行用户的 DSP 程序；
- 修改和检查寄存器；
- 显示数据和程序寄存器；
- 外设、高速缓存（cache）、和流水（pipeline）时序的仿真；
- 提取指令周期时序，用以分析器件的性能；
- 设置断点；
- 跟踪累加器（ACC）、程序计数器（Program Counter）、辅助寄存器（Auxiliary Register）等；
- 单步指令；
- 在用户指定的时间产生中断；
- 对非法操作码和无效数据输入等提供出错信息；比较执行批处理文件中的命令；
- 用文件的方式来快速存储和调用仿真参数；
- 反汇编功能，以便对源语句作编辑和重新汇编；
- 存储器的内容可以同时显示为十六进制的 16-bit 值和汇编后的源代码；
- 多种执行模式（单个/多个指令计数、单个/多个周期计数、Until 条件、While 条件、FOR 循环计数、无限制地运行键入的 halt 等）；
- 周期计数；
- 在单步执行或运行模式下显示时钟周期数；
- 由等待状态来设置的外部产生模式，以便作准确的周期计数。

2.6.4 系统调试和评估工具

系统调试和评估工具包括：DSP 入门套件 DSK（DSP Starter Kit）、评估模块 EVM（Evaluation Module）和扩展的开发系统 XDS（Extended Development System）。

一、DSP 入门套件 DSK（DSP Starter Kit）

DSK 是 TI 公司提供给 TMS320 系列 DSP 的初学者的廉价的开发工具，用户可以用 DSK 来作 DSP 的实验，进行语音信号的采样、还原播放，系统控制等的应用，并可编写和运行实时源代码。

DSK 套件包括一块以 TMS320 DSP 为基础的板，专用的易于使用的汇编器/连接器和调试器，及相应的文本。还有打印口或 RS-232 的串口、电源接口、两个标准的 RCA 插口（提供模块信号的 I/O 口）、数量不等的高速 RAM。

二、评估模块 EVM（Evaluation Module）

TMS320 的评估模块（EVM）是廉价的开发板，用于器件评估、标准程序检查、以及有限的系统调试。EVM 是一个简单的 PC 插件，包括目标处理器、小容量的高速 RAM、有限的外设。

TMS320 EVM 所提供的基本功能有：

- 存储器和寄存的显示和修改；
- 汇编器/链接器；
- 软件单步运行和断点功能；；
- 主机装入/卸装功能；
- I/O 功能；
- 高级语言（HLL）调试接口。

三、扩展的 TMS320 开发系统 XDS(Extended Development System)

扩展开发系统（XDS），是可以用来进行系统级的集成调试，是进行 DSP 硬件和软件开发的最佳工具。

最早的 TMS3201X 和 TMS320C2X 芯片没有提供仿真的信号线，只能进行传统的电路仿真，这种仿真有诸多的限制，一般对使用的环境如连接线的长度，使用的电磁环境等都有比较高的要求，因而这种仿真器比较难做、不易使用。而后续推出的芯片，如 TMS320C2xx、TMS320C3x、TMS320C4x、TMS320C5x，TMS320C54x、TMS320C8x，TMS320C67xx 等均具有了标准的仿真信号线，于是有了先进的扫描仿真器。这类仿真器通过 DSP 芯片上提供的几个仿真引脚实现仿真功能。

扫描仿真消除了传统的电路仿真存在的问题，具有以下的一些优点：

- 没有电缆长度的传输问题。
- 仿真不介入系统，用户程序在目标系统的片内或片外存储器运行，不会因仿真引入额外的等待，也不会因仿真引起系统的不稳定。
- 对信号没有负载问题。
- 仿真系统对存储器使用的限制。
- 系统可以做到在线仿真。

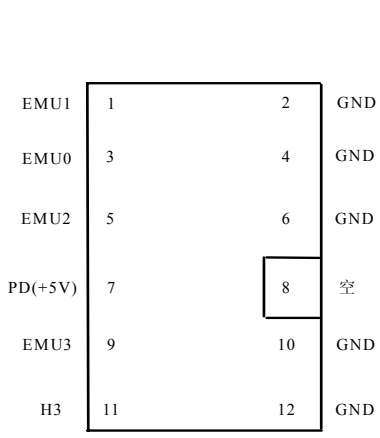
XDS510 是 TI 公司推出的用户 DSP 系统仿真和调试的主要的工具， XDS510 具有仿真器用户界面友好，以普通 PC 为基础的开发系统，可以对 C2xx、C3x、C4x、C5x、C54xx、C8x、C62xx、C67xx 等进行全速扫描仿真，用户可以使用 XDS510 来调试 C 程序、汇编语言程序、或是两者的结合。因此在 DSP 的开发上该仿真器得到广泛的应用，目前国内使用大部分为该种仿真器。

1．XDS510 的主要指标

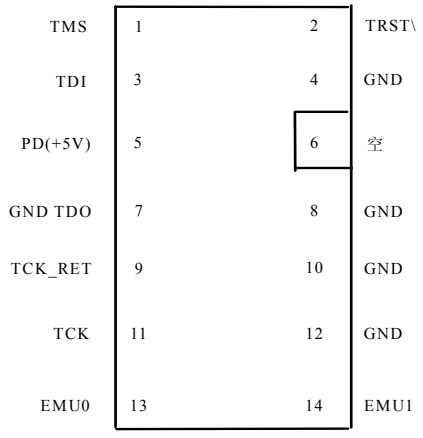
- 通过一个 14 脚（C3x 为 12 脚）的目标连接器，全速执行目标代码和监视目标系统中的器件；
- 并行处理 DSP 的全局运行/停止/断点；
- 高级语言（HLL）调试接口；
- 最多可达 200 个断点的软件断点/跟踪和定时；
- 对所有程序和数据地址作硬件断点/跟踪；
- 单步执行；
- 与 C/汇编源代码接口与调试；
- 所有寄存器和存储器的装入、检查和修改；
- 时钟周期执行时间的标准程序检查。

2．仿真器接口及仿真线的定义

除 TMS320c3x 其仿真头为 12 线之外（如图 2-14），其它仿真器的仿真信号均采用 JTAG 标准 IEEE 1194.1。仿真头一般采用 14 根信号线，14 线仿真头如图 2-11 所示，表 2-2 为其仿真头的定义。其中 TDO 信号与时钟的下降沿对齐，TMS 和 TDI 在 TCK 时钟的上升沿取样。



TMS320C3x



TMS320C54x

图 2-14 JATG 仿真口的定义

表 2-2 JTAG 仿真信号定义

仿真头信号	仿真头状态	DSP 芯片状态	信号说明
TMS	输出	输入	JTAG 测试方法选择
TDI	输出	输入	JTAG 测试数据输入
TDO	输入		JTAG 测试数据输出
TCK	输出	输入	JTAG 测试时钟
TRST\	输入	输入	JTAG 测试复位
EMU0	输入	输入/输出	仿真脚 0

EMO1	输入	输入/输出	仿真脚 1
PD	输入	输出	存在检查，在目标系统中应接至+5V
TCK_RET	输入	输出	JTAG 测试时钟返回，测试时时钟输入至仿真头。

第三章　数字滤波器的 DSP 实现

在无线通信中，数字滤波器得到广泛应用，如梳状滤波器、单边带滤波器及各种带通滤波器。本章将扼要介绍 FIR、IIR 滤波器的基本原理和设计方法，讨论 FIR、IIR 滤波器的 MATLAB、DSP 实现。

3.1 数字滤波的基本概念

数字滤波器和模拟滤波器都具有一定的频率选择特性。但不同的是，数字滤波器是针对离散采样信号或离散序列设计的，通常由专用集成电路（ASIC）或数字信号处理器（DSP）实现。

与模拟滤波器相比，数字滤波器具有可靠、准确、灵活、对温度和增益的敏感度低等优点，并且数字滤波器能满足严格的幅度和相位特性。采用数字信号处理器实现数字滤波器时，通过修改参数就可方便地改变滤波器的带宽、频响，甚至滤波器类型。

采用数字信号处理器实现的数字滤波器，信号输入在数字信号处理器前，先由 A/D 变换器对模拟信号采样，得到离散的输入样值，再由数字信号处理器完成数字滤波，最后经 D/A 变换器输出得到模拟信号。

数字滤波器的设计包括以下步骤：

1. 生成符合设计要求的传输函数；
2. 将传输函数转化为滤波器结构图；
3. 当使用精度有限的处理器时，滤波器系数的精度受到处理器字长的限制，必须进行数据误差控制；
4. 通过软件和硬件实现滤波器。

我们主要讨论两类滤波器的设计：有限冲激响应（FIR）滤波器和无限冲激响应（IIR）滤波器。

首先介绍一些和数字滤波器有关的基本概念。

3.1.1 离散信号

离散信号是由一系列样值组成的序列，其表达式如下：

$$\{x(n)\} = \{x(1), x(2), x(3), \cdots\} \tag{3-1}$$

其中 $x(n)$表示 n 时刻的采样值。

单位取样序列是一种重要的离散序列，定义如下：

$$\delta(n)=\begin{cases}1 & n=0\\ 0 & n\neq 0\end{cases} \tag{3-2}$$

将单位取样序列右移 m 个单位可得到一个移位序列，该移位序列也是一个单位取样序列，可定义为：

$$\delta(n-m)=\begin{cases}1 & n=m\\ 0 & n\neq m\end{cases} \tag{3-3}$$

离散序列 $x(n)$可用单位取样序列及其离散样值表示，表达式如下：

$$x(n)=\sum_{k=-\infty}^{+\infty}x(k)\delta(n-k) \tag{3-4}$$

3.1.2 线性时不变系统

我们讨论的数字系统是线性时不变系统，它的两个重要特性是：满足叠加原理和具有时不变特性。

在数学上，系统定义为：将输入序列 $x(n)$映射成输出序列 $y(n)$的唯一变换或运算。记为：

$$x(n)\ \rightarrow\ y(n)$$

对于线性系统，如果将输入乘以一个常数，那么相应的输出也会等于原输出乘以该常数，即：

$$ax(n)\ \rightarrow ay(n)$$

设 $y_1(n)$和 $y_2(n)$分别是对于输入 $x_1(n)$和 $x_2(n)$的响应，则：

$$ax_1(n)+bx_2(n)\ \rightarrow\ ay_1(n)+by_2(n) \tag{3-5}$$

线性系统的总输出等于对单个输入响应的总和，即满足叠加原理。

时不变系统有如下特征：如果 $y(n)$是系统对 $x(n)$的响应，则 $y(n\text{-}m)$是系统对 $x(n\text{-}m)$的响应，即：

$$x(n-m)\ \rightarrow\ y(n-m) \tag{3-6}$$

3.1.3 卷积

若将单位取样信号作为线性系统的输入，则其输出 $h(n)$称为该系统的单位冲激响应，即：$\delta(n)\rightarrow h(n)$。

当输入被延迟后，输出同样延迟，即：$\delta(n-m) \to h(n-m)$。

若输入的单位取样信号乘以一个常数，那么输出也会同样乘以该常数，即：

$$x(m)\delta(n-m) \to x(m)h(n-m)$$

一般地，输入信号可以表示为一系列延迟的单位取样序列的加权和，如式(3-7)所示：

$$x(n) = \sum_{m=-\infty}^{+\infty} x(m)\delta(n-m) \tag{3-7}$$

由于我们研究的系统是线性系统，根据叠加原理，将系统对各独立的单位取样序列的响应取加权和，就可以得到系统的输出：

$$y(n) = \sum_{m=-\infty}^{+\infty} x(m)h(n-m) \tag{3-8}$$

若该系统是因果系统，则响应 $y(n)$不可能先于输入单位取样序列出现，因此在式(3-8)中，必须满足 $n \geqslant m$，这样我们得到：

$$y(n) = \sum_{m=-\infty}^{n} x(m)h(n-m) \tag{3-9}$$

作进一步的变换，令 $k = n - m$，式(3-9)变为：

$$y(n) = \sum_{k=0}^{\infty} h(k)x(n-k) \tag{3-10}$$

3.2 FIR 滤波器

FIR 滤波器的幅度特性可设计成多种多样，同时还可保证精确、严格的相位特性。FIR 滤波器也可以采用 FFT 变换的方法来设计，从而大大提高运算效率。这些优点使得 FIR 滤波器的运用越来越广泛。

3.2.1 FIR 滤波器的基本原理和设计方法

一、有限冲激响应（FIR）滤波器

FIR 滤波器只存在有限个 $h(n)$，即：

$$y(n) = \sum_{k=0}^{N} h(k)x(n-k) = \sum_{m=0}^{N} h(m)x(n-m) \tag{3-11}$$

因此其系统函数如下：

$$H(z)=\sum_{m=0}^{N}h(m)z^{-m}=h(0)+h(1)z^{-1}+\cdots h(N)z^{-N} \tag{3-12}$$

由式(3-12)，我们得到：

$$Y(z)=H(z)X(z)=h(0)X(z)+h(1)z^{-1}X(z)+\cdots+h(N)z^{-N}X(z) \tag{3-13}$$

同样的，传输函数 $H(z)$也可以表示为：

$$H(z)=\frac{h(0)z^{N}+h(1)z^{N-1}+\cdots+h(N)}{z^{N}} \tag{3-14}$$

由式(3-14)可知，系统只在原点处存在极点，这使得 FIR 滤波器具有稳定性。FIR 滤波器不存在反馈，不需要知道过去的输出结果。FIR 滤波器的另一个重要特性是具有线性相位，可以保证系统的相移和频率成比例，达到无失真的传输。FIR 滤波器的结构如图 3-1 所示。

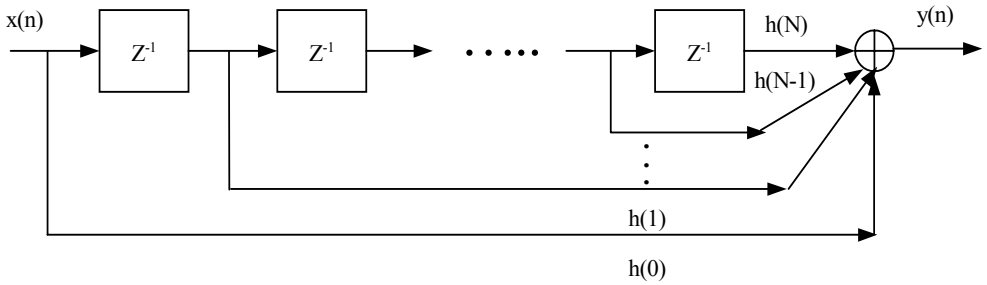


图 3-1 FIR 滤波器的结构

二、FIR 滤波器的设计方法

设计 FIR 滤波器的基本方法是用一个有限级数的傅里叶变换去逼近所要求的滤波器响应。首先，将要求设计的响应用傅里叶级数表示为：

$$H_d(\gamma)=\sum_{n=-\infty}^{+\infty}C_n e^{jn\pi\gamma} \qquad |n|<\infty \tag{3-15}$$

其中,γ 是归一化频率变量，$\gamma = f / f_N$。f_N=f_S/2 为奈奎斯特频率，f_S 为采样频率。$\omega T = 2\pi f / f_S = \pi\gamma$。$H_d(\gamma)$是所要求的传输函数。系数 C_n选择原则是在均方误差最小条件下使 $H(z)$尽量逼近 $H_d(\gamma)$。系数 C_n由式(3-16)求得：

$$C_n=\frac{1}{2}\int_{-1}^{1}H_d(\gamma)e^{-jn\pi\gamma}d\gamma \tag{3-16}$$

假设 $H_d(\gamma)$为偶函数，则：

$$C_n = \int_0^1 H_d(\gamma)\cos n\pi\gamma d\gamma \qquad n \geq 0$$，且 $C_{-n}=C_n$ (3-17)

传输函数 $H_d(\gamma)$ 有无限多个系数 C_n，而实际的滤波器系数个数有限，因此，我们对其表达式作截短，得到近似的传输函数：

$$H_d(\gamma) = \sum_{n=-Q}^{Q} C_n e^{jn\pi\gamma} \tag{3-18}$$

在 $|\gamma| < 1$、Q 为有限正数时，令 $z = e^{j\pi\gamma}$，则

$$H_d(\gamma) = \sum_{n=-Q}^{Q} C_n z^n \tag{3-19}$$

可见上述近似传输函数的冲激响应由一系列的系数 C_{-Q}、C_{-Q+1}、…、C_0、…、C_{Q+1}、C_Q 决定。观察式(3-19)，发现当 $n>0$ 时对应的 $C_n z^n$ 项代表的是非因果的滤波器，即输出先于输入。要得到 n 时刻的输出响应需用到 n+1 时刻的输出响应，将式(3-19)乘以 z^{-Q} 项，得到：

$$H(z) = z^{-Q}H_d(z) = z^{-Q}\sum_{n=-Q}^{Q} C_n z^n = \sum_{n=-Q}^{Q} C_n z^{n-Q}$$

令 i= —(n —Q)，作变量替换得到：

$$H(z) = \sum_{i=2Q}^{0} C_{Q-i} z^{-i} = \sum_{i=0}^{2Q} C_{Q-i} z^{-i} \qquad 0 \leq i \leq 2Q$$

进一步，令 $h_i = C_{Q-i}$，$H(z)$的表达式如下：

$$H(z) = \sum_{i=0}^{N} h_i z^{-i} \qquad 0 \leq i \leq N \tag{3-20}$$

当 $N=2Q$ 时，$h_0=C_Q$、$h_1=C_{Q-1}$、$h_2=C_{Q-2}$、…、$h_Q=C_0$、$h_{Q+1}=C_1$、…、$h_{2Q-1}=C_{-Q+1}$、$h_{2Q}=C_{-Q}$。

当 $N+1=2Q+1$ 时，系数 h_i 关于 h_Q 对称，即 $h_i = C_{Q-i}$ 且 $C_n = C_{-n}$。下面以 $Q=5$ 的 11 个系数的滤波器为例，其系数 h_i 如下：

$h_0=h_{10}=C_5$

$h_1=h_9=C_4$

$h_2=h_8=C_3$

$$h_3=h_7=C_2$$
$$h_4=h_6=C_1$$
$$h_5=C_0$$

式(3-21)所示的卷积公式决定了 FIR 滤波器的响应由 N+1 项构成。

$$y(n)=\sum_{k=0}^{N}h(k)x(n-k) \qquad (3\text{-}21)$$

三、带通、高通及带阻 FIR 滤波器的设计

带通、高通及带阻 FIR 滤波器的表达式可由低通 FIR 滤波器的表达式经过变换得到。低通 FIR 滤波器的表达式如下：

$$C_n=\frac{\sin[(f_c/f_N)n\pi]}{n\pi} \qquad (3\text{-}22)$$

图 3-2 显示了如何由两个低通 FIR 滤波器来得到带通 FIR 滤波器，其系数可以由两个低通滤波器的系数相减得到：

$$C_n=\frac{\sin[(f_{c2}/f_N)n\pi]}{n\pi}-\frac{\sin[(f_{c1}/f_N)n\pi]}{n\pi} \qquad (3\text{-}23)$$

其中 f_{c1} 和 f_{c2} 分别是两个低通滤波器的截止频率，f_N 为采用频率的一半。

图 3-3 显示了如何由一个低通 FIR 滤波器和一个幅频响应恒为 1 的系统来得到高通 FIR 滤波器。由于 $\delta(n)$ 所对应的幅频响应恒为 1，高通 FIR 滤波器的系数表达式如下：

$$C_n=\delta(n)-\frac{\sin[(f_c/f_N)n\pi]}{n\pi} \qquad (3\text{-}24)$$

同样的，带阻 FIR 滤波器的系数可以由 $\delta(n)$ 和带通滤波器的系数相减得到。

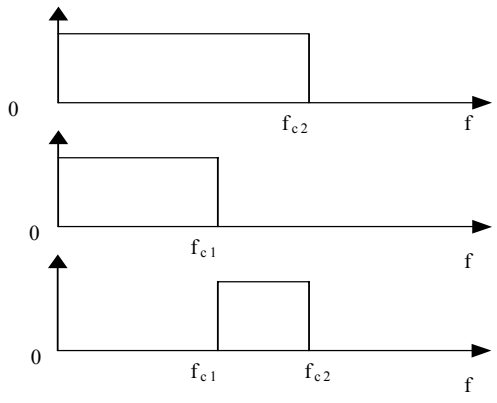


图 3-2　带通 FIR 滤波器示意图

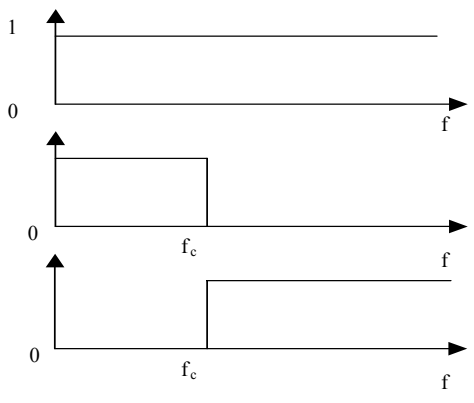


图 3-3　高通 FIR 滤波器示意图

3.2.2 FIR 滤波器的 MATLAB 实现

一、fir1 函数

功能：设计标准频率响应的基于窗函数的 FIR 滤波器

语法：b=fir1(*n*,W_n)

b=fir1(*n*,W_n,'*ftype*')

b=fir1(*n*,W_n,*Window*)

b=fir1(*n*,W_n,'*ftype*',*Window*)

说明：fir1 函数可以实现加窗线性相位 FIR 数字滤波器设计，它可设计出标准的低通、高通、高通和带阻滤波器。

b=fir1(*n*,W_n)可得到 *n* 阶低通，截止频率为 W_n 的汉明加窗线性相位 FIR 滤波器，$0 \le W_n \le 1$，W_n=1 相当于 0.5f_s。滤波器系数包含在 b 中，可表示为：

$$b(z) = b(1) + b(2)z^{-1} + \cdots + b(n+1)z^{-n}$$

当 $\mathbf{W_n} = [W_1 \quad W_2]$ 时，fir1 函数可得到带通滤波器，其通带为 $W_1 < w < W_2$。

b=fir1(*n*,W_n,'*ftype*')可设计高通和带通滤波器，参数 *ftype* 决定滤波器的类型。

当 *ftype=high* 时，设计高通 FIR 滤波器；当 *ftype=stop* 时，设计带阻 FIR 滤波器。

在设计高通和带阻滤波器时，由于对奇次阶的滤波器，其在 Nyquist 频率处的频率响应为零，不适合构成高通和带阻滤波器。因此 fir1 函数总是使用阶数为偶数的滤波器，因此当输入的阶数为奇数时，fir1 函数会自动将阶数增加 1。

b=fir1(*n*,W_n,*Window*)利用参数 *Window* 来指定滤波器采用的窗函数类型。其默认值为汉明窗。

b=fir1(*n*,W_n,'*ftype*',*Window*)可利用 *ftype* 和 *Window* 参数，设计各种滤波器。

例 3-1 设计一个 48 阶 FIR 带通滤波器，通带为 $0.3 < w < 0.6$。

［**程序 3-1**］带通 FIR 滤波器的 MATLAB 实现。

```
b=fir1(48,[0.3 0.6]);
freqz(b,1,512)
```

这可得到如图 3-4 所示的带通 FIR 滤波器特性。

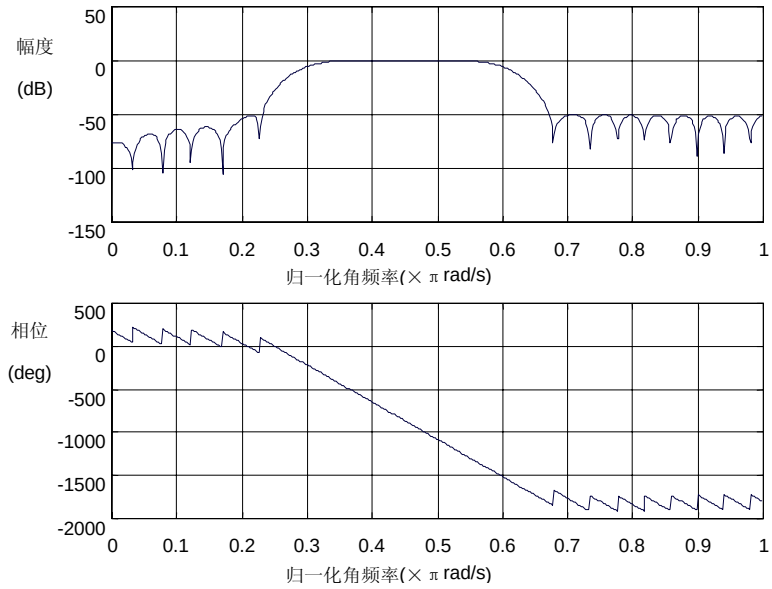


图 3-4　带通 FIR 滤波器的特性

二、fir2 函数

功能：设计任意频率响应的基于窗函数的 FIR 滤波器

语法：　b=fir2(*n*,**f**,**m**)

b=fir2(*n*,**f**,**m**,*Window*)

b=fir2(*n*,**f**,**m**,*npt*)

b=fir2(*n*,**f**,**m**,*npt*,*Window*)

b=fir2(*n*,**f**,**m**,*npt*,*lap*)

b=fir2(*n*,**f**,**m**,*npt*,*lap*,*Window*)

说明：fir2 函数用于设计有任意频率响应的加窗 FIR 滤波器，对标准的低通、带通、高通和带阻滤波器的设计可使用 fir1 函数。

b=fir2(*n*,**f**,**m**)可设计出一个 *n* 阶的 FIR 滤波器，其滤波器的频率特性由参数 **f** 和 **m** 决定。参数 **f** 为频率点矢量，且 $f \in [0,1]$，$f=1$ 对应于 $0.5f_S$。矢量 **f** 按升序排列，且第一个元素必须为 0，最后一个必须为 1，并可以包含重复的频率点。矢量 **m** 中包含了与 **f** 向对应的期望得到的滤波器的幅度。

b=fir2(*n*,**f**,**m**,*Window*)中用参数 *Window* 来指定使用的窗函数类型，默认值为汉明窗。

b=fir2(*n*,**f**,**m**,*npt*)中用参数 *npt* 来指定 fir2 函数对频率响应进行内插的点数。

b=fir2(*n*,**f**,**m**,*npt*,*lap*)中用参数 *lap* 来指定 fir2 载重复频率点附近插入的区域大小。

例 3-2　设计一个 30 阶的低通 FIR 滤波器。

［程序 3-2］低通 FIR 滤波器的 MATLAB 实现。

```
f=[0 0.6 0.6 1];
m=[1 1 0 0];
```

```
b=fir2(30,f,m);
[h,w]=freqz(b,1,128);
plot(f,m,w/pi,abs(h));
```

结果如图 3-5 所示，从中可以对比理想滤波器与实际滤波器的差异。

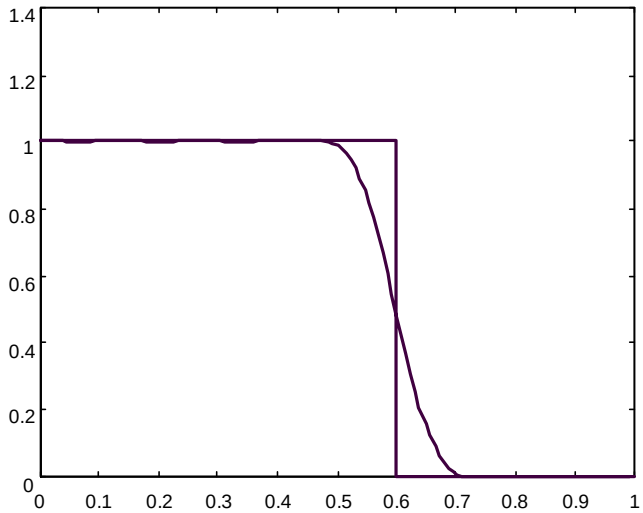


图 3-5 理想和实际滤波器特性

3.2.3 FIR 滤波器的 DSP 实现

［程序 3-3］该程序完成基于 FIR 的带通滤波。

设计参数：

- 采样速率为 16000 样点/秒
- 滤波器阶数 192
- 带宽为 300Hz~3300Hz
- 带内波纹 ＜0.5dB

编程说明：程序中的滤波器系数由 MATLAB 中的 fir1 函数产生。该程序利用了 TMS320C54XX 用一条指令完成乘法、累加和数据移位（MACD）的特点，减少了滤波器的运算时间。

```
;---------------------------------------------------
;       version:        1.0
;       data:           july-21-2000
;       authors:        ren guo chun
;       comment:          original created
;       cpu type:       ti_tms32054xx
;---------------------------------------------------
```

```
;---------------------------------------------------
;       compiler:       tms320c54x assembler
;       version:        1.02 (pc)
;       include files:  .mmregs
;---------------------------------------------------

;---------------------------------------------------
;       functional description:
;       this subroutine perform FIR filtering using
;       MACD instruction.
;       accumlator A (filter out)=h(n)*x(n-i)
;       for i=0,1,2...191
;
;---------------------------------------------------

;---------------------------------------------------
;       activation
;       activation example:
;       call    fir_filter
;---------------------------------------------------

;---------------------------------------------------
;       External Data
        .ref    CRAM2
;---------------------------------------------------

;---------------------------------------------------
;       temporary data:
;       dp=4
;       cram2
;       cram3
;       cram4
;---------------------------------------------------

;---------------------------------------------------
;       input data structure
;       fir_i_b0:   real_data_buf           ;buf_l=192
;---------------------------------------------------
```

```
;-------------------------------------------------
;           inut data:          cram2
;           data format:        16-bit data
;-------------------------------------------------

;-------------------------------------------------
;           output data:        cram4
;           data format:        16-bit data
;-------------------------------------------------


fir_and_hiblt
            STM     #FIR_I_B0,AR3               ;load buffer pointer
            LD      CRAM2,A
            STL     A,*AR3                      ;load sample

            STM     #FIR_I_B0+192-1,AR3         ;load buffer pointer

            RPTZ    A,#192-1                    ;clr A and repeat
            MACD    *AR3-,fir_tab,A             ;mpy and add and mov
            STH     A,1,CRAM4                   ;save filter value

            RET


fir_tab:
            .word   0000cH,00001H,00002H,0000eH,0001aH,00010H,0fff0H,0ffd7H
            .word   0ffddH,0fff6H,00000H,0ffefH,0ffe1H,0ffebH,0fffeH,0fffaH
            .word   0ffe3H,0ffdbH,0fff1H,00003H,0fff5H,0ffdaH,0ffdfH,00000H
            .word   0000cH,0fff1H,0ffd9H,0fff0H,00018H,00017H,0fff0H,0ffe4H
            .word   0000fH,00037H,00021H,0fff3H,0fffcH,00039H,00054H,00026H
            .word   0fffaH,0001fH,00067H,00067H,0001fH,00003H,00048H,0008bH
            .word   00063H,00009H,0000cH,0006fH,00098H,0003eH,0ffe3H,00014H
            .word   00086H,0007eH,0fff3H,0ffafH,00018H,00083H,00031H,0ff84H
            .word   0ff76H,00017H,00059H,0ffafH,0fefcH,0ff46H,00011H,00000H
            .word   0fef9H,0fe6fH,0ff32H,00007H,0ff71H,0fe19H,0fdfcH,0ff54H
            .word   0fffbH,0fea0H,0fd15H,0fdcdH,0ffdeH,0ffefH,0fd58H,0fbd3H
            .word   0fe3fH,0017cH,0ffe8H,0fa27H,0f8d1H,00286H,011f0H,0197eH
            .word   011f0H,00286H,0f8d1H,0fa27H,0ffe8H,0017cH,0fe3fH,0fbd3H
```

```
.word   0fd58H,0ffefH,0ffdeH,0fdcdH,0fd15H,0fea0H,0fffbH,0ff54H
.word   0fdfcH,0fe19H,0ff71H,00007H,0ff32H,0fe6fH,0fef9H,00000H
.word   00011H,0ff46H,0fefcH,0ffafH,00059H,00017H,0ff76H,0ff84H
.word   00031H,00083H,00018H,0ffafH,0fff3H,0007eH,00086H,00014H
.word   0ffe3H,0003eH,00098H,0006fH,0000cH,00009H,00063H,0008bH
.word   00048H,00003H,0001fH,00067H,00067H,0001fH,0fffaH,00026H
.word   00054H,00039H,0fffcH,0fff3H,00021H,00037H,0000fH,0ffe4H
.word   0fff0H,00017H,00018H,0fff0H,0ffd9H,0fff1H,0000cH,00000H
.word   0ffdfH,0ffdaH,0fff5H,00003H,0fff1H,0ffdbH,0ffe3H,0fffaH
.word   0fffeH,0ffebH,0ffe1H,0ffefH,00000H,0fff6H,0ffddH,0ffd7H
.word   0fff0H,00010H,0001aH,0000eH,00002H,00001H,0000cH,00000H
.end
```

3.3 IIR 滤波器的 DSP 实现

IIR 滤波器与 FIR 滤波器相比较，具有相位特性差的缺点，但是其结构简单、运算量小，因此也得到了比较广泛的应用。

3.3.1 IIR 滤波器的基本原理和设计方法

一、无限冲激响应（IIR）滤波器

式(3-25)给出了一个通用的输入－输出关系：

$$y(n)=\sum_{k=0}^{N}a_k x(n-k)-\sum_{j=1}^{M}b_j y(n-j)\qquad n\geq 0 \tag{3-25}$$

若所有的系数b_j等于0，则式(3-25)表示FIR滤波器。若至少有一个系数b_j不等于0，则式(3-25)具有递归性，即n时刻的输出不仅由 $n-N$ 到 n 时刻的输入决定，还和 $n-M$ 到 $n-1$ 时刻的输出有关，这时式(3-25)表示了一个无限冲激响应（IIR）滤波器。对于IIR滤波器，计算$y(n)$需要$y(n\text{-}1)$的值，同样地，计算$y(n\text{-}1)$需要$y(n\text{-}2)$的值。式(3-26)给出了$y(n)$的表达式：

$$\begin{aligned}y(n)=&a_0x(n)+a_1x(n-1)+a_2x(n-2)+\cdots+a_Nx(n-N)\\&-b_1y(n-1)-b_2y(n-2)-\cdots-b_My(n-M)\end{aligned} \tag{3-26}$$

在零初始条件下，对式(3-26)作Z 变换，得到：

$$\begin{aligned}Y(z)=&a_0X(z)+a_1z^{-1}X(z)+a_2z^{-2}X(z)+\cdots+a_Nz^{-N}X(z)\\&-b_1z^{-1}Y(z)-b_2z^{-2}Y(z)-\cdots-b_Mz^{-M}Y(z)\end{aligned}$$

假设N=M，则传输函数为：

$$H(z)=\frac{Y(z)}{X(z)}=\frac{N(z)}{D(z)}=\frac{a_0+a_1z^{-1}+a_2z^{-2}+\cdots+a_Nz^{-N}}{1+b_1z^{-1}+b_2z^{-2}+\cdots+b_Nz^{-N}} \tag{3-27}$$

式(3-27)也可写成：

$$H(z)=\frac{a_0z^N+a_1z^{N-1}+a_2z^{N-2}+\cdots+a_N}{z^N+b_1z^{N-1}+b_2z^{N-2}+\cdots+b_N}=C\prod_{i=1}^{N}\frac{z-z_i}{z-p_i} \tag{3-28}$$

式(3-28)具有N个零点和N个极点。如果有极点位于单位圆外会导致系统不稳定。而对于FIR滤波器，所有的系数b_j均为0时，不存在极点，也不会造成系统的不稳定。对于IIR滤波器，系统稳定的条件如下：

若$|p_i|<1$，当$n\to\infty$时$h(n)\to 0$，系统稳定；

若$|p_i|>1$，当$n\to\infty$时$h(n)\to\infty$，系统不稳定；

IIR滤波器的结构具有多种形式，包括直接形式I、直接形式II、并联形式和级联形式。

二、IIR滤波器的设计

IIR滤波器的设计可以利用模拟滤波器原型，借鉴成熟的模拟滤波器的设计结果，然后进行双线性变换，将模拟滤波器变换为满足预定指标的数字滤波器，即将模拟滤波器的传输函数$H(s)$变换为数字滤波器的传输函数$H(z)$。双线性变换将s平面上的虚轴唯一地映射为z平面上的单位圆。如果$H(s)$所表示的是一个稳定系统，即其所有极点都在s平面的左半面，则对应$H(z)$的所有极点都在z平面的单位圆内，该$H(z)$表示的是一个稳定的数字滤波器。

对模拟滤波器设计的结论如下：

1. 对于比特沃思滤波器，其幅度响应在通带内具有最平特性。
2. 由于切比雪夫滤波器允许在通带内具有波纹，其阶数要小于比特沃思滤波器。
3. 椭圆滤波器的阶数最小，且在通带和阻带具有等波纹性。

将模拟滤波器转换为数字滤波器最常用的方法是双线性变换（BLT），其作用是完成s平面到z平面的映射。

双线性变换的规则如式(3-29)或式(3-30)所示：

$$s=\frac{z-1}{z+1} \tag{3-29}$$

$$z=\frac{1+s}{1-s} \tag{3-30}$$

双线性变换具有以下几条基本性质：

1. s平面的左半平面映射到z平面的单位圆内；
2. s平面的右半平面映射到z平面的单位圆外；
3. s平面上的jw轴映射到z平面的单位圆上。

考虑s平面上的虚轴映射为z平面上的单位圆，令：

$$s=jw_A \tag{3-31}$$

及：

$$z = e^{jw_D T} \tag{3-32}$$

其中 w_A 代表模拟频率，与数字频率 w_D 相对应。由式(3-29)、式(3-31)以及式(3-32)，有：

$$jw_A = \frac{e^{jw_D T} - 1}{e^{jw_D T} + 1} = \frac{e^{jw_D T/2}(e^{jw_D T/2} - e^{-jw_D T/2})}{e^{jw_D T/2}(e^{jw_D T/2} + e^{-jw_D T/2})} \tag{3-33}$$

对上面的方程式求解，得到：

$$w_A = \tan\frac{w_D T}{2} \tag{3-34}$$

式(3-35)给出了模拟频率和数字频率之间的对应关系：

$$H(s)\big|_{s=jw_A} = H(z)\big|_{z=e^{jw_D T}} \tag{3-35}$$

双线性变换提供了模拟频率与数字频率间一到一的映射。当 w_A 在0到1之间时，对应的 w_D 在0到 $w_s/4$ 之间；当 w_A 大于1时，其对应的 w_D 在 $w_s/4$ 到 $w_s/2$ 之间。双线性变换会造成频率失真，通常采用预畸变来补偿频率失真。

下面是采用双线性变换由模拟滤波器得到数字滤波器的步骤：

1、选择一个合适的模拟传输函数*H*(*s*)；

2、通过下式对预定的数字频率 w_D 进行预畸变，得到相应的模拟频率

$$w_A = \tan\frac{w_D T}{2}$$

3、对*H*(*S*)中的频率进行换算

$$H(s)\big|_{s=s/w_A} \tag{3-36}$$

4、用式(3-29)或式(3-37)计算*H*(*z*)

$$H(z) = H(s/w_A)\big|_{s=(z-1)/(z+1)} \tag{3-37}$$

3.3.2 IIR 滤波器的 MATLAB 设计

一、butter 函数

功能：Butterworth 滤波器的设计

语法： [b,a]=butter(n,W_n)；

[b,a]=butter(n,W_n,'*ftype*')；

说明：butter 函数可以设计低通、带通、高通和带阻数字滤波器，其优点为：能使通带内的幅度响应最大限度的平坦，但是会损失截止频率处的下降斜度。在期望通带平滑的情况下，可以使用 butter 函数，在期望下降斜度大的场合，应使用 Chebyshev 滤波器。

[b,a]=butter(n,W_n)可设计出截止频率为 W_n 的 n 阶低通 Butterworth 滤波器，$0 \le W_n \le 10$，W_n=1 相当于 0.5f_S。当 $\mathbf{W_n}=[W_1 \quad W_2]$（$W_1<W_2$）时，butter 函数产生一个 2$n$ 阶的数字带通滤波器，其通带为 $W_1 < w < W_2$。

[b,a]=butter(n,W_n,'*ftype*')中，参数 *ftype* 指定滤波器的类型，当 *ftype=high* 时，可设计出截止频率为 W_n 的高通滤波器；当 *ftype=stop* 时，可设计出带阻滤波器，这时 $\mathbf{W_n}=[W_1 \quad W_2]$，且阻带为 $W_1 < w < W_2$。

二、Buttord 函数

功能：Butterworth 滤波器阶数的选择

语法：[n,W_n]=buttord(W_p,W_s,R_p,R_s)

说明：buttord 可在给定滤波器性能的情况下，选择数字 Butterworth 滤波器的最小阶数，其中 W_p 和 W_s 分别是通带和阻带的截止频率，其值为 $0 \le W_p$(或W_s) ≤ 1，当值为 1 时表示 0.5f_S。R_p 和 R_s 分别是通带和阻带内的纹波系数。

[n,W_n]=buttord(W_p,W_s,R_p,R_s)可得到数字 Butterworth 滤波器的最小阶 n，并使得在通带（0，W_p）内纹波系数小于 R_p，在阻带（W_s，1）内衰减系数大于 R_s。Buttord 还得到了截止频率 W_n，这样利用 butter 函数可得到满足性能指标的滤波器。

利用 buttord 函数，还可以得到高通、带通和带阻滤波器的阶数。当 W_p>W_s 时为高通滤波器；当 W_p、W_s 为二元矢量时，若 W_p<W_s，则为带通或带阻滤波器，此时 W_n 也为二元矢量。

例 3-3 设计一带通 IIR 滤波器，通带范围为 100~250Hz，带外 50Hz 处衰减 30dB。

［程序 3-4］带通 IIR 滤波器的 MATLAB 实现。

```
Wp=[100 250]/500;
Ws=[50 300]/500;
[n,Wn]=buttord(Wp,Ws,3,30);
[b,a]=butter(n,Wn);
freqz(b,a,512,1000);
```

得到的滤波器特性如图 3-6 所示。

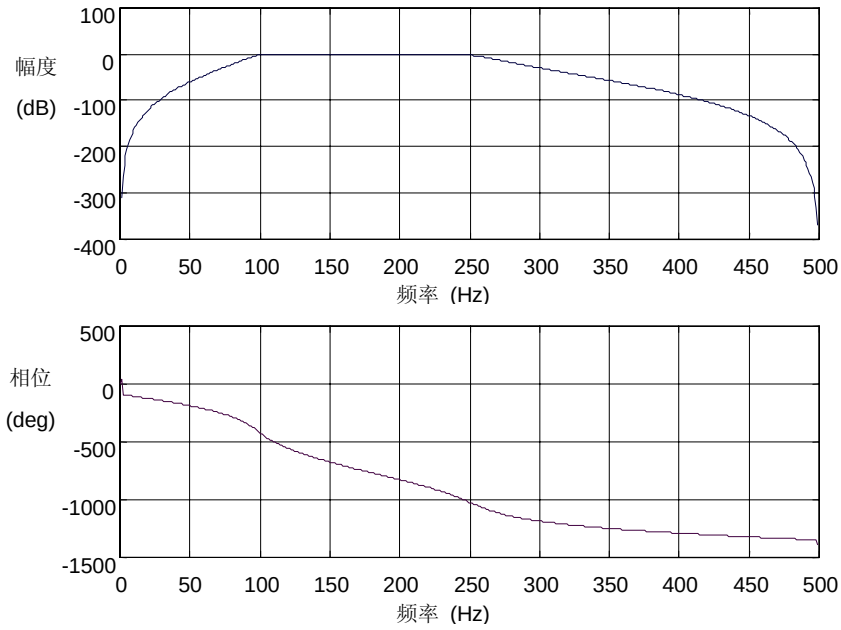


图 3-6　带通滤波器的特性

三、cheby1 函数

功能：ChebyshevI 型滤波器设计（通带等波纹）。

语法：　[b,a]=cheby1(n,R_p,W_n);

[b,a]=cheby1(n,W_n,R_p,'*ftype*');

说明：cheby1 函数可设计低通、带通、高通和带阻数字 ChebyshevI 型滤波器，其通带内为等波纹，阻带内为单调。ChebyshevI 型滤波器的下降斜度要比 II 型大，代价是通带内的纹波较大。

[b,a]=cheby1(n,R_p,W_n)可设计 n 阶低通数字 ChebyshevI 型滤波器，其截止频率为 W_n，通带内的波纹由 R_p 确定。当 $\mathbf{W_n}=[W_1 \quad W_2]$（$W_1<W_2$）时，cheby1 函数产生一个 $2n$ 阶的数字带通滤波器，其通带为 $W1<w<W2$。

[b,a]=cheby1(n,W_n,R_p,'*ftype*')可设计高通或带阻滤波器，*ftype* 的定义与 butter 函数相同。

四、cheb1ord 函数

功能：ChebyshevI 型滤波器阶的选择。

语法：[n,W_n]=cheb1ord(W_p,W_s,R_p,R_s)

说明：在给定滤波器性能的情况下，选择 ChebyshevI 型滤波器的最小阶数，其中 W_p 和 W_s 分别是通带和阻带的截止频率，其值 $0 \le W_p$（或 W_s）≤ 1，R_p 和 R_s 分别是通带和阻带的纹波系数。

由[n,Wn]=cheb1ord(Wp,Ws,Rp,Rs)可以得到数字 ChebyshevI 型滤波器的最小阶数，并使其在通带（0,W_p）内纹波系数小于 R_p，在阻带（W_s，1）内衰减系数大于 R_s，cheb1ord

还得到了截止频率 W_n。再利用 cheby1 函数就可以得到满足指定性能的滤波器。

cheb1ord 还可以得到高通、带通和带阻滤波器的阶数。

五、cheby2 函数

功能：ChebyshevII 型滤波器设计（阻带等波纹）。

语法： [b,a]=cheby2(*n*,R_s,W_n);

[b,a]=cheby2(*n*,W_n,R_s,'*ftype*');

说明：cheby2 函数与 cheby1 函数基本相同，区别只在于 cheby2 函数设计的滤波器其通带内为单调，阻带内为等波纹，由 R_s 指定阻带内的波纹。

六、cheb2ord 函数

功能：ChebyshevII 型滤波器阶的选择。

语法：[*n*,W_n]=cheb2ord(W_p,W_s,R_p,R_s)

说明：cheb2ord 函数与 cheb1ord 函数类似，给出了 ChebyshevII 型滤波器的阶数和截止频 W_n。

3.3.3 IIR 滤波器的 DSP 实现

[程序 3-5]该程序完成基于 IIR 的低通滤波。

设计参数：

- 采样速率为 7200 样点/秒
- 带宽为 0～300Hz

编程说明：在数值运算程序设计过程中，首先要进行小数点定标。在 *Q*15 小数点定标规则中 7FFFH 表示数值 1，1000H 表示数值 0.125，8000H 表示数值-1。在 *Q*12 小数点定标规则中，7800H 表示数值 7.5。在以下程序中，采用 *Q*15 小数点定标规则。为保证系统稳定，要求 *K*1+*K*2=1。

```
;---------------------------------------------------
;       version:            1.0
;       data:               jan-25-1999
;       authors:            Ren Gou Chun
;       comment:              orignal created
;       cpu type:           ti_tms32054xx
;---------------------------------------------------

;---------------------------------------------------
;       compiler:           tms320c54x assembler
;       version:            1.02 (pc)
;       include files:    .mmregs
;---------------------------------------------------
```

```
;-------------------------------------------------
;       functional descriptoin
;       filter order: 4 order(cascade 2nd order+ 2nd order)
;       cut freq. of pass :1000hz
;       cut freq. of stop ;2400h
;
;-------------------------------------------------

;-------------------------------------------------
;       activation
;       activation      example:
;       call    iir_ini
;       call    iir_lowpass
;-------------------------------------------------

;-------------------------------------------------
;       External Data
        .ref    IN_BUF
;-------------------------------------------------

;-------------------------------------------------
;       temporary       data:
;       dp=4
;       cram0
;-------------------------------------------------

;-------------------------------------------------
;       data    structure
;       IN_BUF          .set    400h        ;buf_l=128
;       OUT_BUF         .set    500h        ;buf_l=128
;       IIR_BUF         .set    600h        ;buf_l=6
;       IIR_COFF_TAB    .set    610h        ;buf_l=10
;-------------------------------------------------

;-------------------------------------------------
;       input data:     IN_BUF
;       data format:    16-bit data buffer
;-------------------------------------------------
```

```
;-------------------------------------------------
;           input data:         OUT_BUF
;           data format:        16-bit data buffer
;-------------------------------------------------


iir_lowpass

          STM       #IN_BUF,AR6
          STM       #OUT_BUF,AR7

          STM       #128-1,BRC                  ;filter for 128 point
          RPTB      iir_filter_loop-1
          LD        *AR6+,8,A                   ;load input data

iir_filter
          STM       #IIR_BUF+5,AR5              ;AR5:d(n),d(n-1),d(n-2)
          STM       #IIR_COFF_TAB,AR4           ;AR4:A2,A1,B2,B1,B0
          STM       #2-1,AR1                    ;cascade=2

feedback_path
          MAC       *AR4+,*AR5-,A               ;input+d(n-2)*A2
          MAC       *AR4,*AR5,A                 ;input+d(n-2)*A2+d(n-1)*A1/2
          MAC       *AR4+,*AR5-,A               ;A=A+d(n-1)*A1/2
          STH       A,*AR5+                     ;d(n)=A
          MAR       *AR5+

forward_path
          MPY       *AR4+,*AR5-,A               ;d(n-2)*B2
          MAC       *AR4+,*AR5,A                ;d(n-2)*B2+d(n-1)*B1
          DELAY     *AR5-                       ;d(n-2)=d(n-1)
          BANZ      feedback_path,*AR1-
          MAC       *AR4+,AR5,A
          DELAY     *AR5-                       ;d(n-1)=d(n)
          STH       A,CRAM0
          LD        CRAM0,2,A                   ;scale the output
          STL       A,*AR7+                     ;save data
iir_filter_loop
          RET
```

```
ini_iir
        STM     #IIR_COFF_TAB,AR4       ;move from prog to data buffer
        RPT     #10-1
        MVPD    #iir_table_start,*AR4+
        STM     #IIR_BUF,AR5
        RPTZ    A,#6-1
        STL     A,*AR5+                 ;ini d(n),d(n-1),d(n-2)=0
        RET

iir_table_start
        .word   7fffh                   ;1:A2
        .word   4000h                   ;1:A1
        .word   4000h                   ;1:B2
        .word   4000h                   ;1:B1
        .word   7fffh                   ;1:B0

        .word   7fffh                   ;2:A2
        .word   4000h                   ;2:A1
        .word   4000h                   ;2:B2
        .word   4000h                   ;2:B1
        .word   7fffh                   ;2:B0

      .end
```

第四章 数字调制的 DSP 实现

调制是为了信号特性与信道特性相匹配，不同类型的信道特性将相应地存在不同类型的调制方式。调制的最终目的就是尽可能地减少占用带宽，尽可能地提高信号传输速率和质量。由于无线信道是时变色散信道，存在严重多径和衰落等不利于数据传输的因素，因此，选择适合于无线信道传输的数字调制方式是非常重要的。本章将扼要地介绍数字调制的基本原理，讨论 QPSK 的 DSP 实现。

4.1 无线通信中的数字调制

数字调制与模拟调制相比，具有更好的抗噪声性能和更强的抗信道衰落能力等。数字信号处理器使人们可以完全用软件实现数字调制器和解调器。

4.1.1 影响选择数字调制方式的因素

有几个因素会影响数字调制方案的选择。一个好的调制方案应能在低接收信噪比的条件下提供小的误比特率，对抗多径和衰落性能良好，占用最小的带宽，并且容易实现，价格低廉。现有的调制方案不能同时满足以上所有的要求，有的误比特率性能好，有的带宽利用率高。对于不同应用的要求，需要在选择数字调制方案时进行折衷。

调制方案的性能常用它的功率效率和带宽效率来衡量。功率效率描述了在低功率情况下，一种调制技术保持数字信息信号正确传输的能力。在数字通信系统中，为提供抗噪声性能，有必要提高信号的功率。然而，为得到一定的误码率，所需要提高信号功率的数值，取决于使用的调制方法。一种数字调制方案的功率效率 η_p（有时叫做能量效率）是由它怎样有利于信号误码率和功率之间的折衷来衡量的，通常表示为在接收机输入端特定的误码率下（如 10^{-5}），每比特信号能量和噪声功率谱密度的比值（E_b/N_0）。

带宽效率描述了调制方案在有限的带宽内传输数据的能力。一般说来，提高数据率意味着减少每个数字符号的脉冲宽度。这样，数据率和占用带宽之间就有不可避免的矛盾。有些调制方案在这两者之间的折衷上，性能优于其他的方案。带宽效率反映了对分配的带宽是如何有效利用的，它定义为在给定带宽内每赫兹带宽数据率吞吐量的值。如果 R 是每秒数据率，单位是比特，B 是已调 RF 信号占用的带宽，则带宽效率 η_B 表示为：

$$\eta_B = \frac{R}{B} \qquad \text{bps/Hz} \tag{4-1}$$

如果一种调制制度的 η_B 值大，则在分配的带宽内传输的数据更多，所以数字移动系统的系统容量与调制方案的带宽效率有直接的联系。

带宽效率有一个基本的上限。香农（Shannon）的信道编码理论指出，在加性高斯白

噪声信道中，在一个任意小的错误概率下，最大的带宽效率受限于信道内的噪声，其信道容量如式（4-2）所示：

$$\eta_{B\max} = \frac{C}{B} = \log_2(1+\frac{S}{N}) \tag{4-2}$$

其中，C 是信道容量（单位是 bps），B 是 RF 带宽，S/N 是信噪比。

在数字通信系统的设计中，经常需要在带宽效率和功率效率之间折衷。例如，对信息信号增加差错控制编码，一种方案是提高占用带宽（这样就降低了带宽效率），但同时对于给定的误比特率所必需的接收功率降低了，于是以带宽效率换取了功率效率。另一种方案，采用多进制的调制方案（多进制键控）降低了占用带宽，但是增加了所必需的接收功率，于是以功率效率换取了带宽效率。

4.1.2 数字信号的带宽和功率谱密度

对于信号带宽，实际上并没有一个适用于所有情况的定义。然而，所有的定义都是基于信号功率谱密度（PSD）的某种度量。随机信号 $x(t)$的功率谱密度的定义如下：

$$P_x(f) = \lim_{T\to\infty}\left(\frac{\overline{|X_T(f)|^2}}{T}\right) \tag{4-3}$$

其中，上划线表示总体平均，$X_T(f)$是 $x_T(t)$的傅立叶变换，$x_T(t)$是信号 $x(t)$的截短式，定义为：

$$x_T(t) = \begin{cases} x(t) & -T/2 < t < T/2 \\ 0 & other \quad t \end{cases} \tag{4-4}$$

已调（带通）信号的功率谱密度与基带复包络信号的功率谱密度有关。如果一个基带信号 $s(t)$如下所示：

$$s(t) = \mathrm{Re}\{g(t)\exp(j2\pi f_c t)\} \tag{4-5}$$

其中 $g(t)$是基带信号的复包络，则带通信号的 PSD 如下：

$$P_s(f) = \frac{1}{4}[P_g(f-f_c) + P_g(-f-f_c)] \tag{4-6}$$

其中 $P_g(f)$是 $g(t)$的 PSD。

信号的绝对带宽定义为信号的非零值功率谱在频域上占的范围。像基带矩形脉冲信号，其 PSD 外形为$(sinf)^2/f^2$，在频域上无限延伸，它的绝对带宽就为无限大。更为简单和广泛适用的带宽度量是零点-零点带宽。零点-零点带宽等于频谱主瓣宽度。

常用的带宽度量方法是衡量频谱的分散程度，叫做半功率带宽。半功率带宽定义为 PSD 下降到一半时，或者比峰值低 3dB 时的频率范围。半功率带宽也叫做 3dB 带宽。

4.2 脉冲成形器的设计方法

当矩形脉冲通过带限信道时，脉冲会在时间上扩展，每个符号的脉冲将扩展到相邻符号的码元内。这会造成码间串扰（ISI），并导致接收机在检测一个码元时发生错误的概率增大。一种常用的减少码间串扰的方法是增加信道带宽。然而，无线通信系统要求在减少码间串扰条件下，占用带宽小，并尽可能地减少调制带外辐射。无线系统中在相邻信道内的带外辐射，一般应比带内的辐射低 40dB 到 80dB。因为很难在发射机射频上抑制带外辐射，脉冲成形只能在基带或中频上进行。现有的许多脉冲成形技术都可用来在减少码间串扰的同时，减少已调数字信号占用的带宽。

4.2.1 脉冲成形器的基本原理和设计方法

奈奎斯特指出，只要把通信系统（包括发射机、信道和接收机）的整个响应设计成在接收机端每个抽样时刻只对当前的符号有响应，而对其他符号的响应全等于零，那么码间串扰 ISI 的影响就能完全被抵消。设 $h_{eff}(t)$为整个通信系统的冲激响应，则无码间串扰准则可表示为：

$$h_{eff}(nT_s)=\begin{cases}K & n=0\\ 0 & n\neq 0\end{cases} \tag{4-7}$$

其中，T_s 是符号周期，n 是整数，K 是非零常数。系统传递函数为：

$$h_{eff}(t)=\delta(t)*\rho(t)*h_c(t)*h_r(t) \tag{4-8}$$

其中，$\rho(t)$ 是符号的脉冲波形，$h_c(t)$是信道的冲激响应，$h_r(t)$是接收机冲激响应。

在选取满足式（4-7）的传递函数时有两个重要的地方需要考虑。第一，$h_{eff}(t)$在接近 n $\neq 0$ 的取样点的地方要迅速衰减。第二，如果信道是理想的（$h_c(t)=\delta(t)$），发射端和接收端的成形滤波器必须逼近 $H_{eff}(f)$函数。

显然，式（4-9）所示的冲激响应能消除 ISI 的奈奎斯特条件。

$$h_{eff}(t)=\frac{\sin(\pi t/T_s)}{\pi t/T_s} \tag{4-9}$$

如果整个通信系统的冲激响应如方程（4-9）所示，那么就有可能完全消除 ISI。系统的传递函数可以通过对冲激响应做傅立叶变换得到，如下式所示：

$$H_{eff}(f)=\frac{1}{f_s}\Pi\left(\frac{f}{f_s}\right) \tag{4-10}$$

这个传递函数对应于绝对带宽为 $f_s/2$ 的矩形滤波器，其中 f_s 为符号速率。

在无线通信中最常用的脉冲成形滤波器是升余弦滚降滤波器。这种滤波器能满足奈奎

斯特准则。升余弦滤波器的传递函数为：

$$H_{RC}(f)=\begin{cases}1 & 0\le|f|\le(1-\alpha)/2T_s\\ \dfrac{1}{2}\left[1+\cos\left(\dfrac{\pi(2T_s|f|)-1+\alpha}{2\alpha}\right)\right] & (1-\alpha)/2T_s<|f|\le(1+\alpha)/2T_s\\ 0 & |f|>(1+\alpha)/2T_s\end{cases} \tag{4-11}$$

其中，α 是滚降因子，取值范围为 0 到 1。当 α =0 时，升余弦滚降滤波器对应于具有最小带宽的矩形滤波器。这种滤波器的冲激响应可由对其传递函数做傅立叶变换得到：

$$h_{RC}(t)=\left(\frac{\sin(\pi t/T_s)}{\pi t}\right)\left(\frac{\cos(\pi\alpha t/T_s)}{1-(4\alpha t/(2T_s))^2}\right) \tag{4-12}$$

升余弦滚降滤波器在基带的冲激响应，当 α 为不同值时如图 4-1 所示。与门函数滤波器(α=0)相比，升余弦滚降滤波器在过零点（当 $t>>T_s$ 时，约为 $1/t^3$）衰减得快得多。由图 4-2 可见，随着滚降因子 α 的增加，滤波器带宽也增加，相邻符号间隔内时间旁瓣减小。这意味着增加 α 可以减小对定时抖动的敏感度，但增加了占用的带宽。

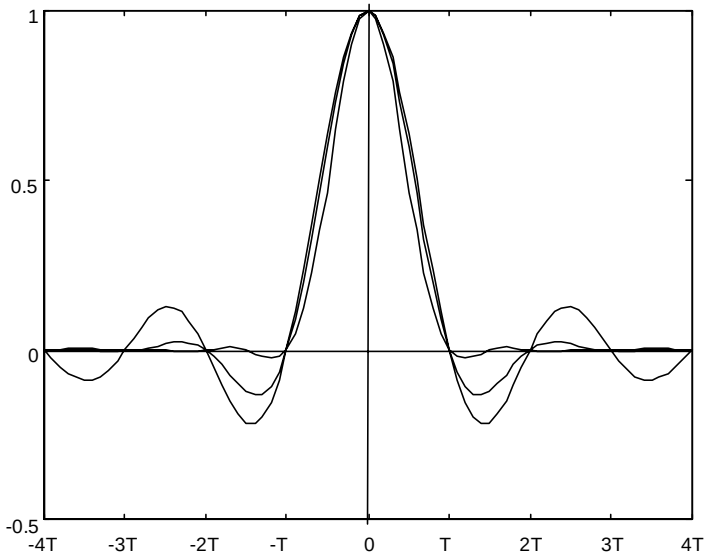


图 4-1 升余弦滚降滤波器的冲激响应

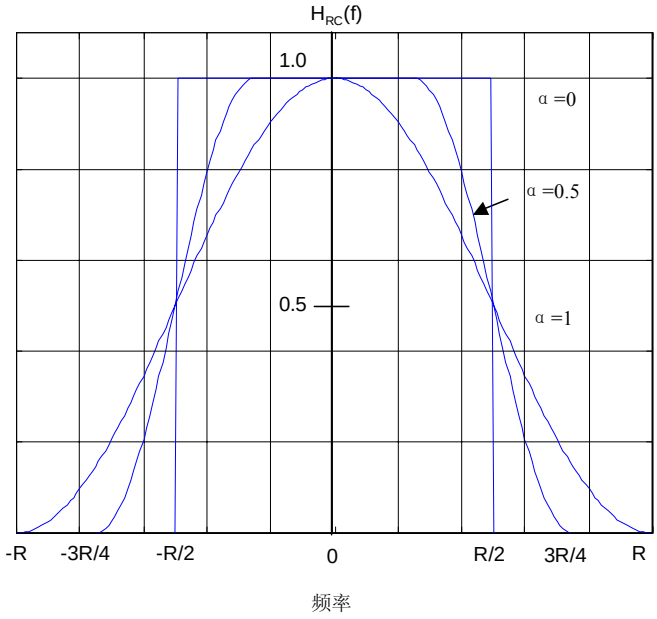


图 4-2 升余弦滤波器的转移函数

4.2.2 脉冲成形器的 MATLAB 设计

实际中常用的脉冲成形器是具有升余弦特性的滤波器，其定义为：

$$X_{rc}(f)=\begin{cases}T & 0\le|f|\le(1-\alpha)/2T\\ \dfrac{T}{2}\left[1+\cos\dfrac{\pi T}{\alpha}\left(|f|-\dfrac{1-\alpha}{2T}\right)\right] & (1-\alpha)/2T<|f|\le(1+\alpha)/2T\\ 0 & |f|>(1+\alpha)/2T\end{cases} \tag{4-13}$$

在理想信道中，在抽样间隔 $t=nT$ 上按无码间串扰的要求，设计发送和接收滤波器。如果发送滤波器的频率响应为 $G_T(f)$，接收滤波器的频率响应为 $G_R(f)$，则 $G_T(f)$ $G_R(f)$设计应满足无码间串扰的条件。即：

$$G_T(f)G_R(f)=X_{rc}(f) \tag{4-14}$$

则在抽样时刻 $t=nT$ 上的码间串扰为 0。

例 4-1：按照式（4-14）设计频率响应分别为 $G_T(f)$，$G_R(f)$的发收滤波器，$G_R(f)$ 是 $G_T(f)$的匹配滤波器。

解：设计数字发收滤波器的最简捷方法是采用具有线性相位（平衡的脉冲响应）的 FIR 滤波器，所求的幅度响应为

$$|G_T(f)|=|G_R(f)|=\sqrt{X_{rc}(f)} \tag{4-15}$$

式中 $X_{rc}(f)$由式(4-14)给出。频率响应与数字滤波器的脉冲响应关系为：

$$G_T(f)=\sum_{n=-(N-1)/2}^{(N-1)/2} g_T(n)e^{-j2\pi fnT_s} \tag{4-16}$$

式中 T_s 为采样间隔，N 为滤波器的阶数（奇数），因为 $G_T(f)$是带限的，抽样频率 F_s 至少为 $2/T$，我们选择为

$$F_s=\frac{1}{T_s}=\frac{4}{T}$$

因此重叠频率为 $F_s/2=2/T$。又因为 $G_T(f)=\sqrt{X_{rc}(f)}$，所以可以在以 $\Delta f=F_s/N$ 为间隔的等间距频率点上对 $X_{rc}(f)$进行抽样，因此可得：

$$\sqrt{X_{rc}(m\Delta f)}=\sqrt{X_{rc}(mF_s/N)}=\sum_{n=-(N-1)/2}^{(N-1)/2} g_T(n)e^{-j2\pi mn/N} \tag{4-17}$$

亦可表达为

$$g_T(n)=\sum_{m=-(N-1)/2}^{(N-1)/2}\sqrt{X_{rc}\left(\frac{4m}{NT}\right)}e^{j2\pi mn/N},\qquad n=0,\pm1,\ldots,\pm\frac{N-1}{2} \tag{4-18}$$

因为 $g_T(n)$必须对称，所求线性相位发送端滤波器的脉冲响应采用延迟 $g_T(n)(N-1)/2$ 个抽样以保证 $g_T(n)$的对称性。

［程序 4-1］脉冲成形器的 MATLAB 实现程序

Main 程序：

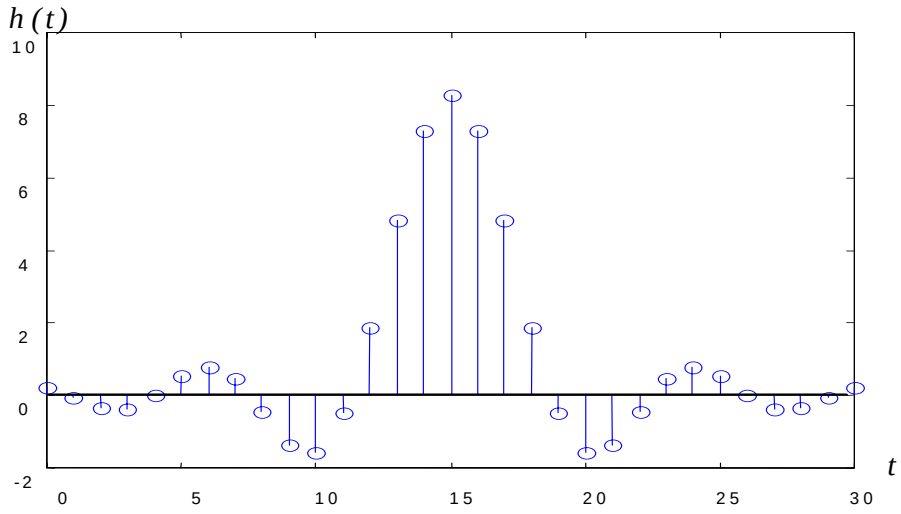
```
N=31;
T=1;
alpha=1/4;
n=-(N-1)/2:1:(N-1)/2;
%g_T 的表达式通过如下方式获得
for i=1:length(n);
    g_T(i)=0;
    for m=-(N-1)/2:1:(N-1)/2;
        g_T(i)=g_T(i)+sqrt(xrc(4*m/(N*T),alpha,T))*exp(j*2*pi*m*(n(i))/N);
    end;
end;
n2=0:N-1;
%得到频率响应特性
[G_T,W]=freqz(g_T,1);
%归一化幅度响应
magG_T_in_dB=20*log10(abs(G_T)/max(abs(G_T)));
```

```
%发射机和接收机滤波器叠加所得的冲激响应
g_R=g_T;
imp_resp_of_cascade=conv(g_R,g_T);
n3=0:length(imp_resp_of_cascade)-1;
%随后为绘图命令
figure(1)
stem(n2,g_T);
figure(2)
plot(W/max(W)/2,magG_T_in_dB);
figure(3)
stem(n3,imp_resp_of_cascade);
```

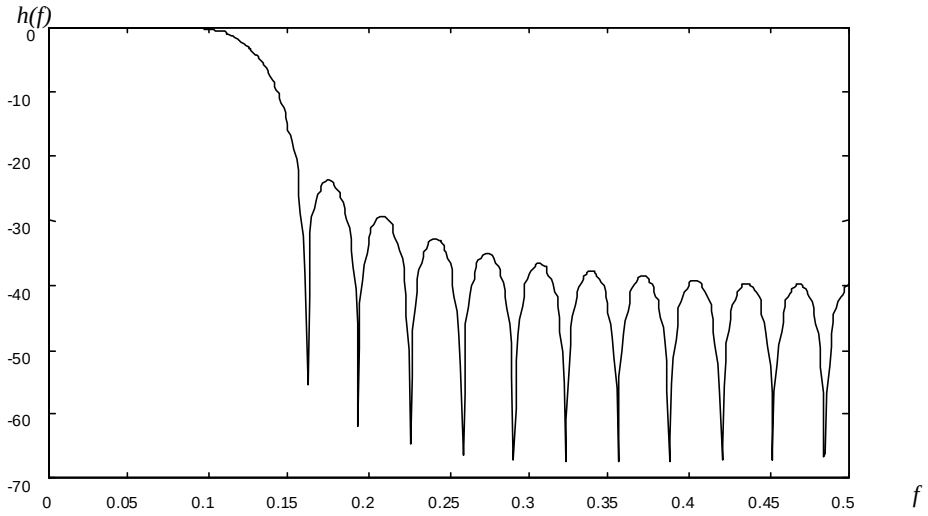
子程序：xrc.m

```
function [y]=xrc(f,alpha,T);
if (abs(f)>((1+alpha)/(2*T)))
    y=0;
elseif (abs(f)>((1-alpha)/(2*T)))
    y=(T/2)*(1+cos((pi*T/alpha)*(abs(f)-(1-alpha)/(2*T))));
else
    y=T;
end;
```

图 4-3(a)给出了 α=1/4，N=31 时的脉冲响应图形。相应的频率响应如图 4-3(b)所示。



(a)



(b)

图 4-3 发送端处截频 FIR 滤波器的脉冲响应和频率响应

在图 4–4 是例 4–1 所求的系统脉冲响应图。

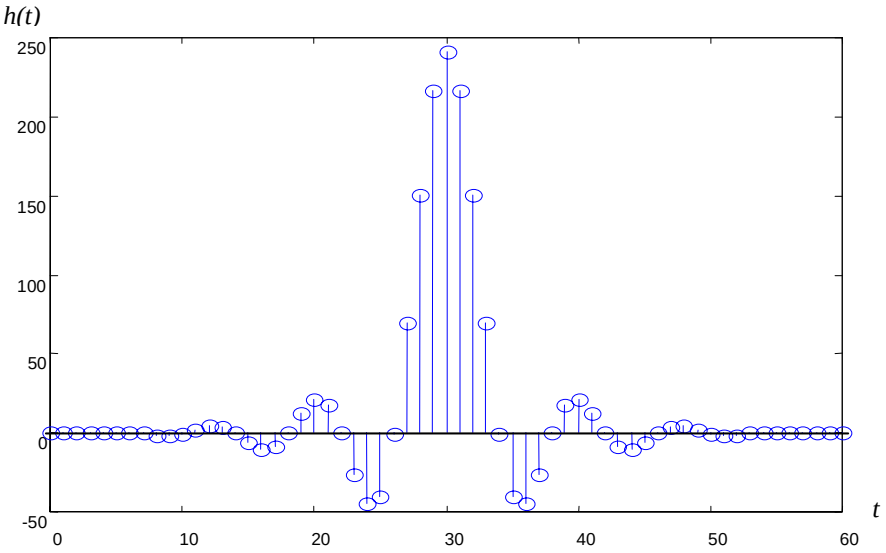


图 4-4 级联的发送滤波器和接收端滤波器的脉冲响应

4.3 QPSK 调制解调的 DSP 实现

四相相移键控（QPSK）是一种性能优良，应用十分广泛的数字调制方式，它的频率利用率高，是二相相移键控（BPSK）的两倍，采用相干检测时其误码性能也与 BPSK 相同。

4.3.1 QPSK 调制解调的基本原理和设计方法

由于在一个调制符号中传输两个比特，四相相移键控（QPSK）比 BPSK 的带宽效率高两倍。载波的相位为四个间隔相等的值，比如 0、π/2、π 和 3π/2，每一个相位值对应于唯一的一对消息比特。这个符号状态集的 QPSK 信号可定义为：

$$S_{QPSK}(t)=\sqrt{\frac{2E_s}{T_s}}\cos\left[2\pi f_c t+(i-1)\frac{\pi}{2}\right]\qquad 0\le t\le T_s\qquad i=1,2,3,4 \tag{4-19}$$

其中 E_s 为每个符号的能量，T_s 为符号持续时间，等于两个比特周期。

使用三角恒等变换，上式在 $0\leqslant t\leqslant T_s$ 可重写为：

$$S_{QPSK}(t)=\sqrt{\frac{2E_s}{T_s}}\cos\left[(i-1)\frac{\pi}{2}\right]\cos(2\pi f_c t)-\sqrt{\frac{2E_s}{T_s}}\sin\left[(i-1)\frac{\pi}{2}\right]\sin(2\pi f_c t) \tag{4-20}$$

如果 QPSK 信号集的基元函数 $\phi_1(t)=\sqrt{2/T_s}\cos(2\pi f_c t)$，$\phi_2(t)=\sqrt{2/T_s}\sin(2\pi f_c t)$ 定义在 $0\leqslant t\leqslant T_s$ 的间隔内，信号集内的四个信号可由基元信号表示出：

$$S_{QPSK}(t)=\left\{\sqrt{E_s}\cos\left[(i-1)\frac{\pi}{2}\right]\phi_1(t)-\sqrt{E_s}\sin\left[(i-1)\frac{\pi}{2}\right]\phi_2(t)\right\}\qquad i=1,2,3,4 \tag{4-21}$$

基于这种表示，QPSK 信号可以用有四个点的二维星座图表示，如图 4-5 所示。需要注意的是，差分 QPSK 信号集可以简单地通过旋转星座得到。例如，图 4-6 给出了另一个 QPSK 信号集，其相位的值为 π/4、3π/4、5π/4 和 7π/4。

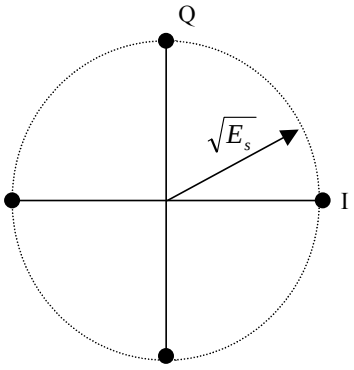


图 4-5 载波相位为 0，π/2，π，3π/2 的 QPSK 星座

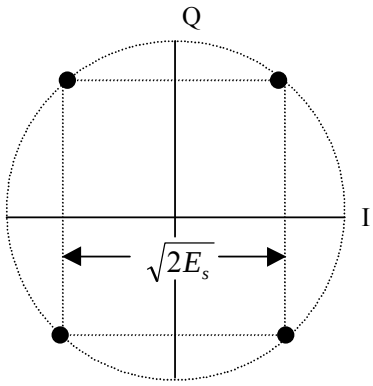


图 4-6 载波相位为 π/4，3π/4，5π/4，7π/4 的 QPSK 星座

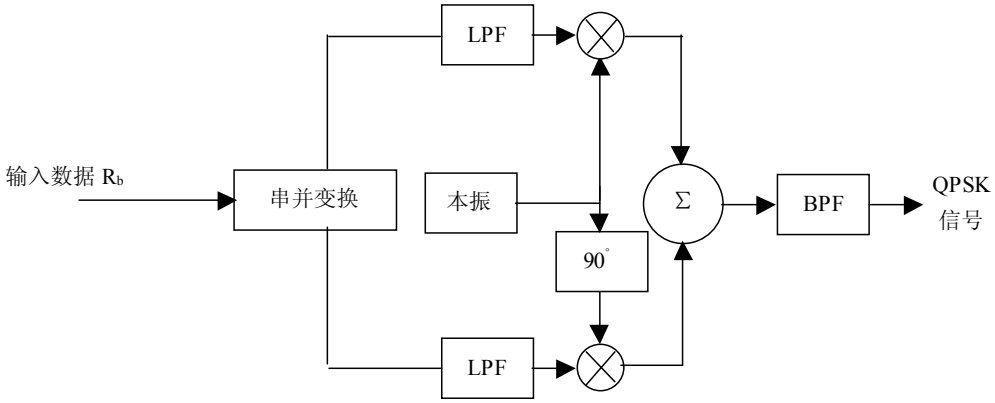


图 4-7 QPSK 发射机框图

图 4-7 给出了典型 QPSK 发射机框图。单极性二进制消息流比特率为 R_b，首先用一个单极性-双极性转换器将它转换为双极性非归零（NRZ）序列。然后比特流 $m(t)$分为两个比特流 $m_I(t)$和 $m_Q(t)$（同相和正交流），每个的比特率为 $R_s=R_b/2$。比特流 $m_I(t)$叫做“偶”流，$m_Q(t)$叫做“奇”流。两个二进制序列分别用两个正交的载波 $\phi_1(t)$和 $\phi_2(t)$调制。两个已调信号每一个都可以看作是一个 BPSK 信号，对它们求和产生一个 QPSK 信号。解调器输出端的滤波器将 QPSK 信号的功率谱限制在分配的带宽内。这样可以防止信号能量泄漏到相邻的信道，还能去除在调制过程中产生的带外杂散信号。在绝大多数实现方式中，脉冲成形在基带进行，从而在发射机的输出端提供适当的 RF 滤波。

QPSK 信号的产生比较容易，下面我们把重点放在 QPSK 信号的解调上面。

4.3.2 QPSK 调制解调的 MATLAB 设计

不妨将一组 M 个载波调相信号波形表达式写为：

$$u_m(t) = Ag_T(t)\cos\left(2\pi f_c t + \frac{2\pi m}{M}\right) \qquad m = 0,1,\ldots,M-1 \tag{4-22}$$

这里 $g_T(t)$为发射端的成形脉冲，决定发送信号的频谱特征，A 是信号振幅。
当 $g_T(t)$为矩形脉冲时，它的定义为：

$$g_T(t) = \sqrt{\frac{2}{T}}, \qquad 0 \le t \le T \tag{4-23}$$

在这种情况下，发送信号波形在符号间隔 $0 \leqslant t \leqslant T$ 内可以表示为（其中 $A = \sqrt{E_s}$ ）

$$u_m(t) = \sqrt{\frac{2E_s}{T}}\cos\left(2\pi f_c t + \frac{2\pi m}{M}\right), \qquad m = 0,1,\ldots,M-1 \tag{4-24}$$

通过将式(4-24)中余弦函数的角度看作为两个角度的和，可将式(4-24)变形为：

$$\begin{aligned} u_m(t) &= \sqrt{E_s}\,g_T(t)\cos\left(\frac{2\pi m}{M}\right)\cos 2\pi f_c t - \sqrt{E_s}\,g_T(t)\sin\left(\frac{2\pi m}{M}\right)\sin 2\pi f_c t \\ &= s_{mc}\psi_1(t) + s_{ms}\psi_2(t) \end{aligned} \tag{4-25}$$

其中

$$\begin{aligned} s_{mc} &= \sqrt{E_s}\cos\left(\frac{2\pi m}{M}\right) \\ s_{ms} &= \sqrt{E_s}\sin\left(\frac{2\pi m}{M}\right) \end{aligned} \tag{4-26}$$

且 $\psi_1(t)$ 和 $\psi_2(t)$ 是正交基函数，定义为：

$$\begin{aligned} \psi_1(t) &= g_T(t)\cos 2\pi f_c t \\ \psi_2(t) &= -g_T(t)\sin 2\pi f_c t \end{aligned} \tag{4-27}$$

通过适当地归一化 $g_T(t)$可将这两个基函数的能量归一化为 1。于是，一个调相信号可被看作为两个正交载波，该载波幅度为每一个信号间隔的发送相位决定。因此，数字调相信号在几何学上表示为具有分量 S_{mc} 和 S_{ms} 的二维矢量。

$$s_m = \left(\sqrt{E_s}\cos\frac{2\pi m}{M} \qquad \sqrt{E_s}\sin\frac{2\pi m}{M}\right) \tag{4-28}$$

将 k 比特信息映射到 $M=2^k$ 个可能相位的方法有多种。最常用的方法是使用格雷编码，这种编码的相邻相位相差一个二进制比特。所以在传送时当由于噪声引起的相邻相位的选择有误时，用格雷编码得到的 k 比特序列中仅有一个比特发生错误。

通过加性高斯白噪声信道传输的一个信号间隔内接收到的带通信号可以表示为：

$$\begin{aligned} r(t) &= u_m(t) + n(t) \\ &= u_m(t) + n_c(t)\cos 2\pi f_c t - n_s(t)\sin 2\pi f_c t \end{aligned} \tag{4-29}$$

其中 $n_c(t)$和 $n_s(t)$为加性噪声的两个正交分量。

接收信号可以与式(4-27)给出的 $\psi_1(t)$和 $\psi_2(t)$进行相关。两个相关器的输出产生含有噪声信号分量，它们可以表示为：

$$\begin{aligned} r &= s_m + n \\ &= \left(\sqrt{E_s}\cos\frac{2\pi m}{M} + n_c \qquad \sqrt{E_s}\sin\frac{2\pi m}{M} + n_s \right) \end{aligned} \tag{4-30}$$

其中 n_c 和 n_s 定义为：

$$\begin{aligned} n_c &= \frac{1}{2}\int_{-\infty}^{\infty} g_T(t) n_c(t) dt \\ n_s &= \frac{1}{2}\int_{-\infty}^{\infty} g_T(t) n_s(t) dt \end{aligned} \tag{4-31}$$

正交噪声分量 $n_c(t)$和 $n_s(t)$是互不相关的零均值高斯随机过程，所以 $E(n_c)=E(n_s)=0$ 且 $E(n_c n_s)=0$。n_c 和 n_s 的方差为：

$$E\left(n_c^2\right) = E\left(n_s^2\right) = \frac{N_0}{2} \tag{4-32}$$

最佳判决器的作用就是将接收到的信号矢量 r 映射到 M 个可能发送的信号矢量$\{s_m\}$上去，并且选出对应于最大映射的矢量。因此，可以获得下述相关：

$$C(r, s_m) = r \cdot s_m, \qquad m = 0,1,\ldots,M-1 \tag{4-33}$$

因为所有的信号均有相等的能量，所以数字相位调制的判决器可以等价为计算接收信号矢量 $r=(r_1，r_2)$的相位，即：

$$\theta_r = \tan^{-1}\frac{r_2}{r_1} \tag{4-34}$$

并且从$\{s_m\}$中选出相位最接近于 θ_r 的信号。

因为二进制相位调制和二进制 PAM 相同，所以在加性高斯白噪声信道下，相位调制判决器的误码率为：

$$P_2 = Q\left(\sqrt{\frac{2E_b}{N_0}}\right) \tag{4-35}$$

其中 E_b 表示每比特的能量。可以将四相调制看作正交载波上的两个二进制相位调制系统，所以四相调制误码率与二进制相位调制的误码率相同。对于 $M>4$ 的情况， P_M 的一个较好近似式为：

$$P_M \approx 2Q\left(\sqrt{\frac{2E_s}{N_0}}\sin\frac{\pi}{M}\right)$$
$$= 2Q\left(\sqrt{\frac{2kE_b}{N_0}}\sin\frac{\pi}{M}\right) \tag{4-36}$$

其中 $K=\log_2 M$ 比特/符号。由于误码率取决于 k 比特符号与相应信号相位间的映射，M 进制相位调制的等价误码率公式很难推导。当采用格雷码时，对应于相邻信号相位的两个 k 比特符号之间仅相差一个比特。由于噪声引起的误差最有可能导致单比特错误，因此 M 进制相位调制的等价误码率约为：

$$P_b = \frac{1}{k}P_M \tag{4-37}$$

例 4-2： 对 QPSK 通信系统进行仿真，该系统的模型如图 4-8 所示。

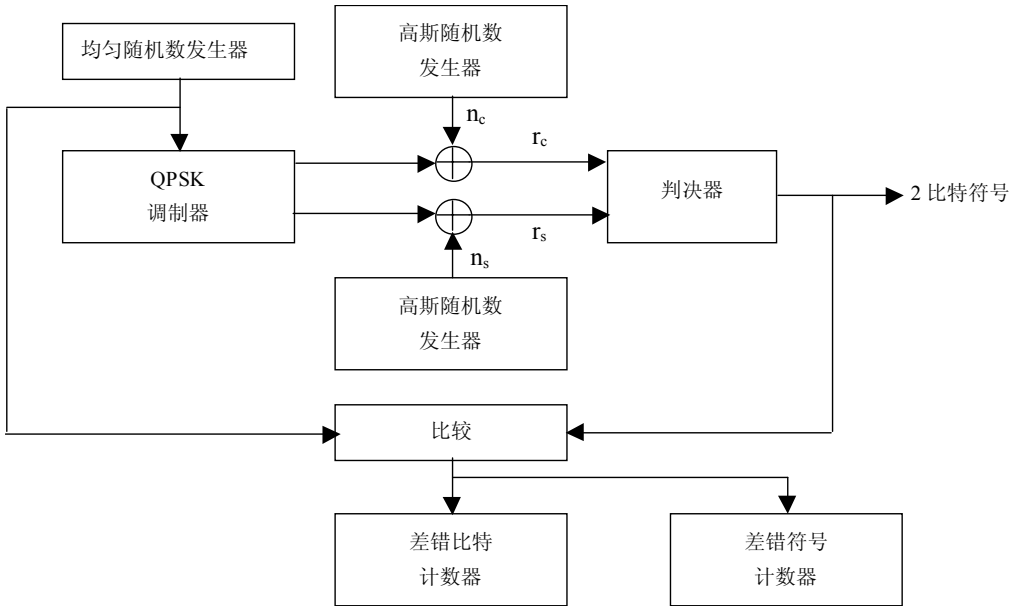


图 4-8 QPSK 系统仿真框图

解：如图所示，仿真式(4-30)给出的随机变量 r，该变量是信号相关器的输出和判决器的输入。首先产生一个正交（2 比特）符号序列，将它映射为相应的四相信号点。为了完成这个任务，使用一个在（0,1）区间内产生均匀分布随机数的随机数发生器。（0,1）区间被分成四个等间隔的区间（0,0.25），（0.25,0.5），（0.5,0.75）和（0.75,1.0），这四个间隔分别对应于信息比特对 00，01，11，10，这些信息比特对被用来选择信号相位矢量 s_m。

加性噪声分量 n_c 和 n_s 是均值为 0、方差为 σ^2、相互统计独立的高斯随机变量。为了方便，可以将方差归一化为 1，即 $\sigma^2=1$，且通过改变信号能量参数来控制接收信号的信噪比，反之亦然。

判决器判决由式(4-30)给出的接收信号矢量 $r=s_m+n$，并计算 r 在四种可能的信号向量 s_m 上的投影（点积）。然后选择对应于最大投影的信号点，将判决器的输出与发送符号相比较。计算误符号数和误比特数。

在不同的信噪比 E_b/N_0 下发送 10000 个符号的仿真结果如图 4-9 所示。其中，$E_b=E_s/2$ 是比特能量。该图也给出了误码率（定义为 $P_s=P_M/2$）和相应的理论误码率（由式(4-36)给出）。

［程序 4-2］QPSK 系统仿真程序。

Main 程序：

```
SNRindB1=0:2:10;
SNRindB2=0:0.1:10;
for i=1:length(SNRindB1)
    [pb,ps]=cm_sm32(SNRindB1(i));                %仿真比特和符号误码率
    smld_bit_err_prb(i)=pb;
    smld_symbol_err_prb(i)=ps;
end;
for i=1:length(SNRindB2)
    SNR=exp(SNRindB2(i)*log(10)/10);             %信噪比
    theo_err_prb(i)=Qfunct(sqrt(2*SNR));         %理论比特误码率
end;
%随后为绘图曲线
semilogy(SNRindB1,smld_bit_err_prb,'*');
hold on;
semilogy(SNRindB1,smld_symbol_err_prb,'o');
semilogy(SNRindB2,theo_err_prb);
hold off;
```

子程序：cm_sm32.m

```
function [pb,ps]=cm_sm32(snr_in_dB)
N=10000;
E=1;                                 %每符号能量
snr=10^(snr_in_dB/10);       %信噪比
sgma=sqrt(E/snr)/2;                  %噪声方差
%信号映射
s00=[1 0];
s01=[0 1];
s11=[-1 0];
s10=[0 -1];
%数据源的产生
for i=1:N
    temp=rand;
```

```
    if (temp<0.25)                  %在区间(0,1)间的一个均匀随机变量
        dsource1(i)=0;              %信源输出为“00”的概率为 1/4
        dsource2(i)=0;
    elseif (temp<0.5)               %信源输出为“01”的概率为 1/4
        dsource1(i)=0;
        dsource2(i)=1;
    elseif (temp<0.75)          %信源输出为“10”的概率为 1/4
        dsource1(i)=1;
        dsource2(i)=0;
    else                            %信源输出为“11”的概率为 1/4
        dsource1(i)=1;
        dsource2(i)=1;
    end;
end;
%判决、误码率的计算
numofsymbolerror=0;
numofbiterror=0;
for i=1:N
    %在判决器处的接收信号，对于第 i 个符号为：
    n(1)=gngauss(sgma);
    n(2)=gngauss(sgma);
    if ((dsource1(i)==0)&(dsource2(i)==0))
        r=s00+n;
    elseif ((dsource1(i)==0)&(dsource2(i)==1))
        r=s01+n;
    elseif ((dsource1(i)==1)&(dsource2(i)==0))
        r=s10+n;
    else
        r=s11+n;
    end;
    %以下为计算互相关量度
    c00=dot(r,s00);
    c01=dot(r,s01);
    c10=dot(r,s10);
    c11=dot(r,s11);
    %第 i 个符号的判决按如下方式进行
    c_max=max([c00 c01 c10 c11]);
    if (c00==c_max)
        decis1=0;decis2=0;
```

```
    elseif (c01==c_max)
        decis1=0;decis2=1;
    elseif (c10==c_max)
        decis1=1;decis2=0;
    else
        decis1=1;decis2=1;
    end;
    %如果判决不对，计错器加 1
    symbolerror=0;
    if (decis1~=dsource1(i))
        numofbiterror=numofbiterror+1;
        symbolerror=1;
    end;
    if (decis2~=dsource2(i))
        numofbiterror=numofbiterror+1;
        symbolerror=1;
    end;
    if (symbolerror==1)
        numofsymbolerror=numofsymbolerror+1;
    end;
end;
ps=numofsymbolerror/N;                    %因为总共有 N 个符号
pb=numofbiterror/(2*N);                   %因为发送了 2N 个比特
```

子程序：Qfunct.m

```
function [y]=Qfunct(x)
y=(1/2)*erfc(x/sqrt(2));
```

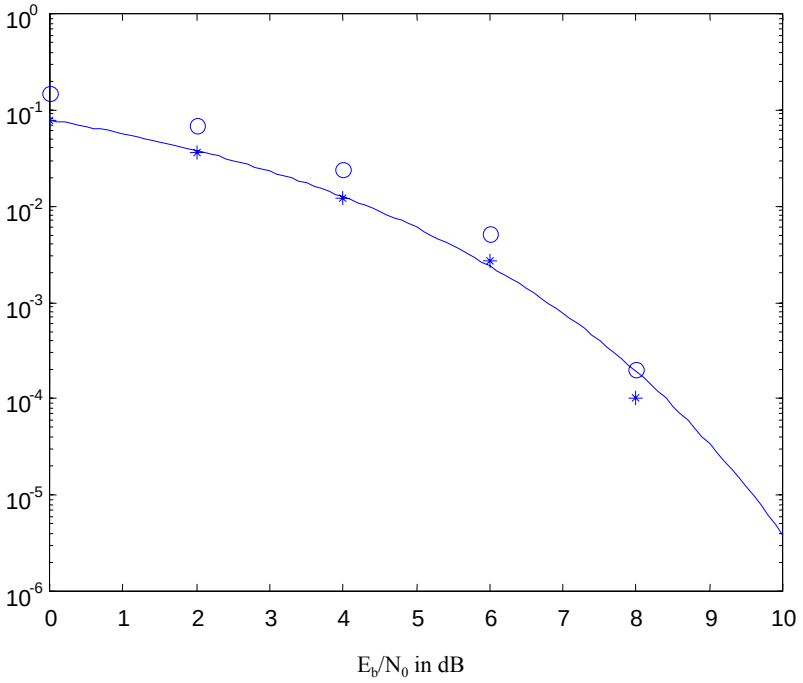


图 4-9　4PSK 系统仿真的误码率

（“—”理论误码率；“*”误符号率；“o”误比特率）

4.3.3 QPSK 调制解调的 DSP 实现

一、QPSK 调制程序

[程序 4-3]　该程序每次读入 2bit 数据进行 QPSK 调制，实现无码间串扰的信号传输。

设计参数：

- 采样速率为 14400 样点/秒
- 载波频率为 1800Hz
- 带宽为 300Hz~3300Hz
- 符号速率为 2400 符号/秒
- 汉明窗平方根升余弦滚降
- 滚降系数为 0.35

编程说明：ini_modem 子程序产生由 2 bit 数据所对应的实部、虚部、无码间串扰脉冲成形波形

wave_filter 子程序完成连续数据流的基带波形成形

fc_modem 子程序完成基带信号对 1800Hz 载波的调制

get_data 子程序对输入的 2 bit 调制数据读表，读出相对应的脉冲响应波形

```
;-------------------------------------------------------
;       version:            1.0
;       data:               july-11-1999
;       authors:            ren guo chun
;       comment:              original created
;       cpu type:           ti_tms32054xx
;-------------------------------------------------------

;-------------------------------------------------------
;       compiler:           tms320c54x assembler
;       version:            1.02 (pc)
;       include files:    .mmregs
;-------------------------------------------------------

;-------------------------------------------------------
;       functional description:
;       QPSK modem
;       2400 sym/s
;       fc=1800Hz
;       fs=14400 sample/s
;       51 order fir filter
;-------------------------------------------------------

;-------------------------------------------------------
;       activation
;       activation example:
;       call      modem_psk
;       call      ini_modem
;-------------------------------------------------------

;-------------------------------------------------------
;       External Data
        .ref      CRAM2
;-------------------------------------------------------

;-------------------------------------------------------
;       temporary data:
;       dp=4
;       cram0
```

```
;           cram1
;           cram2
;           cram3
;           cram4
;           cram5
;           cram6
;           cram7
;-------------------------------------------------------

;-------------------------------------------------------
;           data structure
;           SD_FILA      .set     400H                  ;buf_l=80h
;           SD_FILB      .set     500h                  ;buf_l=80h
;           SD_1800      .set     600h                  ;buf_l=10h
;           SD_NUL       .set     700H                      ;buf_l=51
;           SD_2PP       .set     SD_NUL+1*51               ;buf_l=51
;           SD_2PN       .set     SD_NUL+2*51               ;buf_l=51
;           SD_4PP       .set     SD_NUL+3*51               ;buf_l=51
;           SD_4PN       .set     SD_NUL+4*51               ;buf_l=51
;           SD_DAB0      .set     7000h                 ;sd_dabl=200h
;-------------------------------------------------------

;-------------------------------------------------------
;           ram define:
;           SD_FBIP      as SD_FILA and SD_FILB    input point
;           SD_FBOP      as SD_FILA and SD_FILB    output point
;           SD_DAIP      as SD_DAB0                 input point
;           SD_DAOP      as SD_DAB0                  output point
;-------------------------------------------------------

;-------------------------------------------------------
;           inut data:          cram2
;           data format:        2-bit data
;-------------------------------------------------------

;-------------------------------------------------------
;           output data:        SD_DAB0
;           data format:        16-bit data buffer
;-------------------------------------------------------
```

```
modem_psk:
        CALL    get_data                ;get modem data
        CALL    wave_filter             ;for filter
        CALL    fc_modem                    ;for 1800hz modem
        RET
ini_modem

get_data

        LD      CRAM2,A
        AND     #11,A                       ;read 2-bit data
        SFTL    A,2
        ADD     #wave_rd_tab,A
        READA   CRAM0                       ;for re wave read pointer
        ADD     #1,A
        READA   CRAM1                       ;for im wave read pointer
        RET

wave_filter
        LD      SD_FBIP,A
        STLM    A,AR1                       ;ready re filter buffer in_pointer
        ADD     #100H,A
        STLM    A,AR2                       ;ready im filter buffer in_pointer

        LD      CRAM0,A
        STLM    A,AR3                       ;re wave read tab pointer
        LD      CRAM1,A
        STLM    A,AR4                       ;im wave read tab pointer
        STM     #51-1,BRC               ;repeat 51 times
        RPTBD   wave_add_m00-1
        STM     #80H,BK                     ;circular buffer size

        LD      *AR3+,A
        ADD     *AR1,A
        STL     A,*AR1+%                    ;re wave add
        LD      *AR4+,A
        ADD     *AR2,A
        STL     A,*AR2+%                    ;im wave add
wave_add_m00
```

```
        LD      SD_FBIP,A               ;change filter buffer in_pointer
        SUB     #SD_FILA,A
        ADD     #6,A                    ;inc 6
        AND     #80H-1,A                ;circular buffer size
        ADD     #SD_FILA,A
        STL     A,SD_FBIP
        RET

fc_modem
        LD      #DODP,DP
        LD      SD_DAOP,A               ;D/A buffer out_pointer
        SUB     SD_DAIP,A               ;D/A buffer in_pointer
        NOP
        NOP
        XC      2,ALEQ
        ADD     #SD_DABL,A              ;circular buffer size
        NOP
        SUB     #090H,A
        BC      sd_mod_m00,ALT          ;check if d/a buffer full
        RSBX    SXM
        LD      SD_FBIP,A               ;filter buffer in_pointer
        SUB     SD_FBOP,A               ;filter buffer out_pointer
        NOP
        NOP
        XC      2,ALT
        ADD     #80H,A                  ;circular buffer size
        NOP
        SUB     #10H,A
        SSBX    SXM
        BC      sd_mod_m02,AGEQ         ;check if can do filter

        LD      BIT_FLG,A
        AND     #11B,A
        BC      sd_mod_m00,ANEQ         ;check if D/A buffer full

sd_mod_m02
        LD      SD_FBOP,A
        STLM    A,AR2                   ;re filter buffer out_pointer
```

```
        ADD     #100H,A
        STLM    A,AR3                   ;im filer buffer out_pointer
        LD      SD_DAIP,A
        ADD     #SD_DAB0,A
        STLM    A,AR1                   ;D/A buffer in_pointer

        ST      #3000H,CRAM1

        STM     #SD_1800,AR4            ;cos table pointer
        STM     #SD_1800+2,AR5          ;sin table pointer
        STM     #8-1,BRC                ;repeat 8 times
        RPTBD   sd_mod_m01-1
        STM     #80H,BK                 ;circular buffer size

        MPY     *AR2,*AR4+,A            ;A=cos*re_d
        MACR    *AR3,*AR5+,A            ;A=cos*re_d+sin*im_d
        STH     A,CRAM0
        LD      CRAM1,T                 ;adjust amp
        MPY     CRAM0,A
        STH     A,*AR1+                 ;save D/A data

        LD      #0,A                    ;clear data
        STL     A,*AR2+%
        STL     A,*AR3+%
sd_mod_m01

        MVKD    AR2,SD_FBOP             ;save filter out_pointer
        LD      SD_DAIP,A               ;change D/A buffer in_pointer
        ADD     #8,A                    ;inc 8
        AND     #SD_DABL-1,A
        STL     A,SD_DAIP
sd_mod_m00
        RET
ini_modem:
        SSBX    FRCT                    ;frct=1
        STM     #SD_NUL,AR1
        RPTZ    A,#51-1
        STL     A,*AR1+                 ;clear sd_null buffer
        STM     #SD_2PP,AR1
```

```
        RPT     #51-1
        MVPD    #ep_wavn,*AR1+          ;mov data from program to data ram
        NOP
        STM     #51-1,BRC               ;gen 2pn wave
        STM     #SD_2PP,AR1
        STM     #SD_2PN,AR2
        RPTB    ep_movd_m00-1
        LD      *AR1+,A
        NEG     A                       ;A=-A
        STL     A,*AR2+
ep_movd_m00
        ST      #5A82H,CRAM0            ;as 0.707
        LD      CRAM0,T
        STM     #50,BRC
        STM     #SD_2PP,AR1
        STM     #SD_4PP,AR2             ;gen 4pp wave for 8-psk
        STM     #SD_4PN,AR3             ;gen 4pn wave for 8-psk
        RPTB    ep_movd_m01-1
        MPY     *AR1+,A                 ;mpy as 0.707
        STH     A,*AR2+
        NEG     A                        ;A=-A
        STH     A,*AR3+
ep_movd_m01

        STM     #SD_1800,AR1            ;move f1800 table
        RPT     #10h-1
        MVKD    #ef1800,*AR1+           ;move data from program to data ram
        RET

wave_rd_tab
        .word   SD_2PP,SD_NUL           ;00b
        .word   SD_4PP,SD_4PP
        .word   SD_NUL,SD_2PP           ;for 2-bit 01b
        .word   SD_4PN,SD_4PP
        .word   SD_2PN,SD_NUL           ;for 2-bit 10b
        .word   SD_4PN,SD_4PN
        .word   SD_NUL,SD_2PN           ;for 2-bit 11b
        .word   SD_4PP,SD_4PN
;sqar cos k=0.35
```

```
ep_wave .word     0FFF6H,00003H,00013H,0001EH,00019H,0FFFAH,0FFC1H,0FF89H
        .word     0FF85H,0FFE8H,000C2H,001DCH,002A9H,0026AH,00087H,0FCF9H
        .word     0F89CH,0F53CH,0F52AH,0FA7EH,00633H,01781H,02BB9H,03ED2H
        .word     04C7DH,05174H,04C7DH,03ED2H,02BB9H,01781H,00633H,0FA7EH
        .word     0F52AH,0F53CH,0F89CH,0FCF9H,00087H,0026AH,002A9H,001DCH
        .word     000C2H,0FFE8H,0FF85H,0FF89H,0FFC1H,0FFFAH,00019H,0001EH
        .word     00013H,00003H,0FFF6H

;sin table 360/8 dobble
ef1800
        .word     00000H,05A82H,07FFFH,05A82H,00000H,0A57FH,08002H,0A57FH
        .word     00000H,05A82H,07FFFH,05A82H,00000H,0A57FH,08002H,0A57FH
        .end
```

二、QPSK 解调程序

[程序 4-4] 该程序与程序 4-3 结合完成数据的调制解调。

设计参数：

- 采样速率为 14400 样点/秒
- 载波频率为 1800Hz
- 带宽为 300Hz~3300Hz
- 符号速率为 2400 符号/秒
- 汉明窗平方根升余弦滚降
- 滚降系数为 0.35

编程说明：cal_sin 子程序用线性内插的方法，提高正弦、余弦值的精度
demodem_psk 子程序完成接收信号的波形成形、低通滤波功能
sin_512_tab 的制表精度为 0.703125 度

```
;-------------------------------------------------------
;         version:          1.0
;         data:             july-11-1999
;         authors:          ren guo chun
;         comment:            original created
;         cpu type:         ti_tms32054xx
;-------------------------------------------------------

;-------------------------------------------------------
;         compiler:         tms320c54x assembler
;         version:          1.02 (pc)
;         include files:    .mmregs
;-------------------------------------------------------
```

```
;-------------------------------------------------
;       functional description:
;       QPSK demod
;       2400 sym/s
;       fc=1800Hz
;       fs=14400 sample/s
;       51 order fir filter
;-------------------------------------------------

;-------------------------------------------------
;       activation
;       activation example:
;       call    demod_psk
;-------------------------------------------------

;-------------------------------------------------
;       External Data
        .ref    RV_ADB0
;-------------------------------------------------

;-------------------------------------------------
;       temporary data:
;       dp=4
;       cram0
;       cram1
;       cram2
;       cram3
;       cram4
;       cram5
;       cram6
;       cram7
;-------------------------------------------------

;-------------------------------------------------
;       data structure
;       RV_FILA     .set    1000h           ;buf_l=51h
;       RV_FILB     .set    1100h           ;buf_l=51h
;       RV_BUFA     .set    1200h           ;buf_l=51h
;       RV_BUFB     .set    1400h           ;buf_l=51h
```

```
;           RV_ADB0       .set    6000h                   ;rv_adbl=800h
;-------------------------------------------------------

;-------------------------------------------------------
;           ram define:
;           RV_FBIP       as RV_BUFA and RV_BUFB    input point
;           RV_FBOP       as RV_BUFA and RV_BUFB    output point
;           RV_ADIP       as RV_DAB0                input point
;           RV_ADOP       as RV_DAB0                outpit point
;           DOP_PS1       as SIN_512_TAB            read    point
;-------------------------------------------------------

;-------------------------------------------------------
;           in      data:
;           RV_ADB0       buffer for A/D
;           data format:      16-bit data buffer
;-------------------------------------------------------

;-------------------------------------------------------
;           output data:
;           RV_BUFA     as real data
;           RV_BUFB     as im data
;           data format:      16-bit data buffer
;-------------------------------------------------------

demod_psk
            LD          RV_ADIP,A                   ;A/D buffer in_pointer
            SUB         RV_ADOP,A                       ;A/D buffer out_pointer
            NOP
            NOP
            XC          2,ALT
            ADD         #RV_ADBL,A                  ;circular buffer size
            SUB         #8,A
            BC          rv_dem_m00,ALET         ;check if have sample demod

            LD          RV_FBOP,A                   ;rv filter buffer out_pointer
            SUB         RV_FBIP,A                   ;rv filter buffer in_pointer
            NOP
            NOP
```

```
XC      2,ALEQ
ADD     #200H,A                 ;circular buffer size
SUB     #24,A
BC      rv_dem_m00,ALT          ;check if can do demod

LD      #RV_BUFA,A
ADD     RV_FBIP,A
STLM    A,AR1                   ;re rv filter in_pointer

LD      #RV_BUFB,A
ADD     RV_FBIP,A
STLM    A,AR2                   ;im rv filter in_pointer

LD      RV_ADOP,A
ADD     #RV_ADB0,A
STLM    A,AR3                   ;A/D buffer out_pointer

STM     #50,AR0
STM     #8-1,BRC                ;repeat 8 times
RPTBD   rv_dem_m01-1
STM     #RV_ADBL,BK             ;circular buffer size

SSBX    FRCT                    ;frct=1
STM     #RV_FILA,AR4            ;re fir filter pointer
STM     #RV_FILB,AR5            ;im fir filter pointer

CALL    cal_sin                 ;compute sin and cos

LD      *AR3+%,T                ;demod
MPY     CRAM5,A                 ;sin data
STH     A,*AR4+0                ;save to re fir filter
MPY     CRAM6,A                 ;cos data
STH     A,*AR5+0                ;save to im fir filter
RSBX    FRCT                    ;frct=0

RPTZ    A,#51-1
MACD    *AR4-,ep_wavn,A         ;re fir filtering
STH     A,*AR1+
RPTZ    A,#51-1
```

```
        MACD    *AR5-,ep_wavn,A         ;im fir filtering
        STH     A,*AR2+
rv_dem_m01
        SSBX    FRCT                    ;frct=1
        LD      RV_FBIP,A               ;change rv filter in_pointer
        ADD     #08H,A                  ;inc 8
        AND     #1FFH,A
        STL     A,RV_FBIP

        LD      RV_ADOP,A               ;change A/D out_pointer
        ADD     #08H,A                  ;inc 8
        AND     #RV_ADBL-1,A
        STL     A,RV_ADOP
rv_dem_m00
        RET

cal_sin
        LD      DOP_PS1,A               ;calculate sin and cos in cram5 cram6
        AND     #3FH,A
        STL     A,CRAM0
        LD      CRAM0,9,A
        STL     A,CRAM0                 ;load low 6 bit
        LD      DOP_PS1,10,A
        STH     A,CRAM1                 ;load high 9 bit

        LD      CRAM1,A
        ADD     #sin_512_tab,A          ;read sin_table
        READA   CRAM2                   ;first data d1
        ADD     #1,A
        READA   CRAM3                   ;second data d2
        LD      CRAM3,A
        SUB     CRAM2,A
        STL     A,CRAM4                 ;d1-d2
        LD      CRAM0,T
        MPY     CRAM4,A
        ADD     CRAM2,16,A
        STH     A,CRAM5                 ;sin=(d1-d2)*xxxxxxb
        LD      CRAM1,A
        ADD     #80,A
```

```
        AND     #1FFH,A
        ADD     #sin_512_tab,A
        READA   CRAM2                   ;first data d3
        ADD     #1,A
        READA   CRAM3                   ;second data d4
        LD      CRAM3,A
        SUB     CRAM2,A
        STL     A,CRAM4                 ;d3=d4
        MPY     CRAM4,A
        ADD     CRAM2,16,A
        STH     A,CRAM6                 ;cos=(d3-d4)*xxxxxxb
        LD      DOP_DD1,A               ;for sd and rv dopple data
        ADD     DOP_PS1,A               ;change sin_table read pointer
        ADD     #DEG45K,A
        NOP
        NOP
        XC      2,ALT
        ADD     #8000H,A

        AND     #7FFFH,A
        STL     A,DOP_PS1               ;save sin_table read pointer
        RET

;sqar cos k=0.35
ep_wave .word   0FFF6H,00003H,00013H,0001EH,00019H,0FFFAH,0FFC1H,0FF89H
        .word   0FF85H,0FFE8H,000C2H,001DCH,002A9H,0026AH,00087H,0FCF9H
        .word   0F89CH,0F53CH,0F52AH,0FA7EH,00633H,01781H,02BB9H,03ED2H
        .word   04C7DH,05174H,04C7DH,03ED2H,02BB9H,01781H,00633H,0FA7EH
        .word   0F52AH,0F53CH,0F89CH,0FCF9H,00087H,0026AH,002A9H,001DCH
        .word   000C2H,0FFE8H,0FF85H,0FF89H,0FFC1H,0FFFAH,00019H,0001EH
        .word   00013H,00003H,0FFF6H

;sin table 360/ 512
sin_512_tab
        .word   00000H,00192H,00324H,004B6H,00648H,007D9H,0096AH,00AFBH
        .word   00C8CH,00E1CH,00FABH,0113AH,012C8H,01455H,015E2H,0176EH
        .word   018F9H,01A82H,01C0BH,01D93H,01F1AH,0209FH,02223H,023A6H
        .word   02528H,026A8H,02826H,029A3H,02B1FH,02C99H,02E11H,02F87H
        .word   030FBH,0326EH,033DFH,0354DH,036BAH,03824H,0398CH,03AF2H
```

```
        .word   03C56H,03DB8H,03F17H,04073H,041CEH,04325H,0447AH,045CDH
        .word   0471CH,04869H,049B4H,04AFBH,04C3FH,04D81H,04EBFH,04FFBH
        .word   05133H,05268H,0539BH,054C9H,055F5H,0571DH,05842H,05964H
        .word   05A82H,05B9CH,05CB3H,05DC7H,05ED7H,05FE3H,060EBH,061F0H
        .word   062F1H,063EEH,064E8H,065DDH,066CFH,067BCH,068A6H,0698BH
        .word   06A6DH,06B4AH,06C23H,06CF8H,06DC9H,06E96H,06F5EH,07022H
        .word   070E2H,0719DH,07254H,07307H,073B5H,0745FH,07504H,075A5H
        .word   07641H,076D8H,0776BH,077FAH,07884H,07909H,07989H,07A05H
        .word   07A7CH,07AEEH,07B5CH,07BC5H,07C29H,07C88H,07CE3H,07D39H
        .word   07D89H,07DD5H,07E1DH,07E5FH,07E9CH,07ED5H,07F09H,07F37H
        .word   07F61H,07F86H,07FA6H,07FC1H,07FD8H,07FE9H,07FF5H,07FFDH
        .word   07FFFH,07FFDH,07FF5H,07FE9H,07FD8H,07FC1H,07FA6H,07F86H
        .word   07F61H,07F37H,07F09H,07ED5H,07E9CH,07E5FH,07E1DH,07DD5H
        .word   07D89H,07D39H,07CE3H,07C88H,07C29H,07BC5H,07B5CH,07AEEH
        .word   07A7CH,07A05H,07989H,07909H,07884H,077FAH,0776BH,076D8H
        .word   07641H,075A5H,07504H,0745FH,073B5H,07307H,07254H,0719DH
        .word   070E2H,07022H,06F5EH,06E96H,06DC9H,06CF8H,06C23H,06B4AH
        .word   06A6DH,0698BH,068A6H,067BCH,066CFH,065DDH,064E8H,063EEH
        .word   062F1H,061F0H,060EBH,05FE3H,05ED7H,05DC7H,05CB3H,05B9CH
        .word   05A82H,05964H,05842H,0571DH,055F5H,054C9H,0539BH,05268H
        .word   05133H,04FFBH,04EBFH,04D81H,04C3FH,04AFBH,049B4H,04869H
        .word   0471CH,045CDH,0447AH,04325H,041CEH,04073H,03F17H,03DB8H
        .word   03C56H,03AF2H,0398CH,03824H,036BAH,0354DH,033DFH,0326EH
        .word   030FBH,02F87H,02E11H,02C99H,02B1FH,029A3H,02826H,026A8H
        .word   02528H,023A6H,02223H,0209FH,01F1AH,01D93H,01C0BH,01A82H
        .word   018F9H,0176EH,015E2H,01455H,012C8H,0113AH,00FABH,00E1CH
        .word   00C8CH,00AFBH,0096AH,007D9H,00648H,004B6H,00324H,00192H
        .word   00000H,0FE6FH,0FCDDH,0FB4BH,0F9B9H,0F828H,0F697H,0F506H
        .word   0F375H,0F1E5H,0F056H,0EEC7H,0ED39H,0EBACH,0EA1FH,0E893H
        .word   0E708H,0E57FH,0E3F6H,0E26EH,0E0E7H,0DF62H,0DDDEH,0DC5BH
        .word   0DAD9H,0D959H,0D7DBH,0D65EH,0D4E2H,0D368H,0D1F0H,0D07AH
        .word   0CF06H,0CD93H,0CC22H,0CAB4H,0C947H,0C7DDH,0C675H,0C50FH
        .word   0C3ABH,0C249H,0C0EAH,0BF8EH,0BE33H,0BCDCH,0BB87H,0BA34H
        .word   0B8E5H,0B798H,0B64DH,0B506H,0B3C2H,0B280H,0B142H,0B006H
        .word
0AECEH,0AD99H,0AC66H,0AB38H,0AA0CH,0A8E4H,0A7BFH,0A69DH
        .word   0A57FH,0A465H,0A34EH,0A23AH,0A12AH,0A01EH,09F16H,09E11H
        .word   09D10H,09C13H,09B19H,09A24H,09932H,09845H,0975BH,09676H
        .word   09594H,094B7H,093DEH,09309H,09238H,0916BH,090A3H,08FDFH
```

```
.word   08F1FH,08E64H,08DADH,08CFAH,08C4CH,08BA2H,08AFDH,08A5CH
.word   089C0H,08929H,08896H,08807H,0877DH,086F8H,08678H,085FCH
.word   08585H,08513H,084A5H,0843CH,083D8H,08379H,0831EH,082C8H
.word   08278H,0822CH,081E4H,081A2H,08165H,0812CH,080F8H,080CAH
.word   080A0H,0807BH,0805BH,08040H,08029H,08018H,0800CH,08004H
.word   08002H,08004H,0800CH,08018H,08029H,08040H,0805BH,0807BH
.word   080A0H,080CAH,080F8H,0812CH,08165H,081A2H,081E4H,0822CH
.word   08278H,082C8H,0831EH,08379H,083D8H,0843CH,084A5H,08513H
.word   08585H,085FCH,08678H,086F8H,0877DH,08807H,08896H,08929H
.word   089C0H,08A5CH,08AFDH,08BA2H,08C4CH,08CFAH,08DADH,08E64H
.word   08F1FH,08FDFH,090A3H,0916BH,09238H,09309H,093DEH,094B7H
.word   09594H,09676H,0975BH,09845H,09932H,09A24H,09B19H,09C13H
.word   09D10H,09E11H,09F16H,0A01EH,0A12AH,0A23AH,0A34EH,0A465H
.word   0A57FH,0A69DH,0A7BFH,0A8E4H,0AA0CH,0AB38H,0AC66H,0AD99H
.word   0AECEH,0B006H,0B142H,0B280H,0B3C2H,0B506H,0B64DH,0B798H
.word   0B8E5H,0BA34H,0BB87H,0BCDCH,0BE33H,0BF8EH,0C0EAH,0C249H
.word   0C3ABH,0C50FH,0C675H,0C7DDH,0C947H,0CAB4H,0CC22H,0CD93H
.word   0CF06H,0D07AH,0D1F0H,0D368H,0D4E2H,0D65EH,0D7DBH,0D959H
.word   0DAD9H,0DC5BH,0DDDEH,0DF62H,0E0E7H,0E26EH,0E3F6H,0E57FH
.word   0E708H,0E893H,0EA1FH,0EBACH,0ED39H,0EEC7H,0F056H,0F1E5H
.word   0F375H,0F506H,0F697H,0F828H,0F9B9H,0FB4BH,0FCDDH,0FE6FH
.end
```

第五章 同步功能模块的 DSP 实现

在数字通信系统中，以及在某些采用同步解调的模拟通信系统中，同步是一个非常重要的问题。通信系统能否有效可靠地工作，很大程度上依赖于同步技术的优劣。也就是说，同步技术的好坏将直接影响通信质量的好坏，它是通信系统中的一项关键技术。因此，本章概述同步的基本原理，讨论无线通信系统中同步功能模块的 DSP 实现。

5.1 同步分类

同步的种类很多，按照不同的分类准则，可以将同步分为多种类型。常用的分类方法有两种：第一种是按同步的功用，第二种是按照传输同步信息方式的不同。下面分别予以介绍。

5.1.1 按同步功用分类

按照同步的功用可分为载波同步、位同步、群同步和网同步四种，下面主要对前三种同步方式加以说明。

一、载波同步

在利用同步解调（即相干解调）的传输系统中，接收端需要有一个与发送端同频同相的载波。以 DSB（双边带）信号为例，设发端产生 DSB 信号时用到的载波为 $s_T(t)=\cos(\omega_1 t+\theta_1)$，接收端解调的本地载波为 $s_R(t)=\cos(\omega_2 t+\theta_2)$。

基带信号 $x(t)$ 经过调制后，得双边带信号为 $x(t)\cos(\omega_1 t+\theta_1)$。解调器是一个相乘器，经过解调得到：

$$x(t)\cos(\omega_1 t+\theta_1)\cos(\omega_2 t+\theta_2)$$

$$=\frac{1}{2}x(t)\{\cos[(\omega_1-\omega_2)t+(\theta_1-\theta_2)]+\cos[(\omega_1+\omega_2)t+(\theta_1+\theta_2)]\}$$

经过低通滤波后，可得

$$x'(t)=\frac{1}{2}x(t)\cos[(\omega_1-\omega_2)t+(\theta_1-\theta_2)]$$

(5-1)

当 $\omega_1 = \omega_2, \theta_1 = \theta_2$ 时，$x'(t) = x(t)/2$，这样可以不失真地恢复 $x(t)$。如果 $\omega_1 \neq \omega_2$，$\theta_1 \neq \theta_2$，就会引起波形的失真或者输出幅度的降低。因此接收端同步解调用的载波 $s_R(t)$ 与发送端调制用的载波 $s_T(t)$ 要同频同相。$s_R(t)$ 与 $s_T(t)$ 同频同相的过程称为载波同步；从接收信号中产生 $s_R(t)$ 的过程称为载波提取。

二、位同步（码元同步）

数字通信系统中，消息是由一串连续的码元传递的。这些码元通常都有相同的持续时间。接收端接收这个码元序列时，一般均需知道每个码元的起止时刻，从而对码元进行判别。例如：用取样判决器对信号进行取样判决时，一般均应对准每个码元最大值的位置。因此接收端要产生一个“码元定时脉冲序列”，并且定时脉冲的重复频率和相位（即位置）要与接收码元完全一致。即：

1. 接收端定时脉冲的重复频率和发送端码元速率相同；
2. 脉冲位置（即取样判决时刻）对准最佳取样判决位置。这个码元定时脉冲序列称为“码元同步脉冲”或“位同步脉冲”。

我们把位同步脉冲与接收码元的重复频率和相位一致的过程称为码元同步或位同步。而把位同步脉冲的产生称为位同步提取。

三、群同步（帧同步）

对数字信号传输来说，有了载波同步和位同步，发端发送的代码就可以在收端解调出来，但代码是一连串的“1”、“0”码，例如电传机发出的 5 单位码，英文字母 A、B、C 分别为 11000，10011，01110；解调以后一定要把每个字母区别开来，中间要加特殊的起止码，以便区分各个字母。有时要把若干字组成的句加以区分而插入起止码。在接收端产生与“字”或“句”的起止时刻相一致的定时脉冲序列，称为“字”同步或“句”同步，统称为群同步或帧同步。在传输数据时，为区别各组数据（由若干个码组成）也要插入称为群同步码的专门码组。

5.1.2 按同步方式分类

按传输同步信息方式的不同，可以分为外同步法和自同步法两种。

一、外同步法

由发送端发送专门的同步信息，接收端把这个专门的同步信息检测出来作为同步信号

的方法，称为外同步法。

二、自同步法

发送端不发送专门的同步信息，而是由接收端设法从接收信号中提取同步信息的方法，称为自同步法。

不论哪种同步方式，对正常的信息传输来说，都是非常必要的，只有收发两端之间建立了同步，信息才能可靠地传输，因此同步信息的可靠传输也是非常重要的。

下面主要针对根据同步功用分类情况，介绍载波同步、位同步和群同步的基本工作原理和性能。

5.2 载波同步

载波同步的方法有直接同步法（自同步法）和插入导频法（外同步法）两种。直接同步法不需要传输专门的导频（同步信号），而是从接收信号中设法提取同步载波。插入导频法是在发送有用信号的同时，在适当的频率位置上插入一个或多个称为导频的正弦波（同步载波），在接收端直接从信号中取出同步载波。下面首先介绍直接同步法中常用的平方变换法和平方环法。

5.2.1 平方变换法和平方环法

一、平方变换法

对于双边带（DSB）信号，可以采用图 5-1 所示的方框图，来提取同步载波。图中接收端输入信号包括有用信号 $x(t)\cos\omega_c t$ 和加性高斯白噪声 $n_i(t)$，经过带通滤波器后，滤除带外噪声。其中，$x(t)\cos\omega_c t$ 经过平方律部件后的输出 $e(t)$ 为

$$e(t)=[x(t)\cos\omega_c t]^2=\frac{x^2(t)}{2}+\frac{x^2(t)}{2}\cos 2\omega_c t \tag{5-2}$$

上式右边项 $\frac{1}{2}x^2(t)\cos 2\omega_c t$ 中，$x^2(t)/2$ 一定有直流成分，因此 $\frac{1}{2}x^2(t)\cos 2\omega_c t$ 中含有 $2\omega_c$ 的频率成分。

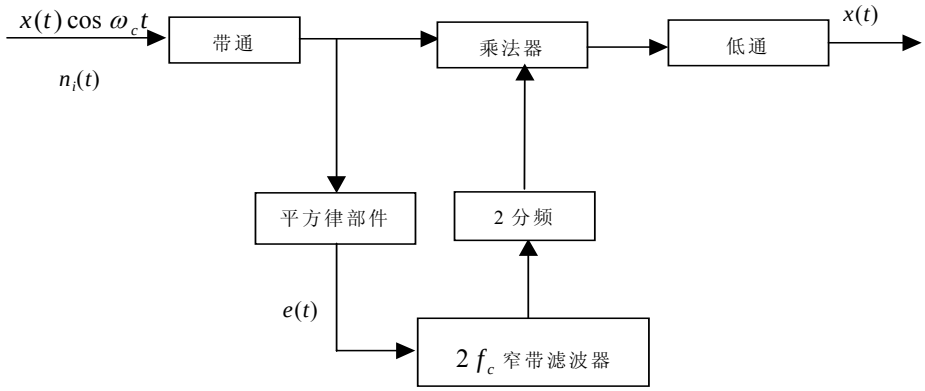


图 5-1 平方变换法提取同步载波

对于 2PSK 信号，$x(t)$ 为双极性矩形波，取值为 ± 1，$x^2(t)=1$，这样，$x(t)\cos\omega_c t$ 经过平方律部件后可得

$$e(t)=\frac{1}{2}+\frac{1}{2}\cos 2\omega_c t \tag{5-3}$$

由上式可见，$e(t)$ 通过带宽为 $2f_c$ 的窄带滤波器可以提取出 $2f_c$ 频率成分，经过二分频器就可以得到 f_c 的频率成分，这就是需要的同步载波。

二、平方环法

为了改善平方变换法的性能，在平方变换法方框图的基础上，把窄带滤波器改为锁相环，这样就变成了平方环法。由于锁相环具有良好的跟踪、窄带滤波和记忆性能，因此平方环法比平方变换法具有更好的性能，因而得到了更为广泛的应用。

下面，我们讨论一下相位模糊的问题。从图 5-1 可以看出，由 $2f_c$ 窄带滤波器可得到 $\cos 2\omega_c t$，经过二分频以后得到的可能是 $\cos\omega_c t$，也可能是 $\cos(\omega_c t+\pi)$。这种相位的不确定性称为相位含糊或相位模糊。相位含糊对模拟通信系统关系不大，因为耳朵听不出相位的变化。但对于数字通信系统来说情况就不同了，相位不同将使解调后码元反相。对于 2PSK 信号就可能出现“反向工作”的问题，为此，可以采用 2DPSK 调制来克服这一问题。

5.2.2 同相正交环法

在同步载波提取过程中，经常会用到同相正交环法。同相正交环法又称科斯塔斯环，其工作原理如下。

一、同相正交环法的工作原理

同相正交环法的方框图如 5-2 所示。

设 输入信号为 $x(t)\cos\omega_c t$，在振荡器锁定后输出为 $v_1=\cos(\omega_c t+\theta)$，$\theta$ 为锁相环的剩余相位误差，通常很小。v_1 经过 -90° 的相移电路后得

$$v_2=\cos(\omega_c t+\theta-90^\circ)=\sin(\omega_c t+\theta) \tag{5-4}$$

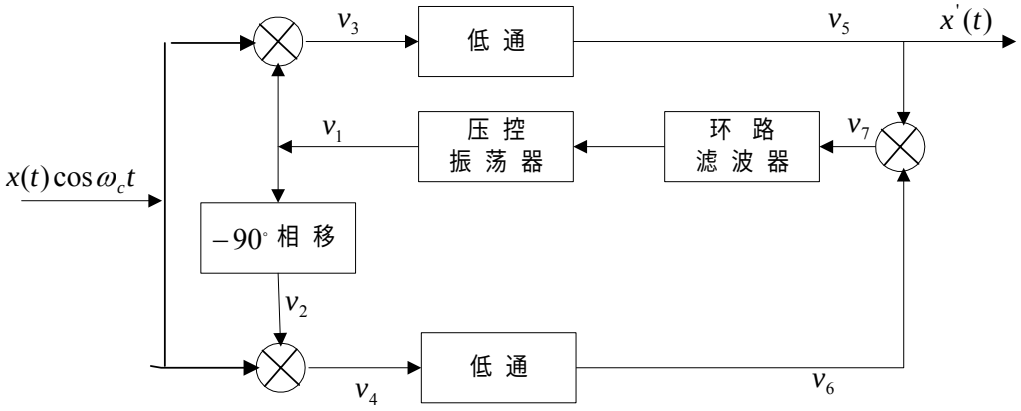


图 5-2 同相正交环法

信号 $x(t)\cos\omega_c t$ 分别与 v_1、v_2 相乘后，可得：

$$\begin{cases} v_3=x(t)\cos(\omega_c t+\theta)=\dfrac{1}{2}x(t)[\cos\theta+\cos(2\omega_c t+\theta)] \\ v_4=x(t)\cos(\omega_c t+\theta)=\dfrac{1}{2}x(t)[\sin\theta+\sin(2\omega_c t+\theta)] \end{cases} \tag{5-5}$$

分别经过低通滤波器，可得：

$$\begin{cases} v_5 = \dfrac{1}{2}x(t)\cos\theta \\ v_6 = \dfrac{1}{2}x(t)\sin\theta \end{cases} \tag{5-6}$$

v_5和v_6通过相乘器，可以得到：

$$v_7 = v_5 v_6 = \frac{1}{4}x^2(t)\sin\theta\cos\theta = \frac{1}{8}x^2(t)\sin 2\theta \approx \frac{1}{8}x^2(t)(2\theta) = \frac{1}{4}x^2(t)\theta \tag{5-7}$$

电压v_7经过环路滤波器后控制压控振荡器（VCO），使 VCO 输出信号频率与$\cos\omega_c t$同频，相位可能只差一个很小的θ，即$\theta \approx 0$。此时，$v_1 = \cos(\omega_c t + \theta)$就是需要提取的载波，而$v_5 = (x(t)/2)\cos\theta \approx x(t)/2$就是解调器的输出。

二、同相正交环的优缺点

同相正交环法的优点在于可以直接解调出$x(t)$。但其电路比较复杂，特别是其中有一个-90°的相移电路，当输入信号载波在某个波段范围内时，即ω_c变化时，相移电路实现起来比较复杂。

最后，讨论一下同相正交环法是否存在相位含糊的问题。初看起来这种方法中没有二分频器，似乎没有相位含糊的问题，但仔细分析起来同样存在相位含糊的问题。因为当$v_1 = \cos(\omega_c t + \theta + 180^\circ)$时，经过计算得到的$v_7$也是$(x^2(t)/8)\sin 2\theta$，因此$v_7$的相位也是不确定的。

5.2.3 同相正交环法的 MATLAB 设计

在介绍了同相正交环法的工作原理和优缺点后，下面给出了它的 MATLAB 实现框图，如图 5-3 所示。

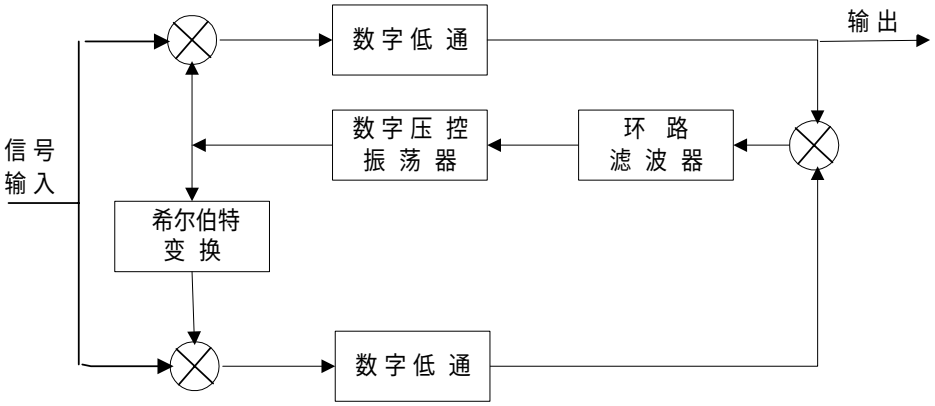


图 5-3 科斯塔斯环的 MATLAB 实现

5.3 位同步

在数字通信系统中，为了恢复发送信息，必须对解调输出的连续信号进行周期性抽样，以每个符号宽度为间隔抽样判决一次。因为通信系统中信道的传播延时一般是未知的，因此抽样判决的时刻就显得十分重要。位同步主要目的就是从接收信号中导出符号定时脉冲，从而取得抽样器判决的最佳时刻。上节我们学习了载波同步，为了说明载波同步和位同步的区别，本节首先介绍位同步和载波同步的区别，然后介绍两种常用的位同步方法：直接法（自同步法）和插入导频法（外同步法）。

5.3.1 位同步和载波同步的区别

为了简洁明了地介绍位同步与载波同步的区别，列举两个常用的例子，一个图 5-4 所示的 2ASK 包络解调的方框图，另一个是图 5-5 所示的 2PSK 相干解调的方框图。

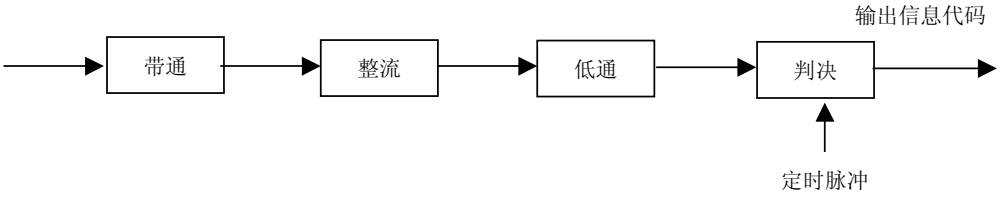


图 5-4 2ASK 非相干解调方框图

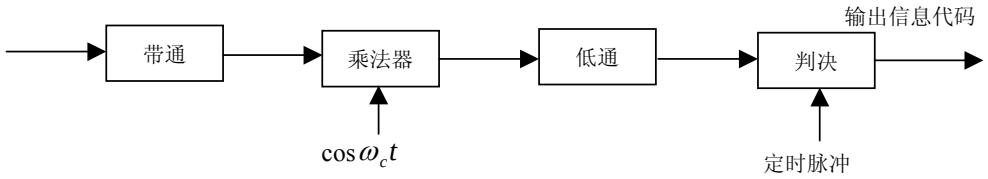


图 5-5 2PSK 相干解调方框图

在非相干解调时，不论是数字信号还是模拟信号都不需要同步载波；只有在相干解调时，才有同步载波提取的问题。另外同步载波的提取在基带传输中也不需要，因为基带传输中根本就没有载波调制和解调的问题。

在模拟通信中，没有位同步的问题，只有载波同步的问题，而且只有接收机采用同步解调时才有载波同步的问题。但在数字通信中一般都存在位同步的问题，不论基带传输还是频带传输。在图 5-4 和图 5-5 中，虽然前者不需要同步载波，而后者需要同步载波，但两者都需要位同步脉冲，即定时脉冲，这个位同步脉冲通过位同步提取电路而获得。在传输数字信号时，由于信号通过信道时受到噪声的影响引起传输波形的失真，因此数字通信系统中接收端都要将接收到的基带信号在抽样判决器中进行抽样判决，以判别是 1 码还是 0 码，抽样判决器判决的时刻就是由位同步脉冲控制。位同步信号一般可以从解调后的基带信号中提取，只有在特殊的情况下才直接从频带信号中提取。而载波同步信号一定要从频带信号中提取。

5.3.2 插入导频位同步法

插入导频法的种类很多，下面主要介绍插入定时导频法和波形变换法两种。

一、插入位定时导频法

前面已经提到位同步信号一般从解调信号中提取。在无线通信中，数字基带信号一般都采用不归零的矩形脉冲，并以此对高频载波作各种调制。解调后得到的也是不归零的矩形脉冲（单极性的或双极性的），码元速率为 f_b，码元宽度为 T_b。对于这种全占空的矩形脉冲，当发送信息中“0”码元和“1”码元出现概率相等时，不论是单极性还是双极性码，其功率谱密度中都没有 f_b 成分，也没有 $2f_b$ 等成分，因此可以在 f_b 处插入位定时导频。

如果基带信号先经过相关编码，则其经相关编码后的功率谱在 $f_b/2$ 处为零，因此可

以在 $f_b/2$ 处插入位定时导频，接收端取出 $f_b/2$ 以后，经过二倍频得 f_b。图 5-6 和图 5-7 分别画出了发端和接收端插入位定时导频和提取位定时导频的方框图。

在发送端要注意插入导频的相位，使导频信号对与数字信号在时间上具有如下的关系：当信号为正、负最大值（即取样判决时刻）时，导频正好是零点，这样避免了导频对信号取样判决的影响。但即使在发端作了这样的安排，接收端仍要考虑抑制导频的问题，这是因为对信道的均衡不一定完善，即所有频率的时延不一定相等，因而信号和导频在发送端具有的时间关系会受到破坏。在接收端抑制插入导频的方法如下：由窄带滤波器取出的导频（ $f_b/2$ ）经过移相和倒相后，再经过相加器把基带数字信号中的导频成分抵消。由窄带滤波器取出导频的另外一路经过移相和放大限幅、微分全波整流、整形等电路，产生位定时脉冲，微分整流电路起到倍频器的作用，因此虽然导频是 $f_b/2$，但定时脉冲的重复频率变为与码元速率相同的 f_b。图 5-7 中两个移相器都是用来消除窄带滤波器引起的相移，这两个移相器可以合用。

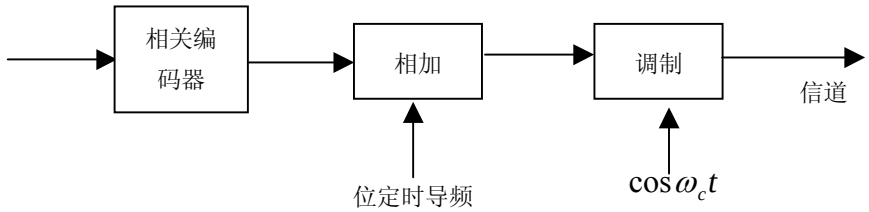


图 5-6　发端插入位定时导频法框图

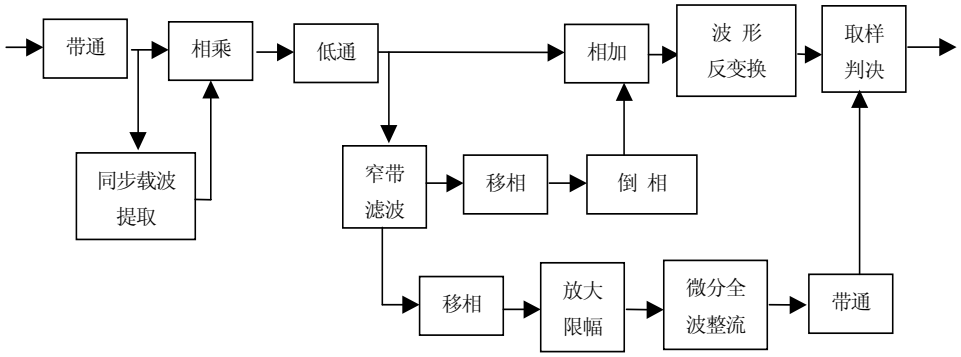


图 5-7 收端提取位定时导频法框图

二、波形变换法（双重调制法）

对于 2PSK 或者 2FSK 信号，可在发送端进行附加的调幅调制，如果附加调制频率

$\Omega = 2\pi f_b$，则在收端采用包络解调就可以提取出 f_b 的成分作为位同步信号。

以 2PSK 信号为例，没有附加调幅时，已调信号为 $\cos[\omega_c t + \varphi(t)]$，其中：

$$\varphi(t) = \begin{cases} 0 & \text{"1"码} \\ \pi & \text{"0"码} \end{cases} \tag{5-8}$$

若用 $\cos\Omega t$ 对它进行附加调制后，得到的已调信号为 $(1+\cos\Omega t)\cos[\omega_c t + \varphi(t)]$，接收端对它进行包络检波，得到包络为 $(1+\cos\Omega t)$，滤除直流成分，即可得 $\Omega = 2\pi f_b$ 的频率成分。

除了上面两种插入位同步信号的方法以外，还可以用时域插入导频的方法，其原理和载波同步中类似。

5.3.3 直接位同步法

直接法提取位同步信号就是直接从接收信号中提取位同步信号，直接提取位同步常用的方法有滤波法和锁相环法。

一、滤波法

全占空的基带随机脉冲序列，不论是单极性还是双极性的，当 $p(0) = p(1) = 0.5$ 时，都没有 f_b、$2f_b$ 等线谱，因此不能直接滤波取出 f_b 成分。但归零的单极性脉冲序列中包含 f_b 的成分，因此只要把基带信号作一些变换，变成单极性归零码，即可直接用窄带滤波器取出 f_b。滤波法的原理如图 5-8 所示。

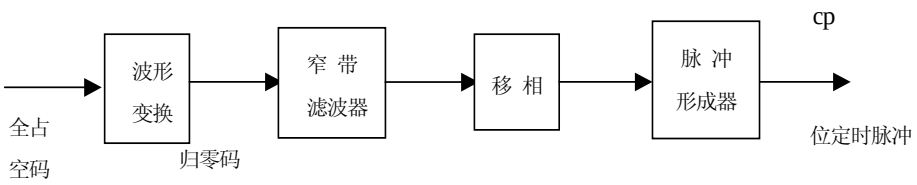


图 5-8　滤波法提取位同步信号框图

二、锁相环法

前面介绍的滤波法中的窄带滤波器可以用简单谐振电路等滤波电路，也可以用锁相环路实现，用锁相环路替代一般窄带滤波器提取位同步信号的方法就是锁相环法。锁相法的基本原理与载波同步原理类似，在接收端利用鉴相器比较接收码元和本地产生的位同步信号的相位，若两者相位不一致（超前或滞后），鉴相器就产生误差信号去调整位同步信号的相位，直至获得精确的位同步为止。在数字通信中常用数字锁相法，主要有微分整流数字锁相，正交积分型数字锁相等。下面对数字锁相法的基本原理和方框图作扼要介绍。

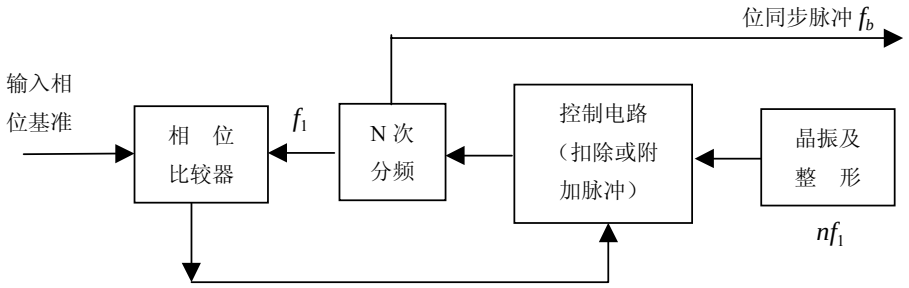


图 5-9　数字锁相环的基本原理图

数字锁相环的基本原理方框图如图 5-9 所示，输入相位基准是接收码元经过零检测（限幅、微分）和单稳电路产生窄脉冲，这些窄脉冲出现的位置精确地位于接收码元的过零点。没有连 0 或连 1 码时，窄脉冲的间隔正好是 T_b，但当接收码元中有连码时，窄脉冲的间隔为 T_b 的整数倍。由于窄脉冲的间隔有时为 T_b，有时为 T_b 的整数倍，因此它不能直接作为位同步信号。输入相位基准与高稳定振荡器产生的经过整形的 n 次分频后的相位脉冲进行比较，由两者相位的超前或滞后，确定扣除或附加一个脉冲（在 T_b 时间内），以调整位同步脉冲的相位。

晶体滤波器经过整形后得到的窄脉冲是周期性的，重复频率为 nf_1，但频率和相位不一定正确，需要进行调整，位同步脉冲是由控制电路输出的脉冲经过 n 次分频得到的，它的重复频率没有经过调整时是 f_1，经过调整以后为 f_b，但在相位上与输入相位基准信号有一个很小的误差。

以上介绍了两类位同步提取方法，即插入导频法和直接位同步法。这两种方法有各自运用的场合：如果要发射的信号中没有 f_b 或 $f_b/2$ 的成分，则必须用插入定时导频法，它的优点在于位同步提取的电路相对来说比较简单，但是由于它的发射信号中有插入的导频

信号，因此发射的效率比较低。如果要发射的信号中含有 f_b 或 $f_b/2$ 的成分，则可以用直接提取同步脉冲的方法，它的优点在于发射的效率较高，但是位同步的提取电路比较复杂。

5.4 群同步法

载波同步解决了同步解调的问题，使得频带信号能够解调为基带信号。而位同步确定数字通信中各个码元的抽样判决时刻，即把每个码元加以区分，使接收端得到一连串的码元序列。这一连串的码元序列代表一定信息，通常由若干个码元代表一个字母（或符号、数字），而由若干个字母组成一个字，若干个字组成一个句。在传输数据时则把若干个码元组成一个码组。群同步的任务是把字、句或码组区分出来。例如一个字由 20 个码元组成，则将码元位同步脉冲频率 f_b 进行 20 次分频，即可得到字的定时脉冲频率。这样看来群同步脉冲的频率是很容易从位同步脉冲经过分频得到的。但每个群的“开头”和“结尾”的时刻，即群同步脉冲的相位是不能直接从同步脉冲中得到的，这个“开头”和“结尾”的时刻确定是群同步需要进一步解决的问题。

5.4.1 群同步法分类

为了确定群同步中“开头”和“结尾”的时刻，即群定时脉冲的相位，通常采用以下两种方法：连贯式插入法和间歇式插入法。连贯式插入法是在每群信息码元的开头发送一个特殊的码组，这个特殊的码组应该不会出现在信息码元序列中，或者是偶然可能出现，但不会重复出现的。此时只要将这个特殊的码组连发几次，收端就能识别出来，收端根据这些特殊的码组的位置就可以确定群的“头”和“尾”，从而实现了群同步。间歇式插入法将群同步码元分散的加入信息中，每隔一定数量信息码元，插入一个群同步码元。

本节首先介绍连贯式插入群同步码法的基本原理，再讨论分散式插入法的基本原理，最后讨论一种称为起止同步的群同步法。

一、连贯式插入法

连贯式插入法的关键是要找出能作为群同步码的特殊码组，这个特殊的码组一方面在信息码元序列中不易出现以便识别，另一方面也要使得识别器尽量简单。目前最常用的群同步码组是“巴克”码。

巴克码是一种具有特殊规律的二进制码组。它的特殊规律是：若一个 n 位的巴克码

$\{x_1, x_2, x_3, \ldots, x_n\}$，每个码元 x_i 只可能取值+1 或-1，则它必然满足条件

$$R(j)=\sum_{i=1}^{n-j} x_i x_{i+j}=\begin{cases} n & \text{当} j=0 \\ 0,+1,-1 & \text{当} 0<j<n \end{cases} \tag{5-9}$$

式中 $R(j)=\sum_{i=1}^{n-j} x_i x_{i+j}$ 称为局部自相关函数。目前已找到的巴克码组如表 5-1 所示。表中“+”表示+1，“−”表示−1。

表 5-1 巴克码组

位数 n	巴 克 码 组
2	+ +；− +
3	+ + −
4	+ + + −；+ + − +
5	+ + + − +
7	+ + + − − + −
11	+ + + − − − + − − + −
13	+ + + + + − − + + − + − +

以 n=7 的巴克码为例，其局部自相关函数如下：

当 $j=0$ 时 $\quad R(j)=\sum_{i=1}^{7} x_i^{\,2}=1+1+1+1+1+1+1=7;$

当 $j=1$ 时 $\quad R(j)=\sum_{i=1}^{6} x_i x_{i+1}=1+1-1+1-1-1=0;$

当 $j=2$ 时 $\quad R(j)=\sum_{i=1}^{5} x_i x_{i+2}=1-1-1-1+1=-1;$

同样可以求出 $j=3,4,5,6,7$ 以及 $j=-1,-2,-3,-4,-5,-6,-7$ 时 $R(j)$ 的值，均为 0 或-1。根据这些值，可以作出 7 位巴克码时，$R(j)$ 与 j 的关系曲线，如图 5-10。由图中可以看出，自相关函数在 $j=0$ 时具有尖锐的单峰特性。局部自相关函数具有尖锐的单峰特性正是连贯式插入群同步码组的主要条件之一。这种尖锐的单峰特性在信号的实际传输中也

具有较强的抗干扰特性。

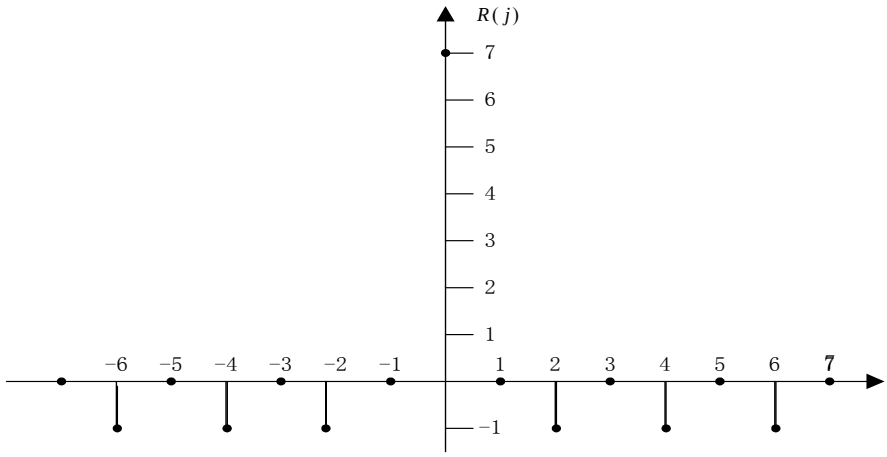


图 5-10　7 位巴克码的局部自相关性

二、间歇式插入法

连贯式插入法主要用在数据传输时，每组数据前面集中插入一组特殊的码字作为群同步，而间歇式插入法比较多的用在多路数字电话系统中。数字电话主要有 PCM 和△M 两种。单路数字电话如果用△M 时，不需要群同步（有时也可以不用位同步）。这是因为△M 系统中只需要一个积分器就可以了。PCM 系统中编、译码器都有定时脉冲，收发端的定时脉冲必须同步，因此一定要有位同步，通常 PCM 都用在多路通信中（特别是电话），多路数字电话系统都需要群同步，而且往往用间歇式插入法。例如某设备中有两路△M 数字电话，群同步就是用 1、0 码相间的群同步码插入第二路话的信息流中。PCM 多路数字电话中群同步码可以连贯式插入也可以间歇式插入，如 32 路数字电话 PCM 中，实际上只有 30 路通电话，另外两路中的一路是作为群同步码用的，另一路字作其他标志信号用，连贯式插入法每帧插入一次。多路数字电话中有时为简化设备，群同步码组不用集中插入的方法，而是将它分散的插入，即每隔一定数量的信息码元，都抽样一次。共有 24 个抽样值，192 个信息码元。例如 24 路 PCM 系统中，一个抽样值用 8 位码表示，此时 24 路电话都抽样一次共有 24 个抽样值，192 个信息码元。192 个信息码元作为一帧，在这一帧插入一个群同步码元，这样一共 193 个码元。由群同步码再得出分路的定时脉冲。

为了便于理解，下面举两个例子加以说明：

在多路 PCM 系统中集中插入群同步码的方法。图 5-11 画出了在 *n* 路 PCM 系统中集中插入群同步码的示意图。设共有 *n* 路，其中有一路专门用来集中插入群同步码和其它标志信号，两者交替插入。（*n*-1）路为话音信号，如果采用 8 位码代表一个抽样值，则群同步码也可以用 8 位码，但也可以用位巴克码（有一位空着）。

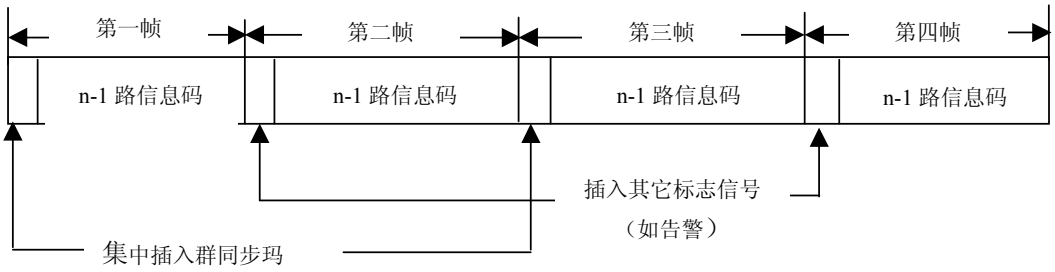


图 5-11　集中插入群同步码示意图

在多路 PCM 系统中分散插入群同步码的方法。图 5-12 中给出了一个在 24 路 PCM 系统中分散插入群同步码的示意图，每帧共 193 个码元（192 个信息码和一个群同步码）。

在图 5-12 中，位同步频率是群同步频率的 193 倍，因此可以由位同步频率经过 193 次分频得到群同步频率。分频后得到的群同步脉冲相位不稳定，在实际的应用中可以采用逐码移位法实现相位的调整。 采用这种方法，群同步的平均建立时间大约为 N^2T_b。

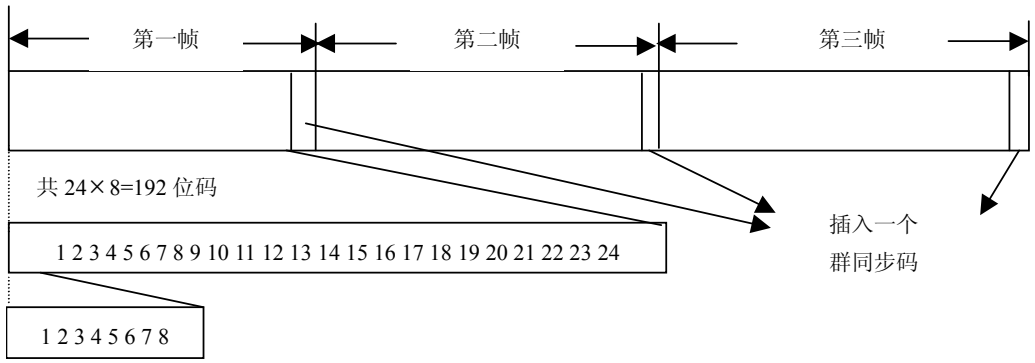


图 5-12　24 路 PCM 系统中分散插入群同步码的示意图

三、起止同步法

除了以上的两种群同步方法，在电传机中广泛使用另一种同步方法，称为起止同步法，它和上面介绍的两种群同步方法有很大的不同。电传机打字机中用 5 个码元代表一个字母（或符号等）在每个字母开始时，先发送一个码元宽度的“无电”脉冲，再传送五个单元的编码信息，接着再发送一个宽度为 1.5 个码元宽度的“有电”脉冲。开头的“无电”脉冲称为“起脉冲”，它起着同步的作用，末尾的“有电”脉冲称为“止脉冲”，它使下一个字母开始之前产生一个间歇。因此一个字母实际上由 7.5 个码元组成。由于这种同步方式中的止脉冲宽度与码元宽度不一致，会给同步数字传输带来不便，另外，在这种起止式同步方式中，7.5 个码元中只有 5 个码元用于传输信息，效率较低，但起止同步的优点是简单。

5.4.2 群同步性能

对群同步系统的主要要求是建立群同步的时间短，建立同步后应该有较强的抗干扰能力。通常用以下三个性能指标来表示同步性能的好坏：（1）漏同步概率；（2）假同步概率；（3）群同步平均建立时间。

下面主要以连贯式插入法为例加以说明。

一、漏同步概率

由于噪声和干扰的影响，会引起群同步码元中一些码元发生错误，从而识别器漏识已发出的群同步码组，出现这种情况的概率称为漏识概率，用符号 p_1 表示。以 n 位巴克码识别器为例，设判决门限为 6，此时七位巴克码中只要有一位码发生错误，七位巴克码全部进入识别器时相加器输出 7 变成了 5，就出现了漏同步，只有一位码也不错才不会发生漏同步。

假设系统的误码率为 p，七位码中一个也不错的概率为 $(1-p)^7$，因此判决门限为 6 时漏同步概率为 $p_1=1-(1-p)^7$。如果为了减少漏同步，判决门限改为 4，此时容许有一个错误，出现一个码元错误的概率为 $c_7^1 p(1-p)^6$。漏同步的概率为 $p_1=1-[(1-p)^7+c_7^1 p(1-p)^6]$。把它写成推广式，设群同步码组的码元数目为 n，判决器容许群同步码组中最大错误码数为 m，则 $p_1=1-\sum_{r=0}^{m} c_n^r p^r (1-p)^{n-r}$。

二、假同步概率

在信息码中也可能出现与所要识别的群同步码组相同的码组，这时识别器会把它误认为群同步码组而出现假同步。出现这种情况的概率就称为假同步概率，用符号 p_2 表示。

计算假同步概率就是计算信息码元中能被判为同步码组的组合数与所有可能的码组数之比。设二进制信息码中 1、0 码等概出现，则由该二进制码元组成 n 位码组的所有可能的码组为 2^n 个，而其中能被判为同步码组的组合数也与 m 有关，若 m=0，只有 c_n^0 个码组能识别；若 m=1，则有 $c_n^0+c_n^1$ 个码组能识别，其余类推。写成普遍式，信息码中能够

被判为同步码组的组合数为 $\sum_{r=0}^{m} c_n^r$，由此得假同步概率的普遍式为：$p_2 = \frac{1}{2^n} \sum_{r=0}^{m} c_n^r$。

由 p_1, p_2 的公式可以看出，$m\uparrow$ 时，$p_1 \downarrow\downarrow$，而 $p_2\uparrow$，两者是矛盾的。另外当 $n\uparrow$ 时，$p_1\uparrow$,而 $p_2\downarrow\downarrow$，两者也是矛盾的。因此对 *m* 和 *n* 的选择要兼顾对 p_1 和 p_2 的要求。

三、同步平均建立时间

对于连贯插入法假设漏同步和假同步都不存在，即 $p_1 = 0, p_2 = 0$。在最不利的情况下，实现群同步最多需要长一点的时间，设一群的码元数为 *N*，则最长的群同步时间为 NT_b。考虑到出现漏同步和假同步时要花费同步建立的时间，由此得群同步的平均建立的时间为 $t_s = (1 + p_1 + p_2)NT_b$ 这个时间与间歇式插入法的 $t_s = N^2 T_b$ 相比，连贯式的同步建立时间要小得多，这就是连贯式插入法得到广泛应用的原因。

5.4.3 PN 码产生器的 MATLAB 设计

伪随机序列或 PN 序列是由 1、0 组成的序列，这样序列的自相关特性与白噪声的自相关特性类似。本节介绍常用 PN 序列的生成方法及其自相关性和互相关性。

目前最常用的二进制伪随机序列就是 *m* 序列，即最大长度移位寄存器序列。其序列长度为 $L = 2^m - 1$ 比特，可用带有线性反馈的 *m* 阶线性移位寄存器产生，*m* 序列线性移位寄存器如图 5-13 所示。

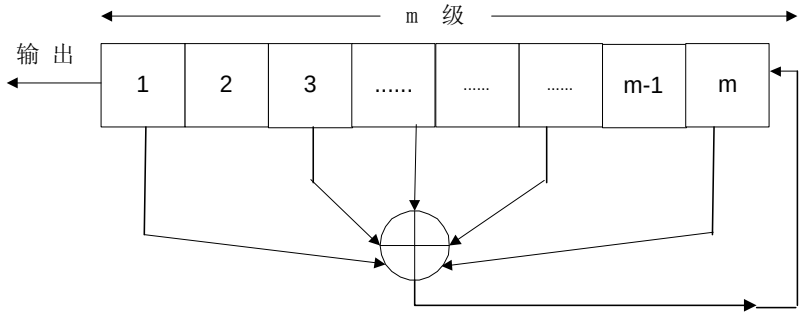


图 5-13　m 序列产生器

产生的伪随机序列的周期为 L。在每一个周期产生的序列中有 2^{m-1} 个 1，$2^{m-1} - 1$ 个 0，即 m 序列具有良好的平衡性。表 6-2 给出了产生 m 序列的移位寄存器的抽头序号。

表 5-2　　产生 *m* 序列的移位寄存器抽头序号列表

m	模 2 累加器级数	*m*	模 2 累加器级数	*m*	模 2 累加器级数
2	1，2	13	1，10，11，13	24	1，18，23，24
3	1，3	14	1，5，9，14	25	1，23
4	1，4	15	1，15	26	1，21，25，26
5	1，4	16	1，5，14，16	27	1，23，26，27
6	1，6	17	1，15	28	1，26
7	1，7	18	1，12	29	1，28
8	1，5，6，7	19	1，15，18，19	30	1，8，29，30
9	1，6	20	1，18	31	1，29
10	1，8	21	1，20	32	1，11，31，32
11	1，10	22	1，22	33	1，21
12	1，7，9，12	23	1，19	34	1，8，33，34

在实际运用中,由于 *m* 序列中互相关性好的码字的个数不是很多，因此人们开始寻找新的码字。目前，人们用的较多的是 gold 码，它是由 *m* 序列中的优选对经过不同的位移后进行模 2 加而形成新的序列，它们具有较好的自相关性和互相关性。

图 5-14 是一个长度为 31 的 gold 码产生器。

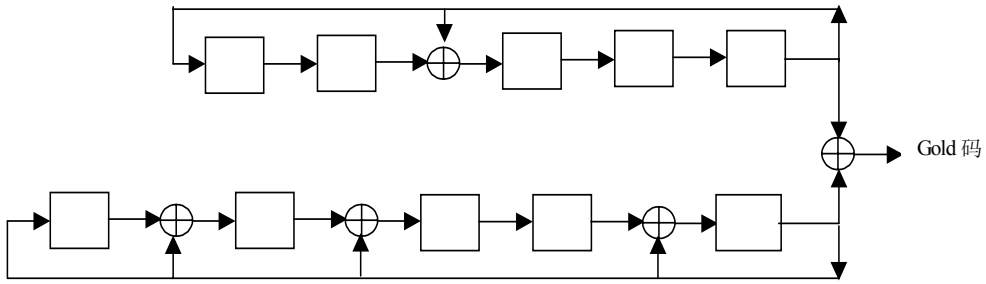


图 5-14 31 位 gold 码的产生器

下面是产生 220 个 1023 点的 gold 码的 MATLAB 程序。

［程序 5-1］gold 码的 MATLAB 程序。

```
m(1)=input('输入一个 PN 码的本原多项式');
m(2)=input('输入另一个本原多项式');
t(1)=m(1);
t(2)=m(2);
```

```
n=(1:2);
%下面是判断输入的本原多项式是几阶的。
for i=1:2
    if   m(i)<13
         n(i)=2;
    elseif   m(i)<23
         n(i)=3;
    elseif   m(i)<45
         n(i)=4;
    elseif   m(i)<103
         n(i)=5;
    elseif   m(i)<211
         n(i)=6;
    elseif   m(i)<435
         n(i)=7;
    elseif   m(i)<1021
         n(i)=8;
    elseif   m(i)<2011
         n(i)=9;
    elseif   m(i)<4005
         n(i)=10;
    elseif   m(i)<10123
         n(i)=11;
    elseif   m(i)<20033
         n(i)=12;
    elseif   m(i)<42103
         n(i)=13;
    elseif   m(i)<100003
         n(i)=14;
    elseif   m(i)<210013
         n(i)=15;
    elseif   m(i)<400011
         n(i)=16;
    elseif   m(i)<1000201
         n(i)=17;
    elseif   m(i)<2000047
         n(i)=18;
    else
         disp('Input number out of the bourn(i)');
```

```
end
  coereg=1:(n(i)+1);%初始化寄存器的系数。
  coereg(n(i)+1)=1; %最高一阶的寄存器系数一定为 1。
  m(i)=m(i)-2^(n(i))*1.25^floor((n(i))/3);
  for k=1:n-1
       coereg(k+1)=floor(m(i)/(2^(n(i)-k)*1.25^floor((n(i)-k)/3)));
       m(i)=m(i)-2^(n(i)-k)*1.25^floor((n(i)-k)/3)*coereg(k+1);
  end
  reg=floor((1:n(i))/n(i));%初始化积存器的状态。
  pnseq=0*(1:(2^n(i)-1));
  for k=1:(2^n(i)-2)
     a=0;
     for j=1:(n(i)-1)
           a=reg(j)*coereg(j)+a;
           reg(j)=reg(j+1);
     end
     reg(n(i))=reg(n(i))*coereg(n(i))+a;
     reg(n(i))=reg(n(i))-2*floor(reg(n(i))/2);
     pnseq(k+1)=reg(1);
   end
%下面是产生两个 PN 序列。
   if i<2
      pnseq1=2*pnseq-1;
   else
      pnseq2=2*pnseq-1;
   end
end
a= n(1)-n(2);
if a==0
     disp('Input PN codes have same length!');
else
     disp('Input PN codes are not in same length!');
     a
     break
     quit;
end

l=2^max(n)-1;
%下面是开始找 gold 码。
```

```
g1=input('输入一个 PN 码的偏移量');
  for i=1:l
      gold1(i)=pnseq1(i)*pnseq2(ceil(rem((i+g1),(l+0.0005))));
  end
```

5.5 位同步群同步的 DSP 实现

［程序 5-2］载波同步、位同步和群同步程序。

设计参数：

- 采样速率为 14400 样点/秒
- 载波频率为 1800Hz
- 符号速率为 2400 符号/秒
- PN 序列码长为 128

编程说明：载波同步、位同步和群同步一般是分开实现的，但是如果采用联合实现方法，则可以大大缩短信号同步时间，从而提高同步性能。本程序利用匹配滤波法在同一接收信号样本中并行提取载波同步、位同步和群同步信息，结合[程序 4-4]可组成一个完整的接收数据解调程序。

```
;-------------------------------------------------------
;       version:            2.0
;       data:               july-05-2001
;       authors:            ren guo chun
;       comment:              original created
;       cpu type:           ti_tms32054xx
;-------------------------------------------------------

;-------------------------------------------------------
;       compiler:           tms320c54x assembler
;       version:            1.02 (pc)
;       include files:    .mmregs
;-------------------------------------------------------

;-------------------------------------------------------
;       functional description:
;       bit and frame syn
;       2400 sym/s
;       fc=1800Hz
```

```
;           fs=14400 sample/s
;-------------------------------------------------------

;-------------------------------------------------------
;           activation
;           activation example:
;           call       syn_task
;-------------------------------------------------------

;-------------------------------------------------------
;           constant:
;           PRE_PN_LK          .set          128       ;for 128 pn syn
;           DODP               .set          4
;           DOPPLE_LOOPK1      .set          32
;-------------------------------------------------------

;-------------------------------------------------------
;           External Data
            .ref       RV_BUFA
            .ref       RV_BUFB
            .ref       RE_TAB
            .ref       DOP_BUF
            .ref       COS_P_BUF
;-------------------------------------------------------

;-------------------------------------------------------
;           temporary data:
;           dp=4
;           cram0
;           cram1
;           cram2
;           cram3
;           cram4
;           cram5
;           cram6
;           cram7
;-------------------------------------------------------

;-------------------------------------------------------
```

```
;           data structure
;           RV_BUFA       .set    1200h           ;buf_l=51h
;           RV_BUFB       .set    1400h           ;buf_l=51h
;           RE_TAB        .set    1500h           ;buf_l=128
;           MACH_RE_B1 .set    5000h            ;BUF_l=128*6
;           MACH_IM_B1 .set    5400h            ;BUF_l=128*6
;           COS_P_BUF     .set    5800h           ;buf_l=20h
;           DOP_BUF       .set    5900h           ;buf_l=20h
;-------------------------------------------------------

;-------------------------------------------------------
;           ram define:
;           RV_FBIP       as RV_BUFA and RV_BUFB    input point
;           RV_FBOP       as RV_BUFA and RV_BUFB    output point
;           MACH_IP       as MACH_RE_B1 AND MACH_IM_B1 input point
;-------------------------------------------------------

;-------------------------------------------------------
;           inut data:
;           RV_BUFA
;           RV_BUFB
;           data format:       16-bit data buffer
;-------------------------------------------------------

;-------------------------------------------------------
;           output data:
;           for second_process
;-------------------------------------------------------

syn_task
            LD          #DODP,DP
            SSBX        SXM                         ;sxm=1
            RSBX        FRCT                        ;frct=0
            RSBX        OVM                          ;ovm=0
            LD          RV_FBIP,A                  ;rv filter in_pointer
            SUB         RV_FBOP,A                   ;rv filter out_pointer
            AND         #RV_BUFL-1,A                ;circular buffer size
            SUB         #20,A                      ;check if have data for syn
            BC          do_mach_filter_end,ALT
```

```
        LD      MACH_IP,A
        ADD     #MACH_RE_B1,A
        STLM    A,AR6                   ;match re filter in_pointer

        LD      MACH_IP,A
        ADD     #MACH_IM_B1,A
        STLM    A,AR7                   ;match im filter in_pointer

        LD      RV_FBOP,A
        ADD     #RV_BUFA,A
        STLM    A,AR2                   ;rv re filter out_pointer

        LD      RV_FBOP,A
        ADD     #RV_BUFB,A
        STLM    A,AR3                   ;rv im filter out_pointer

        ADDM    #1,RV_FBOP              ;inc 1
        ANDM    #RV_BUFL-1,RV_FBOP      ;circular buffer size

        STM     #COS_P_BUF,AR4          ;for sin table read pointer
        STM     #DOP_BUF,AR5            ;for freq. buffer pointer
        STM     #(PRE_PN_LK)*2,AR0      ;for circular address
        STM     #DOPPLE_LOOPK1-1,AR1
        SSBX    FRCT
do_match_m00
        CALL    read_cos_sin            ;read sin and cos data

        LD      CRAM5,T
        MPY     *AR2,A                  ;a*sin
        LD      CRAM6,T
        MAS     *AR3,A                  ;a*sin-b*cos
        STH     A,*AR6+0

        MPY     *AR2,A                  ;a*cos
        LD      CRAM5,T
        MAC     *AR3,A                  ;a*cos+b*sin
        STH     A,*AR7+0
```

```
        BANZ    do_match_m00,*AR1-
        RSBX    FRCT                    ;frct=0

        LD      MACH_IP,A               ;change match filter in_pointer
        ADD     #1,A
        STL     A,MACH_IP
        SUB     #PRE_PN_LK*2-1,A        ;circular buffer size
        NOP
        NOP
        XC      2,AGEQ
        ST      #0,MACH_IP              ;MACH_IP=0

        ST      #0,CRAM7                ;ini cram7=0

do_match_m01
        LD      MACH_IP,A
        ADD     CRAM7,1,A
        ADD     #MACH_RE_B1,A
        STLM    A,AR2                   ;for re match filter in_point

        LD      MACH_IP,A
        ADD     CRAM7,1,A
        ADD     #MACH_IM_B1,A
        STLM    A,AR3                   ;for re match filter in_point

        STM     #3,AR0
        STM     #RE_TAB,AR4
        RPTZ    A,#(PRE_PN_LK)-1
        MAC     *AR2+0%,*AR4+,A         ;match re filtering
        STH     A,-0,CRAM2
        STM     #RE_TAB,AR4
        RPTZ    A,#(PRE_PN_LK)-1
        MAC     *AR3+0%,*AR4+,A         ;match im filtering
        STH     A,-0,CRAM3

        SQUR    CRAM2,A                 ;A=a*a
        SQURA   CRAM3,A                 ;A=a*a+b*b
        STH     A,CRAM5                 ;save   filter |c| data
```

```
        LD      CRAM5,A                 ;compare data
        SUB     MAX_DATA,A
        BC      do_match_m02,ALEQ
        LD      CRAM5,A
        STL     A,MAX_DATA              ;save max data
        LD      MACH_COUNT,A
        STL     A,MAX_ADR               ;save max data address
do_match_m02

        LD      CRAM7,A
        ADD     #(PRE_PN_LK)*2,A
        STL     A,CRAM7
        SUB     #(PRE_PN_LK)*2*DOPPLE_LOOPK1,A
        BC      mach_256_m21,ALT        ;check if all freq. do over

        ADDM    #1,MACH_COUNT
        LD      MACH_COUNT,A
        SUB     #PRE_PN_LK*2,A
        BC      do_match_m03,ANEQ
        ST      #0,MACH_COUNT
        CALL    second_process          ;find syn address go to
do_match_m03                            ;second process
        RET

read_cos_sin
        LD      *AR4,A                  ;calculate sin and cos in cram5 cram6
        AND     #3FH,A
        STL     A,CRAM0
        LD      CRAM0,9,A
        STL     A,CRAM0                 ;load low 6 bit
        LD      *AR4,10,A
        STH     A,CRAM1                 ;load high 9 bit

        LD      CRAM1,A
        ADD     #sin_512_tab,A          ;read sin_table
        READA   CRAM2                   ;first data d1
        ADD     #1,A
        READA   CRAM3                   ;second data d2
        LD      CRAM3,A
```

```
        SUB     CRAM2,A
        STL     A,CRAM4                 ;d1-d2
        LD      CRAM0,T
        MPY     CRAM4,A
        ADD     CRAM2,16,A
        STH     A,CRAM5                 ;sin=(d1-d2)*low_6_bit
        LD      CRAM1,A
        ADD     #80,A
        AND     #1FFH,A
        ADD     #sin_512_tab,A
        READA   CRAM2                   ;first data d3
        ADD     #1,A
        READA   CRAM3                   ;second data d4
        LD      CRAM3,A
        SUB     CRAM2,A
        STL     A,CRAM4                 ;d3=d4
        MPY     CRAM4,A
        ADD     CRAM2,16,A
        STH     A,CRAM6                 ;cos=(d3-d4)*low_6_bit

        LD      *AR5,A                  ;change sin table read pointer
        ADD     *AR5,A
        ADD     *AR5+,A

        ADD     *AR4,A
        BC      read_cos_sin_m01,AGEQ
        ADD     #1,A
        ADD     #7FFFH,A
read_cos_sin_m01
        AND     #7FFFH,A
        STL     A,*AR4+
        RET

        .end
```

第六章 信道编码器的 DSP 实现

数字信号在无线信道中传输时，由于受到各种信道条件的影响，如多径、衰落、干扰和噪声，接收信号不可避免地会发生错误。在已知信噪比的情况下，为达到一定地误比特率指标，首先应合理设计基带信号，选择合适的调制、解调方式，采用均衡手段，使误比特率尽可能的降低。但是，若误比特率仍不能满足要求，则必须采用信道编码技术来进一步地降低误比特率，以满足指标的要求。

差错控制编码的基本思想是：在发送端被传输的信息序列上附加一些监督码元，这些多余的码元与信息码元之间以某种确定的规则互相关联。一旦传输过程中发生错误，则信息码元与监督码元之间的关系将受到破坏，在接收端按照既定的规则检验信息码元与监督码元之间的关系，就可以达到发现和纠正错误的目的。

差错控制编码按照不同的分类准则可以分为多种类型，如：

按照信息码元和附加的监督码元之间的检验关系可以分为线性码和非线性码。若信息码元与监督码元之间为线性关系，即满足一组线性方程组，则称为线性码。反之，若两者不存在线性关系，则称为非线性码。

按照信息码元和监督码元之间的约束方式不同可以分为分组码和卷积码。在分组码中，编码后的码元序列每 n 个分为一组，其中 k 个是信息码元，r 个是附加的监督码元，$r=n-k$ 。监督码元仅与本码组的信息码元有关，而与其它码组的信息码元无关。与此不同的是，卷积码虽然编码后序列也划分为码组，但是其监督码元不仅与本组信息码元有关，而且与前面码组的信息码元也存在约束关系。

按照信息码元在编码后是否保持原来的形式不变，可以划分为系统码和非系统码。由于系统码的编码和译码相对简单，因而得到了广泛的应用。

本章将在概述线性分组码、卷积码基本原理和设计方法的基础上，讨论其 MATLAB、DSP 实现。

6.1 线性分组码的 DSP 实现

线性分组码是分组码中最重要的一类码，它是讨论各类码字的基础。线性分组码由一组长度固定的称为码字的矢量构成。码字的长度等于矢量元素的个数，用 n 表示。码字的元素选自 q 个元素组成的字符集。若字符集由 0、1 两个元素组成时，该码为二进制码。若字符集由 q 个元素(q>2)组成时，该码字为多进制码。

6.1.1 线性分组码的基本原理和设计方法

一、分组码的基本原理

长度为 n 的二进制分组码有 2^n 种可能的码字。从这 2^n 个码字中可以选择 2^k 个码字 $(k<n)$ 组成一种码。这样，一个 k 比特的信息分组就可以映射为一个长度为 n 的码字，这个码字是上面选择的 2^k 个码字之一。这样得到的分组码称为 (n,k) 码，定义 $R_c=k/n$ 为码率。

编码和解码的功能最终体现为对码字进行乘、加算术运算，算术运算必须服从元素所在字符集代数域的规则。一般地，在元素集合构成的域 F 中，对元素定义了加法和乘法两种算术运算，它们满足以下特性：

- 加法：

(1) 集合 F 在加法运算下是封闭的，即若 $a,b\in F$，必有 $a+b\in F$；

(2) 满足加法结合律，即如果 $a,b,c\in F$，则 $a+(b+c)=(a+b)+c$；

(3) 满足加法交换律，即 $a+b=b+a$；

(4) 集合中一定包含一个零元素，满足 $a+0=a$；

(5) 集合中的每个元素都有其对应的逆元素。若 b 是一个元素，那么其逆元素记为 $-b$。两个元素相减，如 $a-b$，定义为 $a+(-b)$。

- 乘法：

(1) 集合 F 在乘法运算下是封闭的，即若 $a,b\in F$，必有 $ab\in F$；

(2) 满足乘法结合律，即 $a(bc)=(ab)c$；

(3) 满足乘法交换律，即 $ab=ba$；

(4) 满足乘法分配律，即 $(a+b)c=ac+bc$；

(5) 集合中一定包含一个单位元，使得对于任意 $a\in F$，满足 $a(1)=a$；

(6) 除零元素外，集合 F 的每个元素都存在逆元素，若 $b\in F(b\neq 0)$，则其逆元素定义为 b^{-1}。两个元素相除 a/b，定义为 ab^{-1}。

我们知道，实数域和复数域含有无穷多个元素，而码字是由有限多个元素的域构成的，具有 q 个元素的有限域通常称为迦罗华域，用 GF(q) 表示。每个域必须有一个零元素和一个单位元，最简单的域是二元域 GF(2)。一般地，若 q 是素数，则可以构成一个由元素 {0, 1, …, q−1} 组成的 q 元域 GF(q)。在域 GF(q) 中定义的加、乘运算是模 q 运算。另外，当 q 为素数的幂时，也可以构成有限域，如当 $q=p^m$ 时（p 为素数，m 为任意正整数），

可以将 GF(p)域扩展为 GF(p^m)，扩展域中元素的加、乘运算也基于模 p 运算。

二、生成矩阵 G 和监督矩阵 H

设 $\mathbf{X}_m=[x_{m1},x_{m2},\cdots,x_{mk}]$ 为输入编码器的 k 个信息比特，编码器的输出记为：

$$\mathbf{C}_m=[c_{m1},c_{m2},\cdots,c_{mn}] \tag{6-1}$$

则编码运算可以用矩阵的形式表示如下：

$$\mathbf{C}_m=\mathbf{X}_m\mathbf{G} \tag{6-2}$$

其中称为该码的生成矩阵：

$$\mathbf{G}=\begin{bmatrix}\mathbf{g}_1\\ \mathbf{g}_2\\ \vdots\\ \mathbf{g}_k\end{bmatrix}=\begin{bmatrix}g_{11} & g_{12} & \cdots & g_{1n}\\ g_{21} & g_{22} & \cdots & g_{2n}\\ \vdots & \vdots & & \vdots\\ g_{k1} & g_{k2} & \cdots & g_{kn}\end{bmatrix} \tag{6-3}$$

任何码字都是生成矩阵 $\mathbf{G}$ 的矢量 $\{\mathbf{g}_i\}$ 的线性组合，即：

$$\mathbf{C}_m=x_{m1}\mathbf{g}_1+x_{m2}\mathbf{g}_2+\cdots+x_{mk}\mathbf{g}_k \tag{6-4}$$

(n,k) 分组码的生成矩阵可以通过运算简化为系统形式：

$$\mathbf{G}=[\mathbf{I}_k\mathbf{P}]=\begin{bmatrix}1 & 0 & 0 & \cdots & 0 & p_{11} & p_{12} & \cdots & p_{1(n-k)}\\ 0 & 1 & 0 & \cdots & 0 & p_{21} & p_{22} & \cdots & p_{2(n-k)}\\ \vdots & \vdots & \vdots & & \vdots & \vdots & \vdots & & \vdots\\ 0 & 0 & 0 & \vdots & 1 & p_{k1} & p_{k2} & \cdots & p_{k(n-k)}\end{bmatrix} \tag{6-5}$$

其中，$\mathbf{I}_k$ 是 $k\times k$ 维矩阵，$\mathbf{P}$ 是 $k\times(n-k)$ 维矩阵。由系统形式的生成矩阵生成的码字称为系统码，每个码字的前 k 位与要发送的信息比特相同，其余 $n-k$ 由 $\mathbf{P}$ 决定，是前 k 个信息的线性组合，这 $n-k$ 个冗余位叫做校验位。

(n,k) 码的任意码字都应正交于监督矩阵 $\mathbf{H}$ 的每一行，即 $\mathbf{C}_m\mathbf{H}'=\mathbf{0}$。

其中，$\mathbf{0}$ 代表由 $n-k$ 个元素组成的全零矢量，$\mathbf{C_m}$ 是 (n,k) 码的一个码字。上式对 (n,k) 码的每个码字都成立，于是 $\mathbf{GH}'=\mathbf{0}$。其中，$\mathbf{0}$ 代表由全零元素组成的 $k\times(n-k)$ 维矩阵。

当 (n,k) 码为系统码时，其监督矩阵为：

$$\mathbf{H}=\left[-\mathbf{P}'\mathbf{I}_{n-k}\right] \tag{6-6}$$

下面，我们来分析如何利用监督矩阵$\mathbf{H}$来检查接收码字中的错误。设发送码组为$\mathbf{C}_m=[c_{m1},c_{m2},\cdots,c_{mn}]$，接收码组为$\mathbf{B}_m=[b_{m1},b_{m2},\cdots,b_{mn}]$，错误图样为$\mathbf{E}=[e_1,e_2,\cdots,e_n]$，$\mathbf{E}=\mathbf{B}_m-\mathbf{C}_m$，错误图样$\mathbf{E}$中哪一位不为0，则说明接收码组中相应的码元发生了错误。令$\mathbf{S}=\mathbf{B}_m\mathbf{H}'$，称为伴随式或校正子。

$$\mathbf{S}=\mathbf{B_m}\mathbf{H}'=\mathbf{(C_m}+\mathbf{E)}\mathbf{H}'=\mathbf{C_m}\mathbf{H}'+\mathbf{E}\mathbf{H}'=\mathbf{E}\mathbf{H}' \tag{6-7}$$

由于$\mathbf{S}$和$\mathbf{E}$之间有确定的线性变换关系，因此$\mathbf{S}$即能代表$\mathbf{B}$中错误的情况，从而完成纠错的功能。

三、汉明码

汉明码是一种高码率的纠单个错误的线性分组码。汉明码既有二进制的，也有非二进制的，这里仅讨论二进制汉明码的性质。汉明码具有的共同特性是：

$$(n,k)=(2^m-1,2^m-1-m) \tag{6-8}$$

其中，m为任意正整数。例如m=3时，有（7,4）汉明码。

汉明码的监督矩阵$\mathbf{H}$具有特殊的性质。一个(n,k)码的监督矩阵有$n-k$行和n列，对于二进制(n,k)汉明码，$n=2^m-1$列包含由$n-k=m$个二进制码元组成的所有可能的非全零组合。如(7,4)汉明码，其校验矩阵由(001)、(010)、(011)、(100)、(101)、(110)、(111）组成。由此，按照系统码监督矩阵的格式可以很容易的组合出系统汉明码的监督矩阵$\mathbf{H}$，从而进一步得到相应的生成矩阵$\mathbf{G}$。通过观察可知，$\mathbf{H}$矩阵中各列都是线性无关的。在$m>1$时，有可能找到$\mathbf{H}$的3个列，它们之和为零矢量，由此可得(n,k)汉明码的最小距离$d_{\min}=3$。

以（7,4）系统汉明码为例，由上面的分析可知，其监督矩阵$\mathbf{H}$为：

$$\mathbf{H}=\begin{bmatrix}1&1&1&0&1&0&0\\1&1&0&1&0&1&0\\1&0&1&1&0&0&1\end{bmatrix} \tag{6-9}$$

对应的生成矩阵$\mathbf{G}$为：

$$\mathbf{G}=\begin{bmatrix}1&0&0&0&1&1&1\\0&1&0&0&1&1&0\\0&0&1&0&1&0&1\\0&0&0&1&0&1&1\end{bmatrix} \tag{6-10}$$

由编码生成规则，我们得到了下面的（7,4）汉明码字表。

表 6-1 （7,4）汉明码字表

信息元	监督元
0 0 0 0	0 0 0
0 0 0 1	0 1 1
0 0 1 0	1 0 1
0 0 1 1	1 1 0
0 1 0 0	1 1 0
0 1 0 1	1 0 1
0 1 1 0	0 1 1
0 1 1 1	0 0 0
1 0 0 0	1 1 1
1 0 0 1	1 0 0
1 0 1 0	0 1 0
1 0 1 1	0 0 1
1 1 0 0	0 0 1
1 1 0 1	0 1 0
1 1 1 0	1 0 0
1 1 1 1	1 1 1

根据伴随式和错误图样的线性变换关系，得到了（7,4）汉明码伴随式与错误图样间的对应关系，如表 6-2 所示。

表 6-2 （7,4）汉明码伴随式与错误图样的对应关系

错误码位	**E**							**S**		
	e_6	e_5	e_4	e_3	e_2	e_1	e_0	s_2	s_1	s_0
/	0	0	0	0	0	0	0	0	0	0
b_0	0	0	0	0	0	0	1	0	0	1
b_1	0	0	0	0	0	1	0	0	1	0
b_2	0	0	0	0	1	0	0	0	1	1
b_3	0	0	0	1	0	0	0	1	0	0
b_4	0	0	1	0	0	0	0	1	0	1
b_5	0	1	0	0	0	0	0	1	1	0
b_6	1	0	0	0	0	0	0	1	1	1

这样由表 6-2 就可以根据伴随式的值来确定错误的位置，以到达纠正错误的目的。

6.1.2 （7,4）汉明码的 MATLAB 实现

MATLAB 提供了 encode 函数和 decode 函数来完成六种主要的差错控制编码：汉明码、线性分组码、循环码、BCH 码、RS 码和卷积码的编码和译码的工作。我们首先介绍汉明码的编码和译码。

一、Encode 函数

功能：编码函数

语法为：code=encode(**msg**, *N*, *K*)

说明：该函数对二进制信息 **msg** 进行汉明编码，*K* 为信息位长度，*N* 为码字长度。**msg** 是一个 *K* 列的矩阵。

二、Decode 函数

功能：译码函数

语法：code=decode(**code**, *N*, *K*)

说明：该函数对接收到的码字译码，恢复出原始的信息，译码参数和方式必须和编码时采用的严格相同。

三、Hammgen 函数

功能：汉明码生成矩阵和校验矩阵产生函数

语法：**h**=hammgen(*M*);

[**h**, **g**]=hammgen(*M*);

[**h**, **g**, *N*, *K*]=hammgen(*M*);

说明：该函数的功能是产生汉明码的生成矩阵和校验矩阵，其中 $M=N-K$ 为校验位的长度，**h** 为汉明码的校验矩阵，**g** 为汉明码的生成矩阵。

下面，以（7,4）汉明码为例，给出一个完整的编译码程序。

［程序 6-1］（7,4）汉明码编译码的 MATLAB 程序。

```
K=4;
N=7;
msg=randint(K*200,1,2);                          %信息产生
code=encode(msg,N,K);                            %汉明编码
code_noise=rem(code+rand(N*200)>0.95,2);         %加噪声
rcv=decode(code_noise,N,K);                      %汉明译码
disp([‘Error rate in the received code:’…
num2str(symerr(code,code_noise)/length(code))])
disp([‘Error rate after decode:’…
num2str(symerr(msg,rcv)/length(msg))])
```

6.1.3 线性分组码编译码的 DSP 实现

［程序 6-2］该程序完成了格雷码的编码。

编程说明：格雷码属于线性分组码，下面将给出一个完整的（24，12）格雷码编码的 DSP 实现程序，其生成多项式为：

$$g(x)=x^{11}+x^{9}+x^{7}+x^{6}+x^{5}+x+1$$

输入的 12 比特数据逐比特读出相应的生成多项式，并进行异或运算得到监督元。

```
;-------------------------------------------------------
;          version:          1.0
;          data:             july-11-1999
;          authors:          ren guo chun
```

```
;       comment:            original created
;       cpu type:           ti_tms32054xx
;-------------------------------------------------------

;-------------------------------------------------------
;       compiler:           tms320c54x assembler
;       version:            1.02 (pc)
;       include files:   .mmregs
;-------------------------------------------------------

;-------------------------------------------------------
;       functional description:
;       golay code
;-------------------------------------------------------

;-------------------------------------------------------
;       activation
;       activation example:
;       call      golay_code
;-------------------------------------------------------

;-------------------------------------------------------
;       External Data
        .ref      CRAM2
;-------------------------------------------------------

;-------------------------------------------------------
;       temporary data:
;       dp=4
;       cram0
;       cram1
;       cram2
;       cram3
;       cram4
;       cram5
;       cram6
;       cram7
;-------------------------------------------------------
```

```
;-----------------------------------------------------
;           inut data:
;           CRAM2
;           data format:      12-bit data
;-----------------------------------------------------

;-----------------------------------------------------
;           output data:
;           cram2
;           cram3
;           data format:      12-bit data
;-----------------------------------------------------

golay_code:
            LD          CRAM2,A
            STL         A,CRAM4                     ;load 12-bit data
            ST          #0,CRAM3
            ST          #gen_code_tab,CRAM1         ;set read table pointer
            STM         #12-1,BRC                   ;repeat 12 times
            RPTB        golay_code_m00-1

            LD          CRAM1,A
            READA       CRAM0                       ;read h data
            ADD         #1,A                        ;inc 1
            STL         A,CRAM1                     ;save read pointer
            BITF        CRAM3,BIT_04
            BC          golay_code_m01,NTC          ;set tc=0
            LD          CRAM3,A
            XOR         CRAM0,A                     ;xor
            STL         A,CRAM3
golay_code_m01
            LD          CRAM4,A                     ;sftl 1
            STL         A,1,CRAM4
golay_code_m00

            RET

;           for gre_cod gen
gen_code_tab:
```

```
        .word   0AE3H,0F92H,0D2BH,0C76H,0CD9H,066DH
        .word   0337H,0B78H,05BCH,02DEH,0B8DH,05C7H
        .end
```

［程序 6-3］格雷码的译码程序。

编程说明：该程序与［程序 6-2］相结合，完成格雷码编译码功能。它采用查表法进行格雷码译码，该方法译码速度快，并能同时得到接收码字的错误位置和错误个数。

```
;-------------------------------------------------------
;       version:            1.0
;       data:               july-11-1999
;       authors:            ren guo chun
;       comment:              original created
;       cpu type:           ti_tms32054xx
;-------------------------------------------------------

;-------------------------------------------------------
;       compiler:           tms320c54x assembler
;       version:            1.02 (pc)
;       activation:         asm500 -s fir.asm
;       include files:    .mmregs
;-------------------------------------------------------

;-------------------------------------------------------
;       functional description:
;       golay decode
;-------------------------------------------------------

;-------------------------------------------------------
;       activation
;       activation example:
;       call      golay_decode
;-------------------------------------------------------

;-------------------------------------------------------
;       External Data
        .ref      CRAM2
        .ref      CRAM3
```

```
        .ref    dec_tab
;-------------------------------------------------------

;-------------------------------------------------------
;       temporary data:
;       dp=4
;       cram0
;       cram1
;       cram2
;       cram3
;       cram4
;       cram5
;       cram6
;       cram7
;-------------------------------------------------------

;-------------------------------------------------------
;       inut data:
;       CRAM2       12bit data
;       cram3       12bit data
;       data format:    12-bit data
;-------------------------------------------------------

;-------------------------------------------------------
;       output data:
;       cram6       12bit data
;       data format:    12-bit data
;-------------------------------------------------------


gray_decode
        LD      CRAM3,A
        STL     A,CRAM5         ;load 12-bit data
        LD      CRAM2,A
        STL     A,CRAM6         ;load 12-bit data
        CALL    gray_code       ;do gray code
        LD      CRAM2,A
        XOR     CRAM5,A         ;xor
        ADD     #dec_tab,A
```

```
        READA    CRAM2                ;read error table

        LD       CRAM2,A
        XOR      CRAM6,A              ;correct error
        AND      #0FFFH,A
        STL      A,CRAM6              ;save 12-bit data
        RET

;           for gre_cod gen
gen_code_tab:
        .word    0AE3H,0F92H,0D2BH,0C76H,0CD9H,066DH
        .word    0337H,0B78H,05BCH,02DEH,0B8DH,05C7H

end
```

6.2 里德－索罗码的 DSP 实现

里德－索罗码简称 RS 码，它是一类非二进制 BCH 码。非二进制 BCH 码由一组固定长度的码字构成，其码字的每个码元是从 q 个符号的集合{0，1,2，…，q-1｝中选出的。通常 $q=2^m$，即 m 位信息比特映射为 q 个符号之一。对于 (n,k) RS 码，输入信息分成 $k\times m$ 比特一组，每组包含 k 个符号，每个符号由 m 个比特组成。在各种非二进制线性分组码中，RS 码是最重要的一种。

6.2.1 里德－索罗码的基本原理和设计方法

二进制码是在 GF(2)上的，而非二进制码则是在 $\mathrm{GF}(q)(q\neq 2)$ 上。这里，我们选择 GF(16)域来进行研究，域中的 16 个元素可以用 4 比特符号来表示，见表 6-3。

令 $a\in$GF(16)是本原域元素，它是 x^4+x+1 的根，则：

$$a^4+a+1=0 \tag{6-11}$$

RS 码是 $\mathrm{GF}(q)(q\neq 2)$ 上码长 $n=q-1$ 的本原 BCH 码。对于一个纠 t 个错误的 (n,k) RS 码，其参数表示如下：

码长：　　$n = q - 1 = 2^m - 1$ 符号或 $m(2^m\text{-}1)$比特

信息段：　　k 符号　　或 mk 比特

监督段：　　$n-k=2t$ 符号或 $m(n-k)$比特

最小码距　　$D_{\min} = n - k + 1 = 2t + 1$ 或 $m(2t+1)$比特

表 6-3　以 $x^4 + x + 1$ 为模的 GF(16)的元素

本原域元素表示	a 多项式表示	4 比特符号表示
0	0	0 0 0 0
a^0	1	0 0 0 1
a	a	0 0 1 0
a^2	a^2	0 1 0 0
a^3	a^3	1 0 0 0
a^4	$a+1$	0 0 1 1
a^5	$a^2 + a$	0 1 1 0
a^6	$a^3 + a^2$	1 1 0 0
a^7	$a^3 + a + 1$	1 0 1 1
a^8	$a^2 + 1$	0 1 0 1
a^9	$a^3 + a$	1 0 1 0
a^{10}	$a^2 + a + 1$	0 1 1 1
a^{11}	$a^3 + a^2 + a$	1 1 1 0
a^{12}	$a^3 + a^2 + a + 1$	1 1 1 1
a^{13}	$a^3 + a^2 + 1$	1 1 0 1
a^{14}	$a^3 + 1$	1 0 0 1
a^{15}	1	0 0 0 1

RS 码特别适合于纠正突发错误，它可以纠正的错误图样有：

总长度为 $b_1=(t\text{-}1)m+1$ 比特的单个突发

总长度为 $b_2=(t\text{-}3)m+3$ 比特的两个突发

⋮

总长度为 $b_i=(t\text{-}2i+1)m+2i-1$ 比特的 i 个突发

RS 码还具有良好的距离特性，并且存在有效的硬判决译码算法，适用于需要长码的应用场合，如美国蜂窝数字分组数据系统（CDPD）中采用的就是 $m=6$ 的（63,47）RS 码。

一、RS 码的编码

对于一个能纠正 t 个错误的 RS 码，其生成多项式的格式为：

$$g(x) = (x + a)(x + a^2)\cdots(x + a^{2t}) \tag{6-12}$$

其中，a^i 是 GF(q) 中的元素。

令：

$$d(x) = c_{n-1}x^{n-1} + c_{n-2}x^{n-2} + \cdots + c_{2t+1}x^{2t+1} + c_{2t}x^{2t} \tag{6-13}$$

为信息多项式，且令：

$$p(x) = c_0 + c_1 x + \cdots + c_{2t-1}x^{2t-1} \tag{6-14}$$

为校验多项式。其中 c_i 为 GF(q) 上的域元素。因此，已编码的 RS 码多项式为：

$$c(x) = d(x) + p(x) = \sum_{i=0}^{n-1} c_i x^i \tag{6-15}$$

当且仅当由域元素 $(c_0, c_1, \cdots, c_{n-1})$ 组成的向量是生成多项式 $g(x)$ 的倍数时，此向量才能作为码字。

RS 码的编码过程与 BCH 码一样，用 $d(x)$ 除以 $g(x)$ 来得到 $p(x)$，即：

$$d(x) = g(x)q(x) + r(x) \tag{6-16}$$

其中，$q(x)$ 为商多项式，$r(x)$ 为余数多项式。

这样，码多项式可被记为：

$$c(x) = p(x) + g(x)q(x) + r(x) \tag{6-17}$$

若取校验多项式 $p(x) = -r(x)$，那么：

$$c(x) = g(x)q(x) \tag{6-18}$$

为了确保码多项式是生成多项式的倍数，关键在于通过除法操作来获得 $p(x)$，可以通过多项式除法电路来完成，如图 6-1 所示，其中的乘法和加法都是 GF(q) 域上的。

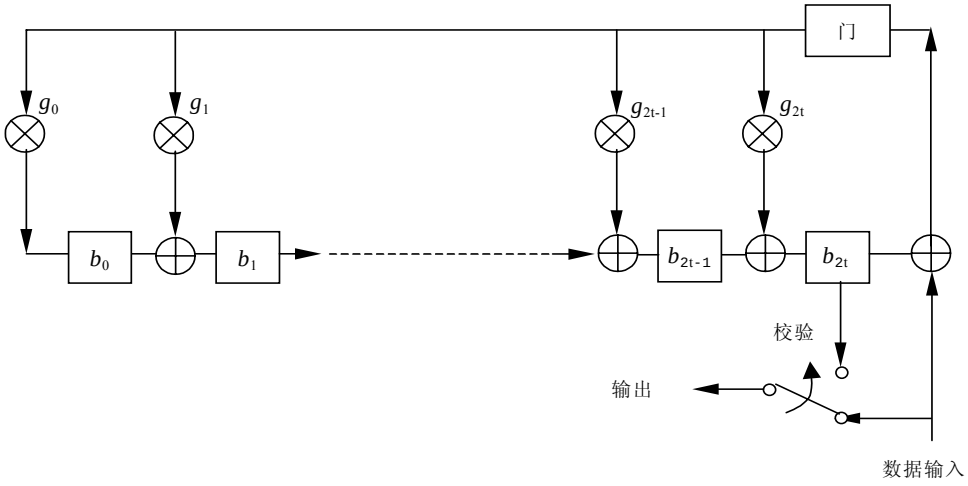


图 6-1 RS 码编码电路

编码开始时，所有寄存器清零，且开关接至数据处。信息多项式的系数 c_{n-1} 到 c_{2t} 被顺序移入编码电路，同时送至输出端。当 c_{2t} 被送入编码电路后，开关切换到校验处，且连接反馈电路的门将断开，不再提供反馈，这时，寄存器 b_0 到 b_{2t-1} 中的内容就是所需要的校验位 p_0 到 p_{2t-1}，将其顺序移出至输出就可以完成 RS 编码。

例 6-1 构造一个能纠正 3 个错误符号，码长为 15，m=4 的 RS 码。求生成多项式和编码电路。

解 当 t=3，最小码距 $D_{\min}$=7，信息元长度 K=9，该码为（15，9）RS 码，其生成多项式为：

$$g(x) = (x+a)(x+a^2)(x+a^3)(x+a^4)(x+a^5)(x+a^6)$$
$$= x^6 + a^{10}x^5 + a^{14}x^4 + a^4x^3 + a^6x^2 + a^9x + a^6$$

其编码电路图如图 6-2 所示。

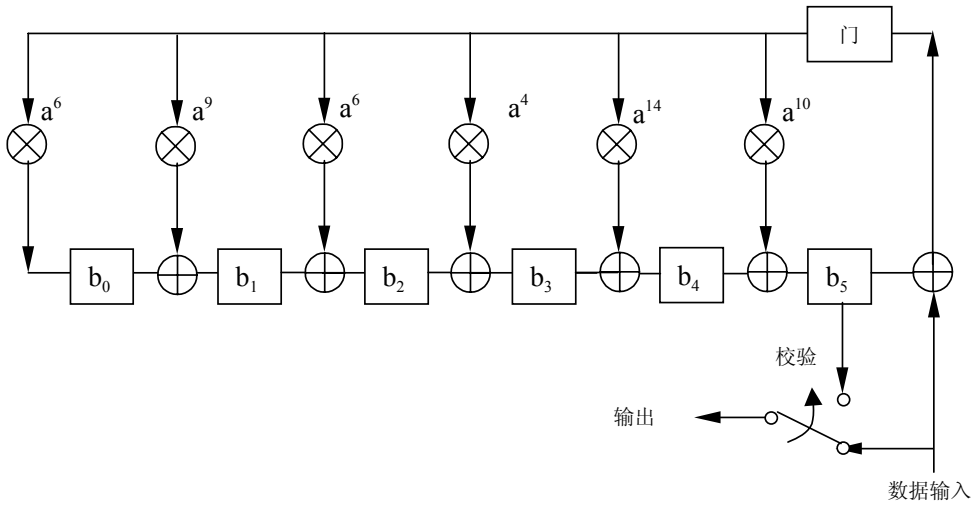


图 6-2 （15，9）RS 码编码电路图

二、RS 码的译码

假设发送码字为：

$$c(x) = v_0 + v_1 x + \cdots + v_{n-1} x^{n-1} \tag{6-19}$$

经过信道后，接收到的码字 $r(x)$为：

$$r(x) = r_0 + r_1 x + \cdots + r_{n-1} x^{n-1} \tag{6-20}$$

误差多项式 $e(x)$为 $c(x)$与 $r(x)$之差，即：

$$e(x) = r(x) - c(x) = e_0 + e_1 + \cdots + e_{n-1} x^{n-1} \tag{6-21}$$

定义 $2t$ 个部分伴随式 $S_i = r(\alpha^i)$ (1≤i≤2t),由于 $\alpha^1, \alpha^2, \cdots, \alpha^{2t}$ 均为码字 $c(x)$的根，即 $c(\alpha^i) = 0$ (1≤i≤2t)，因此：

$$S_i = r(\alpha^i) = c(\alpha^i) + e(\alpha^i) = e(\alpha^i) \tag{6-22}$$

由式(6-22)可知部分伴随式 S_i 只与误差多项式 $e(x)$有关。

假设接收码字中存在 k 个错误（$k<t$），其位置分别在 $x^{j1}, x^{j2}, \cdots, x^{jk}$ 处，其中 $0 \le j_1 < j_2 < \cdots < j_k \le n-1$，令 x^{ji} 处的误差大小为 e_{ji}，则 $e(x)$为：

$$e(x) = e_{j_1} x^{j_1} + e_{j_2} x^{j_{12}} + \cdots + e_{j_k} x^{j_k} \tag{6-23}$$

定义差错位置 $\beta_i = \alpha^{ji}$，i=1,2,⋯,k，可以得到下列 $2t$ 个部分伴随式的方程组：

$$S_1 = e_{j_1}\beta_1 + e_{j_2}\beta_2 + \cdots + e_{j_k}\beta_k$$

$$S_2 = e_{j_1}\beta_1^{\ 2} + e_{j_2}\beta_2^{\ 2} + \cdots + e_{j_k}\beta_k^{\ 2}$$

$$\vdots$$

$$S_{2t} = e_{j_1}\beta_1^{\ 2t} + e_{j_2}\beta_2^{\ 2t} + \cdots + e_{j_k}\beta_k^{\ 2t}$$

求解上述方程组的关键是确定误差位置 β_i，为此，定义差错位置多项式：

$$\sigma(x) = (1+\beta_1)(1+\beta_2)\cdots(1+\beta_k) = \sigma_0 + \sigma_1 x + \cdots + \sigma_k x^k \quad (6\text{-}24)$$

$\sigma(x)$ 的根 β_1^{-1}、β_2^{-1}、…、β_k^{-1} 是差错位置 β_i 的倒数，$\sigma(x)$ 的系数 σ_i 和部分伴随式 S_i 之间的关系如下：

$$S_1 + \sigma_1 = 0$$

$$S_2 + \sigma_1 S_1 + 2\sigma_2 = 0$$

$$S_3 + \sigma_1 S_2 + \sigma_2 S_1 + 3\sigma_3 = 0$$

$$\vdots$$

$$S_k + \sigma_1 S_{k-1} + \sigma_2 S_{k-2} + \cdots + k\sigma_k = 0$$

由上述方程组可以解得差错位置多项式的系数 σ_i，再利用钱搜索法来确定差错的具体位置。

钱搜索法的基本做法是：将 GF(q)中的非零元素元素 α^i 依次代入 $\sigma(x)$，若差错多项式的值为 0，则 α^i 是 $\sigma(x)$ 的根，说明在第 $n-i$ 处出现了差错。

RS 码的译码步骤如下：

(1) 由接收码字 $r(x)$ 求出部分伴随式 S_i 的值，若 S_i 全 0，则输出接收码字 $r(x)$；

(2) 由部分伴随式 S_i 求出 σ_i（ $i=1, 2, \cdots, k$），确定差错位置多项式 $\sigma(x)$；

(3) 通过钱搜索法得到 $\sigma(x)$ 的根，进一步确定差错位置 β_i；

(4) 由部分伴随式 S_i 及差错位置 β_i 求出差错大小；

(5) 由差错位置和差错大小求出误差多项式 $e(x)$，计算

$$\hat{c}(x) = r(x) - e(x)$$

(6) 校验 $\hat{c}(x) \bmod g(x) = 0$ 是否成立，若成立，则输出 $\hat{c}(x)$，否则输出 $r(x)$。

6.2.2 里德－索罗码的 MATLAB 设计

一、Rsenco 函数

功能：里德－索罗编码

语法： **code**=rsenco(**msg**, *N*, *K*);

code=rsenco(**msg**, *N*, *K*, *type_flag*);

code=rsenco(**msg**, *N*,*K*,*type_flag*,**pg**);

[**code**, *added*]=rsenco(…);

说明：**code**=rsenco(**msg**, *N*, *K*)对二进制信息 **msg** 进行信息位为 *K*，码长为 *N* 的编码。一般码长 $n=2^m-1$，m 是大于 2 的整数。输入信息 **msg** 有以下两种形式：

(1) 长度为 *L* 的列矢量，对应的输出长度为 *Lc*1＝ceil(*L*/*m*/*k*)×*m*×*n*；

(2) 大小为 *L*×*m* 的矩阵，对应的输出是大小为 *Lc*2×*m* 的矩阵，其中

*Lc*2＝ceil(*L*/*k*)×*n*。

在 **code**=rsenco(**msg**, *N*, *K*, *type_flag*)中，通过参数 *type_flag* 来定义输入 **msg** 的格式。默认值为'*binary*'；当 *type_flag*＝'*decimal*'时，**msg** 中的元素是 0 到 *N* 的整数；当 *type_flag*＝'*power*'时，**msg** 中的元素以有限域 GF(2^m)中幂的形式表示。对采用'*decimal*'和'*power*'两种格式，其输入信息 **msg** 的格式为：

(1) 长度为 *L* 的列矢量，对应的输出为长度 *Lc*1＝*L*×*n*/*k* 的列矢量；

(2) 大小为 *L*×*k* 的矩阵，输出为 *L*×*n* 的矩阵。

在 **code**=rsenco(**msg**, *N*,*K*,*type_flag*,**pg**)中，通过参数 **pg** 来指定 RS 码的生成多项式。

在[**code**,*added*]=rsenco(…)中，*added* 输出基于 **msg** 格式而添加的列数。

二、Rsdeco 函数

功能：RS 码译码

语法： **msg**=rsdeco(**code**,*N*,*K*);

msg= rsdeco(**code**,*N*,*K*,*type_flag*);

[**msg**,*err*] = rsdeco(...);

[**msg**,*err*,**ccode**] = rsdeco(...);

[**msg**,*err*,**ccode**,**cerr**] = rsdeco(...);

说明：RS码译码函数**msg**=rsdeco(**code**,*N*,*K*)对二进制码字**code**进行译码，恢复出**msg**信息。码长为*N*，信息位为*K*。*N*，*K*的长度要与所选的译码方式一致。其输入信息有一下两种格式：

(1) 长度为*L*的列向量，对应的输出为长度*Lc*1＝*L*×*k*/*n*的列向量；

(2) 大小为*L*×*m*的矩阵，对应的输出为*Lc*2×*m*的矩阵，*Lc*2＝*L*×*k*/*n*。

在 **msg**= rsdeco(**code**,*N*,*K*,*type_flag*)中，通过参数 *type_flag* 定义输入码字的格式。默认值为'*binary*'；当 *type_flag*＝'*decimal*'时，**msg** 中的元素是 0 到 *N* 的整数；当 *type_flag*＝'*power*'时，**msg** 中的元素以有限域 GF(2^m)中幂的形式表示。对采用'*decimal*'和'*power*'两种格式，其输入码字 **code** 的格式为：

(1) 长度为 *L* 的列矢量，对应的输出为长度 *Lc*1＝*L*×*k*/*n* 的列矢量；

(2) 大小为 *L*×*n* 的矩阵，输出为 *L*×*k* 的矩阵。

在[**msg**,*err*] = rsdeco(...)中，*err* 给出错误码字的数目。

在[**msg**,*err*,**ccode**] = rsdeco(...)中，**ccode**给出纠正错误后的码字。

在[**msg**,*err*,**ccode**,**cerr**] = rsdeco(...)中，**cerr** 给出 **ccode** 每列中错误的码字数目。

为了加深对上述函数的理解，下面给出一个完整的 RS 码编码译码程序。

［程序 6-4］RS 码编译码的 MATLAB 程序。

```
function RS()
L=1000;                                          %信息长度
m=4;                                             %设定有限域
N=2^m-1;                                         %计算码长
K＝N－4;                                         %设定信息位长度
msg=randint(L,1);                                %随机产生信息
[code,added]=rsenco(msg,N,K);                    %RS 码编码
noise=rand(length(code)/m,1)<0.03;               %噪声产生
noise=(noise*ones(1,m))';
noise=noise(:);
code_noise=rem(code+noise,2);                    %加上噪声干扰
[dec,err,ccode,cerr]=rsdeco(code_noise,N,K);     %RS 码译码
msg=[msg;zeros(added,1)];
%将信息加上 added 个 1，以便于和译码结果 dec 比较

%显示译码效果
x=[1:length(noise)];
z=[1:m*n:length(noise)];
y=zeros(1:length(z));
z=[z;z];
y=[y+min(cerr);y+max(cerr)];
subplot(211);
plot(x,noise,'yo',x,cerr,'rx',z,y,'g-');         %显示检测到的错误
title('error detection record');
xlable('o--placed errod;x--detected eror');
axis([1,length(noise),min(cerr),max(cerr)]);
```

```
x=[1:length(msg)];
z=[1:m*n:length(msg)];
y=zeros(1:length(z));
z=[z;z];
y=[y;y+max(msg)];
subplot(212);
plot(x,msg,'yo',x,dec,'rx',z,y,'g-');                    %显示译码后的结果
title('message and decoded signal comparison');
xlable('o—original message;x--detected result');
axis([1,length(msg),min(min(msg)),max(max(msg))]);
```

6.2.3 里德－索罗码编/译码的 DSP 实现

[程序6-5]该程序为里德－索罗码的编码程序。

编程说明：程序中采用的RS码的生成多项式为：

$$g(x) = x^4 + \alpha^{13}x^3 + \alpha^6 x^2 + \alpha^3 x + \alpha^{10}$$

编程设计思想为，两个数相乘运算转化为两个数指数形式相加；两个数相除转化为两个数指数形式的相减。这样可以大大的减小程序的运算量，减少内存占用量。

```
;-------------------------------------------------------
;       version:            3.0
;       data:               july-17-1999
;       authors:            ren guo chun
;       comment:              original created
;       cpu type:           ti_tms32054xx
;-------------------------------------------------------

;-------------------------------------------------------
;       compiler:           tms320c54x assembler
;       version:            1.02 (pc)
;       include files:    .mmregs
;-------------------------------------------------------

;-------------------------------------------------------
;       functional description:
;       rs_code
;       for two rs code
;-------------------------------------------------------
```

```
;-------------------------------------------------------
;       activation
;       activation example:
;       call    rs_code
;-------------------------------------------------------

;-------------------------------------------------------
;       External Data
        .ref    CRSBUF
;-------------------------------------------------------

;-------------------------------------------------------
;       temporary data:
;       dp=4
;       cram0
;       cram1
;       cram2
;       cram3
;       cram4
;       cram5
;       cram6
;       cram7
;-------------------------------------------------------

;-------------------------------------------------------
;       input data structure
;       crsbuf:     l=10
;       d00,d01,d02,d03,d04,d05,d06,d07,d08,d09
;       data format:    8-bit data buffer
;-------------------------------------------------------

;-------------------------------------------------------
;       output data structure
;       crsbuf:     l=14
;       d00,d01,d02,d03,d04,d05,d06,d07,d08,d09,s00,s01,s02,s03
;       data format:    8-bit data buffer
;-------------------------------------------------------
```

```
rs_code
        LD      #DODP,DP
        STM     #CRSBUF+10,AR3          ;set pointer to
        RPTZ    A,#3
        STL     A,*AR3+                 ;clr s00,s01,s02,s03

        ST      #rs_h_tab,CRAM2         ;set read h table pointer
        STM     #10-1,AR2               ;set main loop counter
        STM     #CRSBUF,AR1             ;take data address point
rs_code_02
        LD      *AR1,A
        AND     #1111B,A                ;load low 4-bit for first code
        ADD     #rs_dat_exp_tab,A
        READA   CRAM0                   ;first code 4-bit 2 exp
        LD      *AR1+,A
        SFTL    A,-4
        AND     #1111B,A                ;load high 4 bit for second code
        ADD     #rs_dat_exp_tab,A
        READA   CRAM1                   ;second code 4-bit 2 exp
        LD      #10-1,A
        ADD     #CRSBUF+1,A
        STLM    A,AR3                   ;set save sxx pointer
        STM     #3,AR4                  ;set second loop counter
rs_code_01
        LD      CRAM2,A
        READA   CRAM3                   ;read h table
        LD      CRAM1,A
        ADD     CRAM3,A
        ADD     #rs_exp_dat_tab,A
        READA   CRAM4                   ;exp 2 norm
        LD      CRAM4,A
        SFTL    A,4
        STL     A,CRAM4
        LD      CRAM0,A
        ADD     CRAM3,A
        ADD     #rs_exp_dat_tab,A
        READA   CRAM5                   ;exp 2 norm
        LD      CRAM5,A
        OR      CRAM4,A                 ; or
```

```
        XOR     *AR3,A                  ; xor
        STL     A,*AR3+                 ;save s00,s01,s02,s03
        ADDM    #1,CRAM2                ;inc 1
        BANZ    rs_code_01,*AR4-        ;second loop back
        BANZ    rs_code_02,*AR2-        ;main loop back
        RET


rs_h_tab
        .word   0DH,03H,07H,06H
        .word   0BH,0AH,06H,01H
        .word   06H,0DH,03H,05H
        .word   0AH,0EH,0CH,08H
        .word   0DH,0EH,09H,0DH
        .word   03H,0CH,04H,05H
        .word   0AH,0DH,0DH,0BH
        .word   01H,0BH,05H,0BH
        .word   01H,07H,08H,08H
        .word   0DH,06H,03H,0AH

rs_dat_exp_tab
        .word   01EH,003H,002H,006H
        .word   001H,009H,005H,00BH
        .word   000H,00EH,008H,00DH
        .word   004H,007H,00AH,00CH

rs_exp_dat_tab
        .word   08H,04H,02H,01H,0CH,06H,03H,0DH
        .word   0AH,05H,0EH,07H,0FH,0BH,09H,08H
        .word   04H,02H,01H,0CH,06H,03H,0DH,0AH
        .word   05H,0EH,07H,0FH,0BH,09H,00H,00H
        .word   00H,00H,00H,00H,00H,00H,00H,00H
        .word   00H,00H,00H,00H,00H,00H,00H,00H
        .word   00H,00H,00H,00H,00H,00H,00H,00H
        .word   00H,00H,00H,00H,00H,00H,00H,00H
        .end
```

[程序 6-6]该程序为里德－索罗码的译码程序。

```
;-------------------------------------------------------
;       version:            3.0
;       data:               july-17-1999
;       authors:            ren guo chun
;       comment:              original created
;       cpu type:           ti_tms32054xx
;-------------------------------------------------------

;-------------------------------------------------------
;       compiler:           tms320c54x assembler
;       version:            1.02 (pc)
;       include files:    .mmregs
;-------------------------------------------------------

;-------------------------------------------------------
;       functional description:
;       rs_decode
;-------------------------------------------------------

;-------------------------------------------------------
;       activation
;       activation example:
;       call      rs_decode
;-------------------------------------------------------

;-------------------------------------------------------
;       External Data
        .ref      DRSBUF
;-------------------------------------------------------

;-------------------------------------------------------
;       temporary data:
;       dp=4
;       cram0
;       cram1
;       cram2
;       cram3
;       cram4
;       cram5
```

```
;           cram6
;           cram7
;-------------------------------------------------

;-------------------------------------------------
;           input data structure
;           drsbuf:
;           d00,d01,d02,d03,d04,d05,d06,d07,d08,d09,S00,S01,S02,S03
;           data format:      4-bit data buffer
;-------------------------------------------------

;-------------------------------------------------
;           output data structure
;           drsbuf:
;           d00,d01,d02,d03,d04,d05,d06,d07,d08,d09
;           data format:      4-bit data buffer
;-------------------------------------------------


rs_decode
            CALL      rs_dec_sub00          ;for compute s01,s02,s03,s04
            CALL      rs_dec_sub10            ;for compute a1,a2
            CALL      rs_dec_sub20            ;find error address
            CALL      rs_dec_sub30            ;correct error
            RET

*this is rs decod program
rs_dec_sub00
            STM       #DRSBUF,AR1               ;take data point
            STM       #14-2,AR2                 ;set main loop

            LD        *AR1+,A                   ;take one data l 4bit
            AND       #1111B,A
            ADD       #rs_dat_exp_tab,A
            READA     CRAM0

            STM       #SBUF0,AR3                ;set s01-s04    save pointer
            LD        CRAM0,A
            RPT       #3
```

```
        STL     A,*AR3+                 ;exp to s01-s04

rs_dec_sub00_01
        LD      *AR1+,A                 ;take second data
        AND     #1111B,A
        STL     A,CRAM1                 ;load 4-bit

        ST      #1,CRAM2
        STM     #SBUF0,AR3              ;set s01-s04 save pointer
        STM     #3,BRC
        RPTB    rs_dec_sub00_02-1

        LD      *AR3,A                  ;S01-s04
        ADD     CRAM2,A                 ;exp add for norm mpy
        ADD     #rs_exp_dat_tab,A
        READA   CRAM3
        LD      CRAM1,A
        XOR     CRAM3,A
        ADD     #rs_dat_exp_tab,A       ;norm 2 exp
        READA   CRAM3
        LD      CRAM3,A
        STL     A,*AR3+                 ;save S01-s04
        ADDM    #1,CRAM2
rs_dec_sub00_02
        BANZ    rs_dec_sub00_01,*AR2-   ;main loop back

        RET

rs_dec_sub10
        LD      S02,A                   ;compute a1 a2
        ADD     S02,A
        STL     A,CRAM0                 ;s2*s2
        LD      S01,A
        ADD     S03,A
        ADD     #rs_exp_dat_tab,A
        READA   CRAM1                   ;S1*S3
        LD      CRAM0,A
        ADD     #rs_exp_dat_tab,A
        READA   CRAM0                   ;exp to norm data
```

```
        LD      CRAM0,A
        XOR     CRAM1,A
        STL     A,CRAM1
        ADD     #rs_dat_exp_tab,A
        READA   CRAM0                   ;S1*S3-S2*S2
        LD      CRAM1,A
        BC      rs_dec_sub10_02,ANEQ
        LD      S02,A                   ;1 error
        SUB     S01,A
        ADD     #rs_exp_dat_tab+0FH,A
        READA   CRAM0
        LD      CRAM0,A
        ADD     #rs_dat_exp_tab,A
        READA   S03
        ST      #1EH,S04                ; 1/a
        B       rs_dec_sub10_05
rs_dec_sub10_02
        LD      S01,A                   ;>1 error
        ADD     S04,A
        ADD     #rs_exp_dat_tab,A
        READA   CRAM1                   ;S1*S4
        LD      S02,A
        ADD     S03,A
        ADD     #rs_exp_dat_tab,A
        READA   CRAM3
        LD      CRAM3,A
        XOR     CRAM1,A                 ;S2*S3
        ADD     #rs_dat_exp_tab,A
        READA   CRAM1                   ;S1*S4-S2*S3
        LD      S03,A
        ADD     S03,A
        ADD     #rs_exp_dat_tab,A
        READA   CRAM2                   ;S3*S3
        LD      S02,A
        ADD     S04,A                   ;S2*S4
        ADD     #rs_exp_dat_tab,A
        READA   CRAM3
        LD      CRAM3,A
        XOR     CRAM2,A
```

```
        ADD     #rs_dat_exp_tab,A
        READA   CRAM2                   ;S3*S3-S2*S4

        LD      CRAM1,A
        SUB     CRAM0,A
        ADD     #rs_exp_dat_tab+0FH,A
        READA   CRAM1                   ;exp 2 norm
        LD      CRAM1,A
        ADD     #rs_dat_exp_tab,A
        READA   S03
        LD      CRAM2,A
        SUB     CRAM0,A
        ADD     #rs_exp_dat_tab+0FH,A
        READA   CRAM2                   ;exp 2 norm
        LD      CRAM2,A
        ADD     #rs_dat_exp_tab,A
        READA   S04                     ;norm 2 exp
rs_dec_sub10_05
        RET

rs_dec_sub20
        LD      #1EH,A
        STL     A,CRAM4                 ;ini error address 1
        STL     A,CRAM5                 ;ini error address 2
        LD      S03,A
        STL     A,CRAM0
        LD      S04,A
        STL     A,CRAM1
        STM     #14,AR1
rs_dec_sub20_01
        LD      CRAM0,A
        ADD     #1,A
        STL     A,CRAM0
        ADD     #rs_exp_dat_tab,A
        READA   CRAM2                   ;exp 2 norm
        LD      CRAM1,A
        ADD     #rs_exp_dat_tab+2,A
        READA   CRAM3                   ;exp 2 norm
        LD      CRAM3,A
```

```
        ADD     #rs_dat_exp_tab,A
        READA   CRAM1                   ;norm 2 exp
        LD      CRAM3,A
        XOR     CRAM2,A
        SUB     #8,A
        BC      rs_dec_sub20_02,ANEQ
        MVKD    AR1,CRAM2
        LD      CRAM2,A
        SUB     SEEZ0,A
        BC      rs_dec_sub20_03,AGEQ    ;check if can not decode
        LD      CRAM4,A                 ;mov data
        STL     A,CRAM5
        LD      CRAM2,A                 ;save error address
        STL     A,CRAM4
rs_dec_sub20_02
        BANZ    rs_dec_sub20_01,*AR1-
rs_dec_sub20_03
        RET

rs_dec_sub30
        NOP                             ;correct error
        LD      CRAM5,A
        SUB     #1EH,A
        BC      rs_dec_sub30_01,ANEQ    ;jmp to for 2 error
        LD      CRAM4,A
        SUB     #1EH,A
        BC      rs_dec_sub30_02,AEQ     ;jmp 0 error
                                        ;or can not correct
        LD      S01,A                   ;for    one error
        ADD     #0FH,A
        SUB     CRAM4,A
        ADD     #rs_exp_dat_tab,A
        READA   CRAM0                   ;exp 2 norm
        LD      CRAM0,A
        BC      rs_dec_sub30_02,AEQ
        LD      SEEZ0,A
        SUB     CRAM4,A                 ;error address
        ADD     #DRSBUF-1,A
        STLM    A,AR1                   ;ready correct address
```

```
        NOP
        NOP
        LD      *AR1,A
        XOR     CRAM0,A                 ;correct one error
        STL     A,*AR1
        B       rs_dec_sub30_03

rs_dec_sub30_01
        LD      CRAM4,1,A
        ADD     #rs_exp_dat_tab,A       ;exp 2 norm
        READA   CRAM0
        LD      CRAM0,A
        ADD     #rs_dat_exp_tab,A       ;norm 2 exp
        READA   CRAM0                   ;x1>2
        LD      CRAM5,1,A
        ADD     #rs_exp_dat_tab,A       ;exp 2 norm
        READA   CRAM1
        LD      CRAM1,A
        ADD     #rs_dat_exp_tab,A       ;norm 2 exp
        READA   CRAM1                   ;x2>2

        LD      CRAM0,A
        ADD     CRAM5,A
        ADD     #rs_exp_dat_tab,A       ;exp 2 norm
        READA   CRAM2
        LD      CRAM1,A
        ADD     CRAM4,A
        ADD     #rs_exp_dat_tab,A       ;exp 2 norm
        READA   CRAM3
        LD      CRAM3,A
        XOR     CRAM2,A
        ADD     #rs_dat_exp_tab,A       ;norm 2 exp
        READA   CRAM2                   ;/a

        LD      S01,A
        ADD     CRAM1,A
        ADD     #rs_exp_dat_tab,A       ;exp 2 norm
        READA   CRAM1
        LD      S02,A
```

```
ADD       CRAM5,A
ADD       #rs_exp_dat_tab,A        ;exp 2 norm
READA     CRAM6
LD        CRAM6,A
XOR       CRAM1,A
ADD       #rs_dat_exp_tab,A        ;norm 2 exp
READA     CRAM1
LD        CRAM1,A
ADD       #0FH,A
SUB       CRAM2,A
ADD       #rs_exp_dat_tab,A        ;exp 2 norm
READA     CRAM1                        ;y1 first error data

LD        S01,A
ADD       CRAM0,A
ADD       #rs_exp_dat_tab,A        ;exp 2 norm
READA     CRAM0
LD        S02,A
ADD       CRAM4,A
ADD       #rs_exp_dat_tab,A        ;exp 2 norm
READA     CRAM6
LD        CRAM6,A
XOR       CRAM0,A
ADD       #rs_dat_exp_tab,A        ;norm 2 exp
READA     CRAM0
LD        CRAM0,A
ADD       #0FH,A
SUB       CRAM2,A
ADD       #rs_exp_dat_tab,A        ;exp 2 norm
READA     CRAM0                        ;y2 second error data

LD        SEEZ0,A
SUB       CRAM4,A                  ;error address
ADD       #DRSBUF-1,A
STLM      A,AR1                    ;ready correct error address
NOP
NOP
LD        *AR1,A
XOR       CRAM1,A                  ;correct first error
```

```
        STL     A,*AR1
        LD      SEEZ0,A
        SUB     CRAM5,A                 ;error address
        ADD     #DRSBUF-1,A
        STLM    A,AR1                   ;ready correct error address
        NOP
        NOP
        LD      *AR1,A                  ;correct second error
        XOR     CRAM0,A
        STL     A,*AR1
rs_dec_sub30_03
rs_dec_sub30_02
        RET

;decode program end

rs_h_tab
        .word   0DH,03H,07H,06H
        .word   0BH,0AH,06H,01H
        .word   06H,0DH,03H,05H
        .word   0AH,0EH,0CH,08H
        .word   0DH,0EH,09H,0DH
        .word   03H,0CH,04H,05H
        .word   0AH,0DH,0DH,0BH
        .word   01H,0BH,05H,0BH
        .word   01H,07H,08H,08H
        .word   0DH,06H,03H,0AH

rs_dat_exp_tab
        .word   01EH,003H,002H,006H
        .word   001H,009H,005H,00BH
        .word   000H,00EH,008H,00DH
        .word   004H,007H,00AH,00CH

rs_exp_dat_tab
        .word   08H,04H,02H,01H,0CH,06H,03H,0DH
        .word   0AH,05H,0EH,07H,0FH,0BH,09H,08H
        .word   04H,02H,01H,0CH,06H,03H,0DH,0AH
        .word   05H,0EH,07H,0FH,0BH,09H,00H,00H
```

```
.word   00H,00H,00H,00H,00H,00H,00H,00H
.word   00H,00H,00H,00H,00H,00H,00H,00H
.word   00H,00H,00H,00H,00H,00H,00H,00H
.word   00H,00H,00H,00H,00H,00H,00H,00H
.end
```

6.3 卷积码的 DSP 实现

卷积码和分组码的根本区别在于，它不是把信息序列分组后再进行单独地编码，而是由连续输入的信息序列得到连续输出的已编码序列。采用卷积码将 k 个信息码元编为 n 个码元时，这 n 个码元不仅与当前段的 k 个信息有关，而且与前面的 m 段信息有关，称 $N=m+1$ 为编码约束度，表示编码过程中相互约束的码段的个数。常把卷积码记为 (n,k,m)，或 (nN,kN)，码率 $R_c=k/n$。卷积码的纠错能力随 N 的增加而增大。在编码器复杂性相同的情况下，卷积码可以比分组码获得更高的编码增益。

6.3.1 卷积码的基本原理和设计方法

一、卷积码编码

卷积码编码器的一般结构包括两部分：一个由 N 段组成的输入移位寄存器，每段有 k 级，共 Nk 位寄存器；n 个模 2 和相加器，其输入分别对应于 n 个基于生成多项式的线性代数方程。每输入 k 个比特，编码器输出 n 个比特。

为了对卷积码作进一步的说明，图 6-3 给出了一个（2,1,2）二进制卷积码的编码器。若输入编码器一个新的信息元 k_i，存储器内的数据右移一位，k_i 与前三个输入的信息 k_{i-1}、k_{i-2}、k_{i-3} 按图中线路所确定的规则进行运算，得到两个输出码元 n_{i1}、n_{i2}，共同组成子码 $c_i=(n_{i1},n_{i2})$。

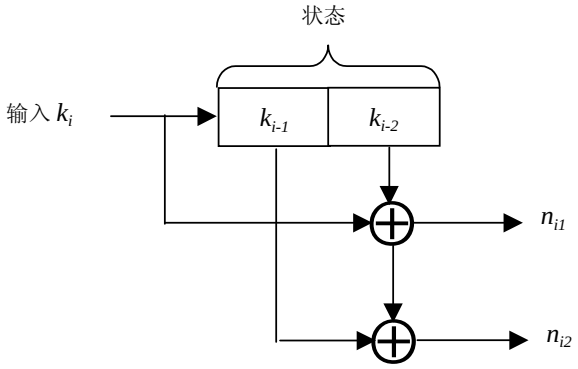


图 6-3 （2,1,2）二进制卷积码的编码器

卷积码的编码过程中，编码器从全 0 状态出发，最后必须回到全 0 状态，因此当送完信息后，还要向编码器再送入 m 段全 0 信息，以迫使编码器回到全 0 状态。

在卷积码的译码过程中，不仅要根据此时刻输入到译码器的子码，而且还要根据以后 m_d 段收到的子码才能译出一个子码信息元，通常取 $m_d \geq m$，称 $N_d = m_d + 1$ 为译码约束度，表示译码过程中互相约束的码段的个数。

描述卷积码编码过程的方法很多，如：矩阵法、多项式法、码树和网格图等，这里我们主要介绍和卷积码编码器结构密切相关的多项式法，以及与卷积码译码密切相关的网格图法。由卷积码的生成多项式可以直接得到编码器的结构图。

如上面例子中的(2,1,2)卷积码的生成多项式矩阵为：

$$G(D) = [1 + D^2, 1 + D + D^2] \tag{6-25}$$

也可以写成二进制形式或八进制形式

gen=[101, 111]

gen=[5, 7]

其中，D 是延迟算子，生成多项式的第一项为 $1+D^2$，表示输出编码的第一个码元等于输入码元 k_i 与前输入码元 k_{i-2} 的模 2 和，其第二项为 $1+D+D^2$，表示输出编码的第二个码元为输入码元和前二个输入码元的 k_{i-1}、 k_{i-2} 三项的模 2 和。

为了更好的理解卷积码生成多项式矩阵和编码器之间的关系，下面我们再举一个 k>1 的例子，（3,2,2）卷积码的其生成多项式矩阵为：

$$\mathbf{G}(D) = \begin{bmatrix} 1 & 0 & 1+D^2 \\ 0 & 1 & 1+D \end{bmatrix} \tag{6-26}$$

对应的卷积码编码器如图 6-4 所示：

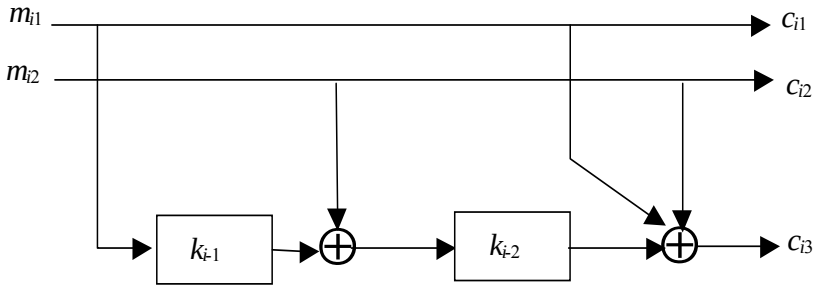


图 6-4 （3,2,2）卷积码编码器

其中，$c_{i3} = m_{i1} \oplus m_{i2} \oplus m_{(i-2)1} \oplus m_{(i-1)2}$，这与生成多项式矩阵 $\mathbf{G}(D)$ 中第三列的两项相吻合。

用来表示卷积码的另一种常用方法是网格图，即时间与对应状态的转移图。考虑如图 6-2 所示的（2,1,2）卷积码，该码具有 4 个状态 00、01、10、11，分别对应于移位寄存器两个单元的内容，在每个时间周期内，用 4 点代替这 4 个状态，然后根据状态之间可能发

生的转移情况将这些点连接起来，就可以得到网格图。该码的网格图如图 6-5 所示。

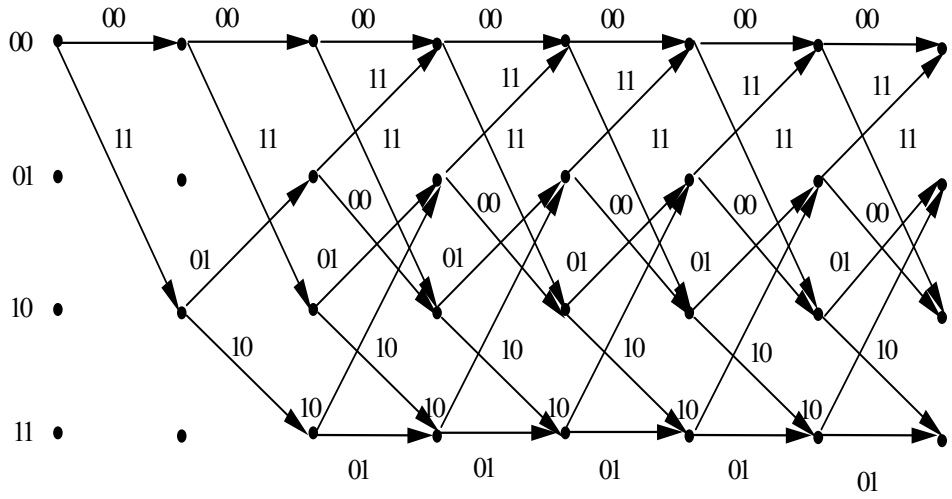


图 6-5 （2,1,2）卷积码的网格图

图 6-5 中在连接两个状态的每一条支路上，用两个二进制符号表示与转移相对应的编码器输出，卷积码的码字与网格上相应的路径相对应，开始于全 0 状态且返回全 0 状态。

二、卷积码的维特比译码

卷积码的译码方法主要包括：维特比算法、费诺算法和堆栈算法等等，其中最重要的是维特比算法。卷积码译码时的判决可以采用硬判决，也可以采用软判决，软判决的特性比硬判决要好 2～3dB。

维特比算法是一种最大似然算法，它不是在网格图上依次比较所有的可能路径，而是接收一段，计算、比较一段，保留最有可能的路径，从而达到整个码序列是一个最大似然序列。

维特比算法可以描述如下：

把在阶段 i，状态 S_j 所对应的网格图节点记作 $S_{j,i}$，给每个网格图节点赋值 $V(S_{j,i})$。节点值按照如下步骤计算。

(1)将接收序列分为 L 个长度为 n_0 的码段，画出长度为 $L+m$ 段的网格图；

(2)设 $V(S_{0,0})=0$，$i=1$；

(3)在 $i=1$ 阶段，计算进入每一状态的分支的部分路径长度，挑选并存储最小部分路径长度的路径以及长度 $V(S_{j,1})$，称此部分路径为幸存路径；

(4)i 增加 1，把此阶段进入每一状态的所有分支长度，和同这一分支相连的前一阶段的幸存路径的长度 $V(S_{j,i})$相加，挑选最小的加以存储并删去其它所有路径，得到该阶段的幸存路径和长度 $V(S_{j,i+1})$，使幸存路径的长度增加了一个分支；

(5)若 $j<L$，则返回第 4 步，否则跳入第 6 步；

(6)从 $L+1$ 阶段的全 0 状态开始，通过网格图中的幸存路径返回到初始的全 0 状态，该路径即为最大似然路径，对应于它的输入比特序列是最大似然解码的信息序列。

（n_0，k_0，m）卷积码编码器共有 $2^{k_0 m}$ 个状态，因此维特比译码器必需具有同样的 $2^{k_0 m}$ 个状态，并且在译码过程中要存储各状态的幸存路径以及长度，维特比译码器的复杂程度随 $2^{k_0 m}$ 指数增加，一般要求 $m \leqslant 10$。同样的，当 L 很大时也会造成译码器的存储量太大而难以实现，可以采用截尾译码来解决这个问题，只要存储 $\tau << L$ 段子码即可。当译码器接收并处理完 τ 个码段后，译码器中的路径存储器已经全部存满，当处理第 $\tau+1$ 个码段时，必须对路径存储器中的第一段信息作出判决并输出。截尾译码可以大大降低译码的复杂性，但其性能可能稍差，如果 τ 选择足够大，则对译码错误概率的影响很小。一般的 $\tau = (5 \sim 10)m$ 。

为了更好的理解维特比译码，下面给出一个具体的译码过程。

假设采用硬判决译码，量化接收序列为 y=10 11 01 11 00 01 00, 采用的卷积码如图 6-3 所示，维特比解码的网格图如图 6-6 所示，其中实线表示输入 0，虚线表示输入 1，每一条支路上的两个二进制符号表示对应的编码器输出。

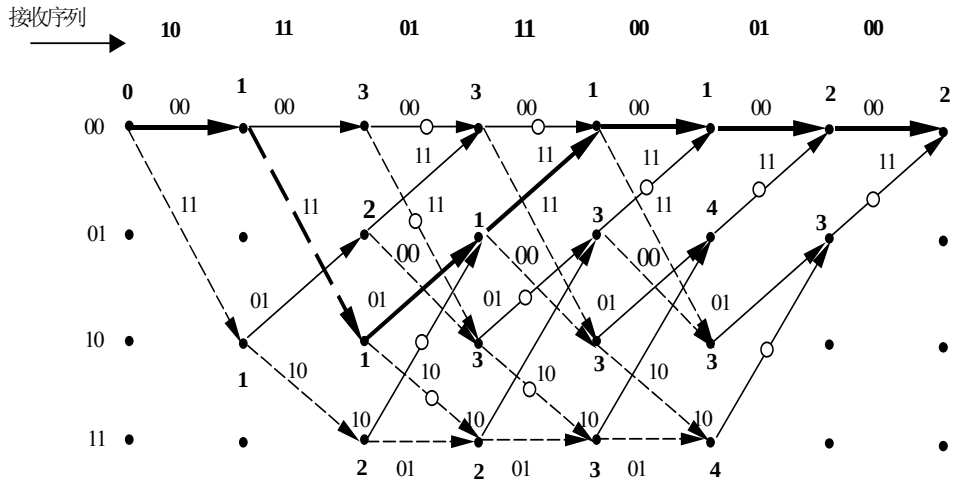


图 6-6　维特比译码示意图

前面我们已经提出，最后 m 段信息一定是 0，所以在图 6-5 中的最后 2 段只画出了输入 0 的情况。全 0 初始状态的长度设置为 0，并计算到下一个阶段的部分路径长度，由于只存在一条路径，因此不需要删除路径。而到了第 3 阶段，出现了进入同一状态的不同路径，则需要选出幸存路径，这里，我们将删除的路径用圈标出，若两条路径的部分路径长度相同，则不进行路径删除，可以任选一条作为幸存路径。最后，从全 0 状态沿幸存路径返回初始状态，就可以完成译码过程。在图 6-6 中，用粗线表示的即为最后的幸存路径，由此得到的译码结果为 01000。

6.3.2 卷积码的 MATLAB 实现

一、convenco 函数

功能：卷积码编码函数

语法：**code**＝convenco(**msg**, **tran_func**)

说明：**code**＝convenco(**msg**, **tran_func**)的功能是对 K 位信息进行卷积编码。**trans_func** 是采用的（N, K, M）卷积码的转移函数矩阵，可由 sim2tran 函数得到。输入信息 **msg** 矩阵的列数为 K，输出码字的列数为 N，行数为 **msg** 矩阵的行数加上记忆长度 M。

二、sim2gen 函数

功能：将 simulink 的流程框图转化为卷积码生成多项式矩阵（八进制形式）

语法：**trans_func**＝sim2gen（*sim_file*）

说明：*sim_file* 为 simulink 流程框图，由寄存器、异或单元（逻辑加）、输入单元（信息输入）和输出单元（编码输出）等组成。sim2gen 函数的功能是从卷积码的结构图中抽象出八进制形式的生成多项式矩阵。生成多项式矩阵给出了卷积编码输入信息和输出码字间的关系，其结构为：

$$\mathbf{gen}=\begin{bmatrix} g_{11} & g_{12} & \cdots & g_{1N} \\ g_{21} & g_{22} & \cdots & g_{2N} \\ \vdots & \vdots & & \vdots \\ g_{K1} & g_{K2} & \cdots & g_{KN} \end{bmatrix} \tag{6-27}$$

其中，g_{ij} 为第 i 个输入到第 j 个输出的传递函数。例如，当 g_{ij}＝32 时，生成多项式矩阵中对应项为 $\mathbf{g}(D)=1+D+D^3$。由该函数得到的生成多项式矩阵可用于 convenco 函数来对信息进行卷积编码和译码。

三、sim2tran 函数

功能：将 simulink 的流程框图转化为卷积码的传递函数

语法：**tran_func**=sim2tran(*sim_file*)

说明：*sim_file* 为 simulink 流程框图。输出的传递函数可表示为：

$$\mathbf{tran\_func}=\begin{bmatrix} \mathbf{A} & \mathbf{B} & & \mathbf{V} \\ \mathbf{C} & \mathbf{D} & & \end{bmatrix} \tag{6-28}$$

$$\mathbf{tran\_func}=\begin{bmatrix} a_{11} & a_{12} & \cdots & a_{1M} & b_{11} & b_{12} & \cdots & b_{1K} & N \\ & & \cdots & & & & \cdots & & K \\ a_{M1} & a_{M2} & \cdots & a_{MM} & b_{M1} & b_{M2} & \cdots & b_{MK} & M \\ c_{11} & c_{12} & \cdots & c_{1M} & d_{11} & d_{12} & \cdots & d_{1K} & 0 \\ & & \cdots & & & & \cdots & & \\ c_{N1} & c_{N2} & \cdots & c_{NM} & d_{N1} & d_{N2} & \cdots & d_{NK} & -Inf \end{bmatrix}$$

卷积码的编码过程可以用下列方程来表示：

$$x(k+1)=\mathbf{A}x(k)+\mathbf{B}\mu(k) \tag{6-29}$$

$$y(k)=\mathbf{C}x(k)+\mathbf{D}\mu(k) \tag{6-30}$$

其中，$\mu(k)$ 是输入矢量，$x(k)$ 为状态，$y(k)$ 为输出矢量。由该函数得到的传递函数同样可用于 convenco 函数来对对信息进行卷积编码和译码。

6.3.3 维特比译码的 MATLAB 实现

viterbi 函数

功能：维特比译码

语法：**msg**=viterbi(**code**,**tran_func**)

msg=viterbi(**code**,**tran_func**, *memory_length*)

msg=viterbi(**code**,**tran_func**, *memory_length*,**tran_prob**)

[**msg**,**metric**]=viterbi(…)

说明：维特比译码函数是基于最大似然译码基础上的译码方法对接收到的卷积码进行译码。该函数还可以生成网格图来帮助分析译码过程。

在 **msg**=viterbi(**code**,**tran_func**)中，用和编码方式相对应的 **tran_func** 来对接收到的码字 **code** 进行译码。**code** 是一个列数为 N 的矩阵，**msg** 是译码后的信息，列数为 K，行数为 **code** 的行数－(M－1)。

在 **msg**=viterbi(**code**,**tran_func**, *memory_length*)中，*memory_length* 表示在译码过程中的最大存储长度。一般的，当 *memory_length* 大于 $5\times N\times M$ 时，就可以获得比较精确的译码结果。

当信道为离散无记忆信道时，可以使用 **tran_prob** 来表征信道的转移特性，并使用 **msg**=viterbi(**code**,**tran_func**, *memory_length*,**tran_prob**)来进行译码。图 6-7 所示的概率转移图可以用式(6-31)来表示：

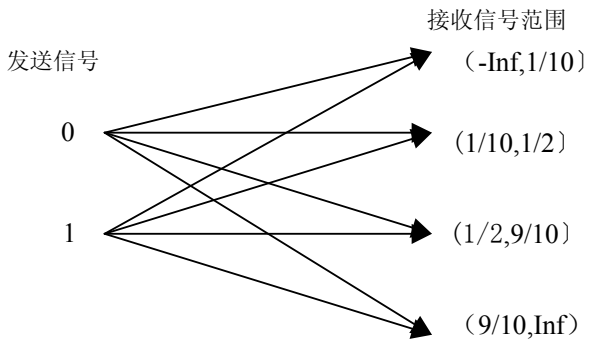


图 6-7　概率转移图

$$\mathbf{tran\_prob} = \begin{bmatrix} -Inf & 1/10 & 1/2 & 9/10 \\ 2/5 & 1/3 & 1/5 & 1/15 \\ 1/6 & 1/8 & 3/8 & 7/16 \end{bmatrix} \tag{6-31}$$

tran_prob 矩阵的第一行给定了接收范围，第二行给出了信息“0”在接收端落在各接收范围内的概率，第三行给出了信息“1”在接收端落在各接收范围内的概率。在进行软判决时需要给出 **tran_prob** 矩阵。

[**msg**, **metric**]=viterbi(…)可以返回每步计算出的汉明路径。

为了加深对上述函数的理解，我们给出一个（3,2,2）卷积码的 MATLAB 实现过程。其 simulink 的流程框图如图 6-8 所示，其文件名为 sim1.mdl：

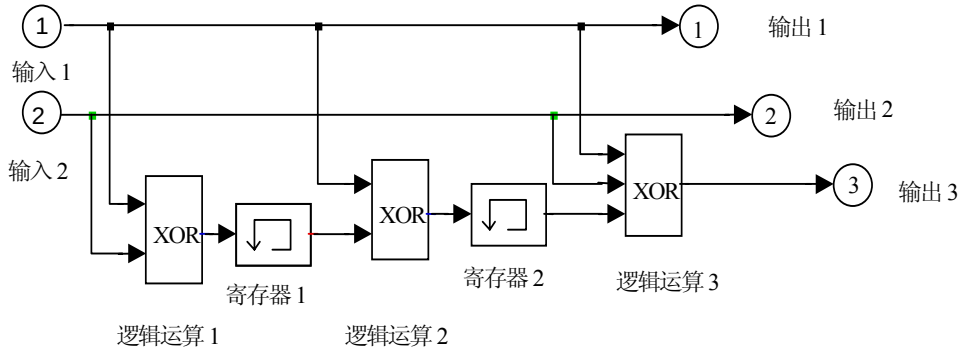


图 6-8 （3,2,2）卷积码 simulink 流程框图

［程序 6-7］卷积码编译码 MATLAB 程序。

```
tran_func=sim2tran(sim1);                    %由流程框图得到传递函数
msg=randint(200,2,2);                        %信息
code=convenco(msg,tran_func);                %卷积码编码
k1=size(code,1);
k2=size(code,2);
code_noise=rem(code+rand(k1,k2,1)>0.95,2);    %加噪声
rcv=viterbi(code_noise,tran_func);           %卷积码译码
```

6.3.4 卷积码的 DSP 实现

[程序 6-8]该程序完成（2,1,7）卷积码的编码，其编码器结构如图 6-9 所示。

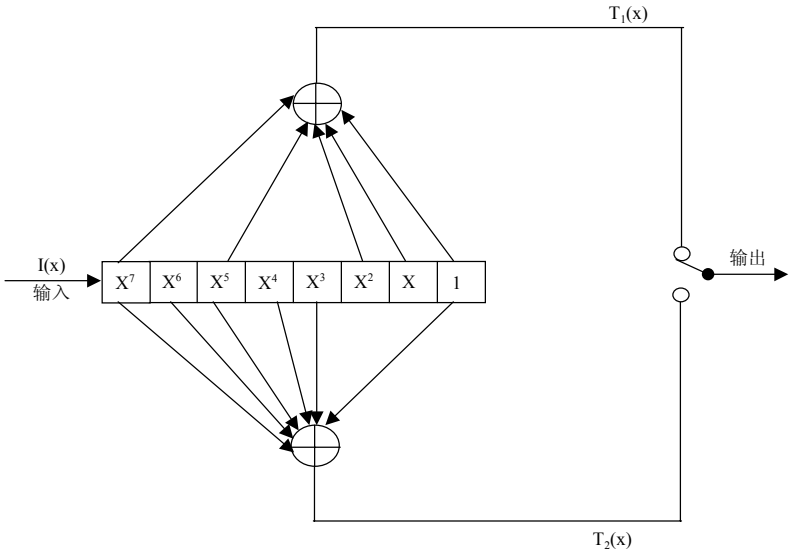


图 6-9　(2,1,7)卷积码编码器结构图

```
;-------------------------------------------------------
;           version:            1.0
;           data:               july-11-1999
;           authors:            ren guo chun
;           comment:              original created
;           cpu type:           ti_tms32054xx
;-------------------------------------------------------


;-------------------------------------------------------
;           compiler:           tms320c54x assembler
;           version:            1.02 (pc)
;           include files:    .mmregs
;-------------------------------------------------------


;-------------------------------------------------------
;           functional description:
;           con_code
;

;             +---------+-------------+----+----+---->t1(x)
```

```
;            |             |                   |     |     |
;            |             |                   |     |     |
;           ---   ---   ---   ---   ---   ---   ---   ---
;    -->--|D|--|D|--|D|--|D|--|D|--|D|--|D|--|D|--|D|->
;           ---   ---   ---   ---   ---   ---   ---   ---
;            |     |     |     |     |                   |
;            |     |     |     |     |                   |
;            +----+----+----+----+--------------+--->t2(x)
;
;           code(2,1,7)
;-------------------------------------------------------

;-------------------------------------------------------
;           activation
;           activation example:
;           call       con_code
;-------------------------------------------------------

;-------------------------------------------------------
;           External Data
            .ref       CON_ROR
;-------------------------------------------------------

;-------------------------------------------------------
;           temporary data:
;           dp=4
;           cram0
;           cram1
;-------------------------------------------------------

;-------------------------------------------------------
;           in     data:
;           CON_ROR
;           data format:       16-bit data buffer
;-------------------------------------------------------

;-------------------------------------------------------
;           output data:
;           cram0                ;t1(x)
```

```
;           cram1                ;t2(x)
;           data format:       1-bit data
;-------------------------------------------------


;this is con_code prog
con_code
            CALL      con_one                ;t1(x)
            CALL      con_two                ;t2(x)
            RET

con_one
            STM       #0,AR0                  ;clear counter
            BITF      CON_ROR,BIT_00          ;check bit 00
            NOP
            NOP
            XC        1,TC                   ;test tc
            MAR       *AR0+                   ;inc ar0

            BITF      CON_ROR,BIT_02          ;check bit 02
            NOP
            NOP
            XC        1,TC                   ;test tc
            MAR       *AR0+                   ;inc ar0

            BITF      CON_ROR,BIT_05          ;check bit 05
            NOP
            NOP
            XC        1,TC                   ;test tc
            MAR       *AR0+                   ;inc ar0

            BITF      CON_ROR,BIT_06          ;check bit 06
            NOP
            NOP
            XC        1,TC                   ;test tc
            MAR       *AR0+                   ;inc ar0

            BITF      CON_ROR,BIT_07          ;check bit 07
            NOP
```

```
        NOP
        XC      1,TC                    ;test tc
        MAR     *AR0+                   ;inc ar0

        MVKD    AR0,CRAM1               ;code 1
        ANDM    #1B,CRAM1
        RET

con_two
        STM     #0,AR0
        BITF    CON_ROR,BIT_00          ;check bit 00
        NOP
        NOP
        XC      1,TC                    ;test tc
        MAR     *AR0+                   ;inc ar0

        BITF    CON_ROR,BIT_01          ;check bit 01
        NOP
        NOP
        XC      1,TC                    ;test tc
        MAR     *AR0+                   ;inc ar0

        BITF    CON_ROR,BIT_02          ;check bit 02
        NOP
        NOP
        XC      1,TC                    ;test tc
        MAR     *AR0+                   ;inc ar0

        BITF    CON_ROR,BIT_03          ;check bit 03
        NOP
        NOP
        XC      1,TC                    ;test tc
        MAR     *AR0+                   ;inc ar0

        BITF    CON_ROR,BIT_04          ;check bit 04
        NOP
        NOP
        XC      1,TC                    ;test tc
        MAR     *AR0+                   ;inc ar0
```

```
        BITF    CON_ROR,BIT_07          ;check bit 07
        NOP
        NOP
        XC      1,TC                    ;test tc
        ADD     *AR0+,A                 ;inc ar0

        MVKD    AR0,CRAM0               ;code 2
        ANDM    #1B,CRAM0
        RET

        .end
```

[程序 6-9]该程序完成（2,1,7）卷积码的 Viterbi 译码。
编程说明:本程序采用软判决 Viterbi 译码算法，角度量化量为 128。

```
;---------------------------------------------------
;       version:        1.0
;       data:           july-11-1999
;       authors:        ren guo chun
;       comment:          original created
;       cpu type:       ti_tms32054xx
;---------------------------------------------------

;---------------------------------------------------
;       compiler:       tms320c54x assembler
;       version:        1.02 (pc)
        include files:  .mmregs
;---------------------------------------------------

;---------------------------------------------------
;       functional description:
;       con_decode

;
;        +---------+-------------+----+----+---->t1(x)
;        |         |             |    |    |
;        |         |             |    |    |
;       ---  ---  ---  ---  ---  ---  ---  ---
;  -->--|D|--|D|--|D|--|D|--|D|--|D|--|D|--|D|->
```

```
;           ---   ---   ---   ---   ---   ---   ---   ---
;            |     |     |     |     |                   |
;            |     |     |     |     |                   |
;            +----+----+----+----+-------------+---->t2(x)
;
;           code(2,1,7)
;-------------------------------------------------------

;-------------------------------------------------------
;           activation
;           activation example:
;           call      con_decode
;           call      ini_dec
;-------------------------------------------------------

;-------------------------------------------------------
;           External Data
            .ref      CRAM0
            .ref      CRAM1
;-------------------------------------------------------

;-------------------------------------------------------
;           temporary data:
;           dp=4
;           cram0
;           cram1
;           cram2
;           cram3
;           cram4
;           cram5
;           cram6
;           cram7
;-------------------------------------------------------

;-------------------------------------------------------
;           data structure
; TRE_BUF .set      01080H
; WAT_B0   .set      TRE_BUF                  ;l=vl
; WAT_B1   .set      WAT_B0+TRE_VL             ;l=vl
```

```
; DAT_B0    .set       WAT_B1+TRE_VL              ;l=vl*4
; DAT_B1    .set       DAT_B0+TRE_VL*4            ;l=vl*4
; WAT_ADR .set         DAT_B1+TRE_VL*4            ;l=vl
; WAT_RDB .set         WAT_ADR+TRE_VL             ;l=6
; WAT_RD0 .set         WAT_RDB
; WAT_RD1 .set         WAT_RDB+1
; WAT_RD2 .set         WAT_RDB+3
; WAT_RD3 .set         WAT_RDB+4
;-------------------------------------------------------

;-------------------------------------------------------
;               constant data
; TRE_VL    .set       128
;
;-------------------------------------------------------

;-------------------------------------------------------
;             in      data:
;             CRAM0
;             CRAM1
;             data format:       8-bit data
;-------------------------------------------------------

;-------------------------------------------------------
;             output data:
;             DAT_B0       buffer
;             data format:       16-bit data
;-------------------------------------------------------

;this is con_decode prog
con_decode

              STM        #WAT_RDB,AR1                ;use cram0 cram1

              LD         CRAM0,A                     ;for 00b
              ADD        CRAM1,A
              STL        A,*AR1+                     ;save weight for 00b
              MAR        *AR1+
              STL        A,*AR1-                     ;save weight for 00b
```

```
        LD      #100H,A                 ;for 11b
        SUB     CRAM0,A
        SUB     CRAM1,A
        STL     A,*AR1+                 ;save weight for 11b
        MAR     *AR1+

        LD      #80H,A                  ;for 01b
        ADD     CRAM1,A
        SUB     CRAM0,A
        STL     A,*AR1+                 ;save weight for 01b
        MAR     *AR1+
        STL     A,*AR1-                 ;save weight for 01b

        LD      #80H,A                  ;for 10b
        SUB     CRAM1,A
        ADD     CRAM0,A
        STL     A,*AR1+                 ;save weight for 10b

hop_tree_m01
        XORM    #BIT_09,BIT_FLG         ;mod 2 counter
        BITF    BIT_FLG,BIT_09
        BC      tree_bb_m00,NTC         ;check if go to bb_state

tree_aa_m00                             ;aa_state
        STM     #WAT_B0,AR1             ;AR1:weight rd
      ;a_buf pointer 1
        STM     #WAT_B0+TRE_VL/2,AR2    ;AR2:weight rd
                                        ;a_buf pointer 2
        STM     #WAT_B1,AR3             ;AR3:weight wr
   ;b_buf pointer
        STM     #DAT_B0,AR4             ;AR4:data rd a_buf pointer 1
        STM     #DAT_B0+TRE_VL*2,AR5    ;AR5:data rd a_buf pointer 2
        STM     #DAT_B1+3,AR6           ;AR6:data wr b_buf pointer
        STM     #TRE_VL/2-1,BRC         ;repeat tre_vl/2 times
        STM     #04H,AR0                ;AR0=04
        ST      #tre_tab,CRAM2          ;set read table pointer
        RPTB    tre_da_m05-1
        LD      CRAM2,A
```

```
        READA   CRAM0                   ;read table
        MVDK    CRAM0,AR7               ;read weight address
        ADDM    #1,CRAM2                ;inc read pointer
        NOP
        LD      *AR7+,A
        ADD     *AR1,A
        STL     A,*AR3                  ;save weight to b_buf
        LD      *AR7+,A
        ADD     *AR2,A
        SUB     *AR3,A
        BC      tre_da_m01,AGT          ;check if change data
        ADD     *AR3,A
        STL     A,*AR3+                 ;save weight
        LD      *AR5+,16,B              ;load pointer 2 data
        ADDS    *AR5+,B
        LD      *AR5+,16,A
        ADDS    *AR5+,A                 ;64-bit data
        MAR     *AR5-0
        B       tre_da_m02
tre_da_m01
        MAR     *AR3+
        LD      *AR4+,16,B              ;load pointer 1 data
        ADDS    *AR4+,B
        LD      *AR4+,16,A
        ADDS    *AR4+,A
        MAR     *AR4-0                  ;64-bit data
tre_da_m02
        RSBX    C                       ;C=0
        ROR     A                       ;ror A
        ROR     B                       ;ror b
        STL     A,*AR6-                 ;save A
        STH     A,*AR6-
        STL     B,*AR6-                 ;save B
        STH     B,*AR6-
        MAR     *AR6+0
        MAR     *AR6+0

        LD      CRAM2,A
        READA   CRAM0                   ;read table
```

```
        MVDK    CRAM0,AR7               ;read weight address
        ADDM    #1,CRAM2                ;inc read pointer
        NOP
        LD      *AR7+,A
        ADD     *AR1+,A
        STL     A,*AR3                  ;wave weight to b_buf
        LD      *AR7+,A
        ADD     *AR2+,A
        SUB     *AR3,A
        BC      tre_da_m03,AGT          ;check if change data
        ADD     *AR3,A
        STL     A,*AR3+
        LD      *AR5+,16,B              ;load pointer 2 data
        ADDS    *AR5+,B
        LD      *AR5+,16,A
        ADDS    *AR5+,A
        MAR     *AR4+0
        B       tre_da_m04
tre_da_m03
        MAR     *AR3+                   ;load pointer 1 data
        LD      *AR4+,16,B
        ADDS    *AR4+,B
        LD      *AR4+,16,A
        ADDS    *AR4+,A
        MAR     *AR5+0
tre_da_m04
        SSBX    C                       ;C=1
        ROR     A                       ;ror A
        ROR     B                       ;ror b
        STL     A,*AR6-                 ;save A
        STH     A,*AR6-
        STL     B,*AR6-                 ;save B
        STH     B,*AR6-
        MAR     *AR6+0
        MAR     *AR6+0

tre_da_m05
        B       tre_end
```

```
tree_bb_m00
        STM     #WAT_B1,AR1             ;AR1:weight rd
                                        ;b_buf pointer 1
        STM       #WAT_B1+TRE_VL/2,AR2  ;AR2:weight rd
                                        ;b_buf pointer 2
        STM     #WAT_B0,AR3             ;AR3:weight wr
                                        ;a_buf pointer
        STM     #DAT_B1,AR4             ;AR4:data rd
                                        ;b_buf pointer 1
        STM     #DAT_B1+TRE_VL*2,AR5    ;AR5:data rd
                                        ;b_buf pointer 2
        STM     #DAT_B0+3,AR6           ;AR6:data wr a_buf pointer
        STM     #TRE_VL/2-1,BRC         ;repeat tre_vl/2 times
        STM     #04H,AR0                ;AR0=04
        ST      #tre_tab,CRAM2          ;set read table pointer
        RPTB    tre_db_m05-1
        LD      CRAM2,A
        READA   CRAM0                   ;read table
        MVDK    CRAM0,AR7               ;read weight address
        ADDM    #1,CRAM2                ;inc read pointer
        NOP
        LD      *AR7+,A
        ADD     *AR1,A
        STL     A,*AR3                  ;save weight to b_buf
        LD      *AR7+,A
        ADD     *AR2,A
        SUB     *AR3,A
        BC      tre_db_m01,AGT          ;check if change data
        ADD     *AR3,A
        STL     A,*AR3+                 ;save weight
        LD      *AR5+,16,B              ;load pointer 2 data
        ADDS    *AR5+,B
        LD      *AR5+,16,A
        ADDS    *AR5+,A
        MAR     *AR5-0                  ;64-bit data
        B       tre_db_m02
tre_db_m01
        MAR     *AR3+
        LD      *AR4+,16,B              ;load pointer 1 data
```

```
        ADDS    *AR4+,B
        LD      *AR4+,16,A
        ADDS    *AR4+,A                 ;64-bit data
        MAR     *AR4-0
tre_db_m02
        RSBX    C                       ;C=0
        ROR     A                       ;ror A
        ROR     B                       ;ror b
        STL     A,*AR6-                 ;save A
        STH     A,*AR6-
        STL     B,*AR6-                 ;save B
        STH     B,*AR6-
        MAR     *AR6+0
        MAR     *AR6+0

        LD      CRAM2,A                 ;read table
        READA   CRAM0                   ;read weight address
        MVDK    CRAM0,AR7               ;inc read pointer
        ADDM    #1,CRAM2
        NOP
        LD      *AR7+,A
        ADD     *AR1+,A
        STL     A,*AR3                  ;save weight to b_buf
        LD      *AR7+,A
        ADD     *AR2+,A
        SUB     *AR3,A
        BC      tre_db_m03,AGT          ;check if change data
        ADD     *AR3,A
        STL     A,*AR3+                 ;save weight
        LD      *AR5+,16,B              ;load pointer 2 data
        ADDS    *AR5+,B
        LD      *AR5+,16,A
        ADDS    *AR5+,A
        MAR     *AR4+0                  ;64-bit data
        B       tre_db_m04
tre_db_m03
        MAR     *AR3+
        LD      *AR4+,16,B              ;load pointer 1 data
        ADDS    *AR4+,B
```

```
        LD      *AR4+,16,A
        ADDS    *AR4+,A                 ;64-bit data
        MAR     *AR5+0
tre_db_m04
        SSBX    C                       ;C=1
        ROR     A                       ;ror A
        ROR     B                       ;ror b
        STL     A,*AR6-                 ;save A
        STH     A,*AR6-
        STL     B,*AR6-                 ;save B
        STH     B,*AR6-
        MAR     *AR6+0
        MAR     *AR6+0

tre_db_m05
tre_end
        NOP
        RET

ini_dec
                                        ;ini weight_adr buffer
        STM     #WAT_ADR,AR1
        RPT     #TRE_VL-1
        MVPD    tre_tab,*AR1+           ;move data from
                                        ;prog 2 dada ram
        STM     #WAT_B0,AR1
        RPTZ    A,#TRE_VL*10-1          ;clr weight buffer data
        STL     A,*AR1+

        STM     #WAT_B0+1,AR1
        LD      #0FFH,A
        RPT     #TRE_VL-2
        STL     A,*AR1+                 ;ini weight buffer data=0ffh
        RET

tre_tab
        .word   WAT_RD0,WAT_RD1,WAT_RD2,WAT_RD3
        .word   WAT_RD1,WAT_RD0,WAT_RD3,WAT_RD2
        .word   WAT_RD2,WAT_RD3,WAT_RD0,WAT_RD1
```

```
.word   WAT_RD3,WAT_RD2,WAT_RD1,WAT_RD0
.word   WAT_RD2,WAT_RD3,WAT_RD0,WAT_RD1
.word   WAT_RD3,WAT_RD2,WAT_RD1,WAT_RD0
.word   WAT_RD0,WAT_RD1,WAT_RD2,WAT_RD3
.word   WAT_RD1,WAT_RD0,WAT_RD3,WAT_RD2

.word   WAT_RD3,WAT_RD2,WAT_RD1,WAT_RD0
.word   WAT_RD2,WAT_RD3,WAT_RD0,WAT_RD1
.word   WAT_RD1,WAT_RD0,WAT_RD3,WAT_RD2
.word   WAT_RD0,WAT_RD1,WAT_RD2,WAT_RD3
.word   WAT_RD1,WAT_RD0,WAT_RD3,WAT_RD2
.word   WAT_RD0,WAT_RD1,WAT_RD2,WAT_RD3
.word   WAT_RD3,WAT_RD2,WAT_RD1,WAT_RD0
.word   WAT_RD2,WAT_RD3,WAT_RD0,WAT_RD1

.word   WAT_RD3,WAT_RD2,WAT_RD1,WAT_RD0
.word   WAT_RD2,WAT_RD3,WAT_RD0,WAT_RD1
.word   WAT_RD1,WAT_RD0,WAT_RD3,WAT_RD2
.word   WAT_RD0,WAT_RD1,WAT_RD2,WAT_RD3
.word   WAT_RD1,WAT_RD0,WAT_RD3,WAT_RD2
.word   WAT_RD0,WAT_RD1,WAT_RD2,WAT_RD3
.word   WAT_RD3,WAT_RD2,WAT_RD1,WAT_RD0
.word   WAT_RD2,WAT_RD3,WAT_RD0,WAT_RD1

.word   WAT_RD0,WAT_RD1,WAT_RD2,WAT_RD3
.word   WAT_RD1,WAT_RD0,WAT_RD3,WAT_RD2
.word   WAT_RD2,WAT_RD3,WAT_RD0,WAT_RD1
.word   WAT_RD3,WAT_RD2,WAT_RD1,WAT_RD0
.word   WAT_RD2,WAT_RD3,WAT_RD0,WAT_RD1
.word   WAT_RD3,WAT_RD2,WAT_RD1,WAT_RD0
.word   WAT_RD0,WAT_RD1,WAT_RD2,WAT_RD3
.word   WAT_RD1,WAT_RD0,WAT_RD3,WAT_RD2
.end
```

第七章 均衡器的 DSP 实现

7.1 概述

理想低通和等效理想低通滤波器都能满足奈氏第一准则，即在抽样时刻没有码间串扰。但是由于信道特性的变化和设计制造的误差，一个实际的基带传输系统不可能完全满足理想的无码间串扰的条件，因而串扰几乎是不可避免的。为了减小这种串扰，可以采用串接一个滤波器进行校正，这个滤波器通常称为均衡器。校正可以从频域和时域两个不同的角度考虑。在频域校正称频域均衡，它是通过调整均衡器把信道和均衡器总的频谱特性校正为理想低通或等效低通特性，从而实现无码间串扰传输。若从时域考虑问题，它是以奈氏第一准则为依据，通过调整抽头系数，从时间波形上把畸变了的波形校正为在取样点上无码间串扰，我们把这种均衡称为时域均衡。随着数字信号处理理论和超大规模集成电路的发展，时域均衡已成为当今高速数据传输中所使用的主要方法。

调整滤波器抽头系数的方法有手动调整和自动调整等。如果接收端知道信道特性，包括信道冲激响应或频率响应，一般采用比较简单的手动调整方式。由于无线通信信道具有随机性和时变性，即信道特性事先却是未知的，信道响应是时变的。这就要求均衡器必须能够实时地跟踪无线通信信道的时变特性，均衡器必须可以根据信道响应自动调整抽头系数。我们称这种可以自动调整滤波器抽头系数的均衡器为自适应均衡器。

自适应均衡器一般包含两种工作模式，即训练模式和跟踪模式。在训练模式中，发射机发射一个已知的、定长的训练序列，以便接收机处的均衡器可以作出正确的设置。典型的训练序列是一个二进制伪随机信号或是一串预先指定的数据位，而紧跟在训练序列之后被传送的是用户数据。接收机处的均衡器将通过递归算法来评估信道特性，并且修正均衡器系数以对信道作出补偿。在设计训练序列时，要求做到即使在最差的信道条件下，均衡器也能通过这个序列获得正确的滤波器系数。这样就可以在接收训练序列后，使得均衡器的滤波器系数已经接近于最佳值。而在接收用户数据时，均衡器的自适应算法就可以跟踪不断变化的信道。其结果就是，自适应均衡器将不断改变其滤波特性。

均衡器从调整参数至形成收敛，整个过程的时间跨度是均衡器算法、结构和信道变化率的函数。为了保证能有效地消除码间干扰，均衡器需要周期性地做重复训练。均衡器被大量用于数字通信系统中，因为在数字通信系统中用户数据是被分成若干段并被放在相应的时间段中传送的。时分多址（TDMA）无线通信系统特别适合于使用均衡器。这是因为 TDMA 系统在长度固定的时间段中传送数据，且训练序列通常在时间段的头部被发送。每当收到新的时间段，均衡器将用同样的训练序列进行修正。

在现实中，平台的费用、功耗消耗，以及无线传播特性支配着均衡器的结构及其算法的选择。在便携式无线电话的应用中，当需要让用户的通话时长尽量加长时，用户单元的电池使用时间是最关键的。只有当均衡器所带来的链路性能的改进能抵消费用和功耗所带

来的负面影响时，均衡器才会得到应用。由于自适应均衡器是对未知的时变信道作出补偿，因而它需要有特别的算法来更新均衡器的系数，以跟踪信道的变化。关于滤波器系数的算法有很多，对自适应算法的详细研究是一项很复杂的工作。

早期的自适应滤波器的研究是针对自适应天线系统和数字传输系统的均衡器来设计的。本章所讨论的自适应均衡器结构是 FIR 类型均衡器结构，而逼近算法采用的是迫零算法、LMS 算法和 RLS 算法。这些结构和算法是自适应技术的基础。

绝大多数的均衡器是基于最新发展的 DSP 来设计的。基于 DSP 的自适应均衡器比用硬件实现的自适应均衡器有很多优点，其功率消耗以及体积更小、更容易实现。特别重要的是修改程序使系统很容易升级，功能可进一步完善。一个自适应均衡器实现的复杂性通常是用它所需的乘法次数和阶数来衡量的。基于 DSP 实现的自适应均衡器，其 DSP 的数据吞吐量和数据处理能力也是重要因素。大多数 DSP 都有并行的硬件乘法器、流水结构以及快速的片内存储器，这些资源使自适应均衡器的实现更容易且更有效。

本章将描述自适应均衡器的基本原理及其 DSP 实现中的一些实际问题，介绍两种自适应 FIR 型滤波器结构，然后介绍三种常用自适应算法，即迫零算法、LMS 算法和 RLS 算法，最后，以横向滤波型 LMS 自适应均衡器以例，说明其实现及编程要点，并对实际应用中实现这些均衡器所涉及的问题进行讨论。

7.2 无线通信中的自适应均衡技术

7.2.1 无线通信中的均衡器

在无线通信系统中，均衡器常被放在无线接收机的基带或中频部分实现。因为基带包络的复数表达式可以描述带通信号波形，所以信道响应、解调信号和自适应均衡器的算法通常都可以在基带部分被仿真和实现。

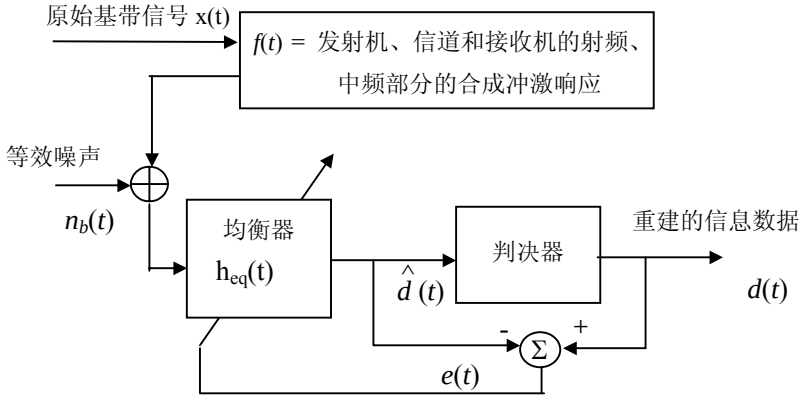


图 7-1　使用均衡器的通信系统的结构框图

图 7-1 是通信系统的结构框图，其接收机中包含有自适应均衡器。如果 $x(t)$ 是原始信息信号，$f(t)$ 是等效的基带冲激响应，即综合反映了发射机、信道和接收机的射频、中频部分的总的传输特性，那么均衡器收到的信号可以表示成：

$$y(t) = x(t) \otimes f^*(t) + n_b(t) \tag{7-1}$$

其中，$f^*(t)$ 是 $f(t)$ 的复共轭函数，$n_b(t)$ 是均衡器输入端的基带噪声，$\otimes$ 为卷积操作符。如果均衡器的冲激响应是 $h_{eq}(t)$，则均衡器的输出为：

$$\begin{aligned} \hat{d}(t) &= x(t) \otimes f^*(t) \otimes h_{eq}(t) + n_b(t) \otimes h_{eq}(t) \\ &= x(t) \otimes g(t) + n_b(t) \otimes h_{eq}(t) \end{aligned} \tag{7-2}$$

其中，$g(t)$ 是发射机、信道接收机的射频、中频部分和均衡器四者的等效冲激响应。横向滤波均衡器的基带复数冲激响应可以描述如下：

$$h_{eq}(t) = \sum_n c_n \delta(t - nT) \tag{7-3}$$

其中，c_n 是均衡器的复数滤波系数。均衡器的期望输出值为原始信息 $x(t)$。假定 $n_b(t)$=0，那么为 了使公式(7-2)中的 $\hat{d}(t) = x(t)$，必须要求

$$g(t) = f^*(t) \otimes h_{eq}(t) = \delta(t) \tag{7-4}$$

均衡器的目的就是实现公式(7-4)，其频域表达式为

$$H_{eq}(f)F^*(-f) = 1 \tag{7-5}$$

其中 $H_{eq}(f)$ 和 $F(f)$ 是 $h_{eq}(t)$ 和 $f(t)$ 所对应的傅里叶变换。

公式(7-5)表明均衡器实际上是传输信道的反向滤波器。如果传输信道是频率选择性的，那么均衡器将增强频率衰落大的频谱部分，而削弱频率衰落小的频谱部分，以使所收到的频谱的各部分衰落趋于平坦，相位趋于线性。对于时变信道，自适应均衡器可以跟踪信道的变化，以使公式(7-5)基本满足。

7.2.2 均衡技术分类

均衡技术可被分为两类：线性均衡和非线性均衡。这两类的判别主要在于自适应均衡器的输出被用于反馈控制的方法。通常，模拟信号经过接收机中的判决器，然后由判决器

进行限幅或阈值操作，并决定信号的数字逻辑值 *d(t)*（参见图 7-1）。如果 *d(t)*被应用于反馈逻辑中并帮助改变了均衡器的后续输出，那么均衡器是非线性的。实现均衡的滤波器结构有许多种，而且每种结构在实现时又有许多种算法。图 7-2 是按均衡器所用的类型、结构和算法的不同，对常用的均衡技术进行分类的结果。本章后续各节主要讲述不同结构及算法的均衡器的基本原理及其 DSP 实现。

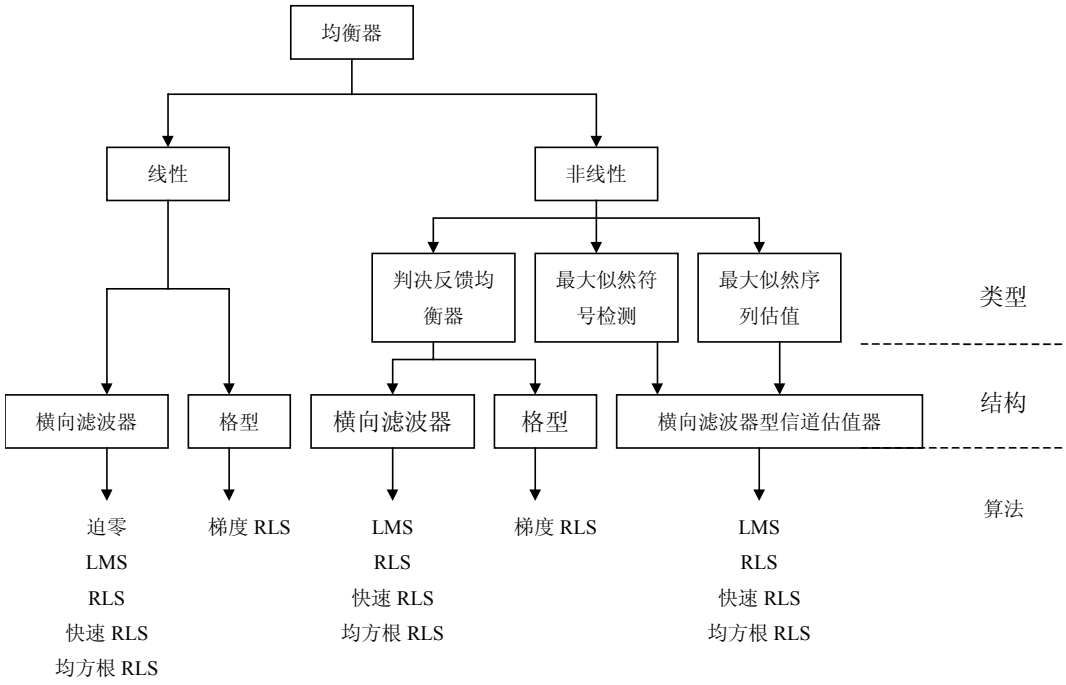


图 7-2 均衡器类型、结构和算法

7.3 自适应均衡器的基本原理

自适应均衡器的结构可以是 IIR 型结构，也可以是 FIR 型结构。但在实际应用中，一般都采用 FIR 型。其主要原因是 FIR 结构的自适应技术实现更容易，其权系数的修正就调节了均衡器的性能，同时还可以保证其稳定性。对于 IIR 型均衡器则存在不稳定性问题，当进行自适应处理过程中出现极点移出单位圆之外时，会使均衡器产生不稳定。因此，本章所讨论的自适应均衡器均采用 FIR 型结构。

7.3.1 自适应均衡器的结构体系

正如上节所述，自适应均衡器的结构可以是横向结构以及格形结构。下面我们介绍这两种结构形式的自适应均衡器结构。

一、横向型结构

横向型结构是在大多数应用情况下所采用的最主要的自适应均衡器结构，这种滤波器在可用的类型中是最简单的，它把所收到信号的当前值和过去值按滤波系数（即权重）作线性迭加，并把生成的和作为输出，如图 7-3 所示。均衡器的输出 $y(n)$ 表示为

$$y(n)=\mathbf{W}^T(n)\mathbf{X}(n)=\sum_{i=0}^{N-1}w_i(n)x(n-i) \tag{7-6}$$

其中 $\mathbf{X}(n)=\begin{bmatrix}x(n) & x(n-1) & \cdots & x(n-N-1)\end{bmatrix}^T$ 为输入矢量，$\mathbf{W}(n)=\begin{bmatrix}w(n) & w(n-1) & \cdots & w(n-N-1)\end{bmatrix}^T$ 是权系数矢量，T 为转置符，n 为时间序列，N 为均衡器的阶数。由式（7-6）可见 $y(n)$ 实际是两矢量的内积——即把 $\mathbf{X}(n)$ 和 $\mathbf{W}(n)$ 相卷积的结果，用 C 语言可编程如下：

```
y[n]=0;
for(i=0;i<N;i++)
{y[n]+=wn[i]*xn[i];
    }
```

程序中 wn[i]表示 $w_i(n)$，而 xn[i]代表 $x(n-i)$。

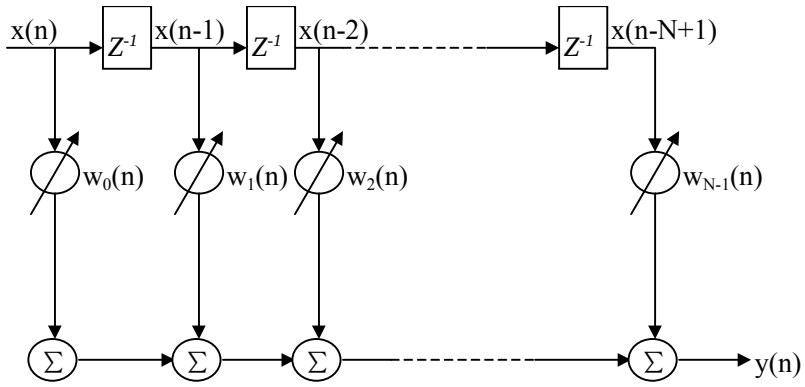


图 7-3 横向型滤波器结构

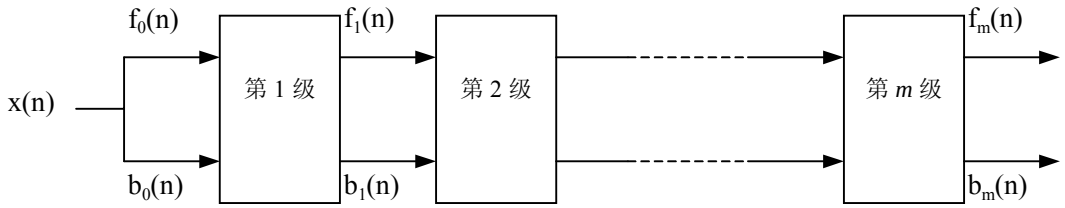
二、格形结构的自适应均衡器

另一种迭代 FIR 均衡器结构是格形结构，格形结构的引出是用 Durbin 算法求解自适应均衡器的最佳权系数所导出来的。在使用 LMS 算法求解自适应均衡器最佳权系数（详细介绍见 7.2.3.1）时，可以发现，最佳权系数满足一个线性方程组，该方程组的系数矩阵具有 Toeplitz 性质，即该矩阵为对称矩阵，且方阵的任一子方阵的对角线上的元是相等的。具有 Toeplitz 性质的线性方程组可按下列的思路求解：先假定导出了只有 N-1 元的方程组的解，并由此推导出 N 元方程组的解。这是一个递推算法，将它应用于最佳权系数估值就形成了 Durbin 算法。这里不详尽推导 Durbin 算法，直接给出格式结构，如图 7-5 所示。

它是基于一系列预测误差滤波器的去相关传输结构。格形结构的误差预测器的递推公式表示如下：

$$\begin{aligned} f_0(n) &= b_0(n) = x(n) \\ f_m(n) &= f_{m-1}(n) - k_m(n) b_{m-1}(n-1) \\ b_m(n) &= b_{m-1}(n-1) - k_m(n) f_{m-1}(n) \end{aligned} \qquad (m=1,2,\cdots\cdots M) \quad (7\text{-}7)$$

其中，$f_m(n)$ 代表前向预测误差；$b_m(n)$ 代表后向预测误差；$k_m(n)$ 代表反向系数，m 为阶数序列值，M 为串联的总级数。格形结构的优点是按阶递归，故增加或者减少级数不会影响存在的阶数设计。



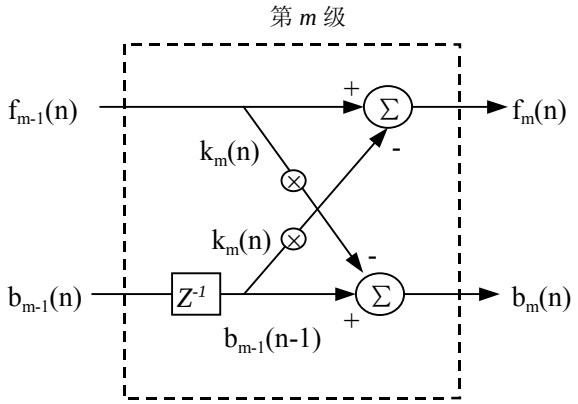


图 7-5 格形结构

要用格形滤波器进行实际数据处理，首先要知道 $k_m(n)$，这些系数可以使用 Durbin 算法由自相关系数的估值来算出，其运算量很大。可以使用下面的反向系数的递推方程来递推估计：

$$k_m(n+1) = k_m(n) + u[f_m(n) b_{m-1}(n-1) + b_m(n) f_{m-1}(n)] \qquad 0 < m \le M \quad (7\text{-}8)$$

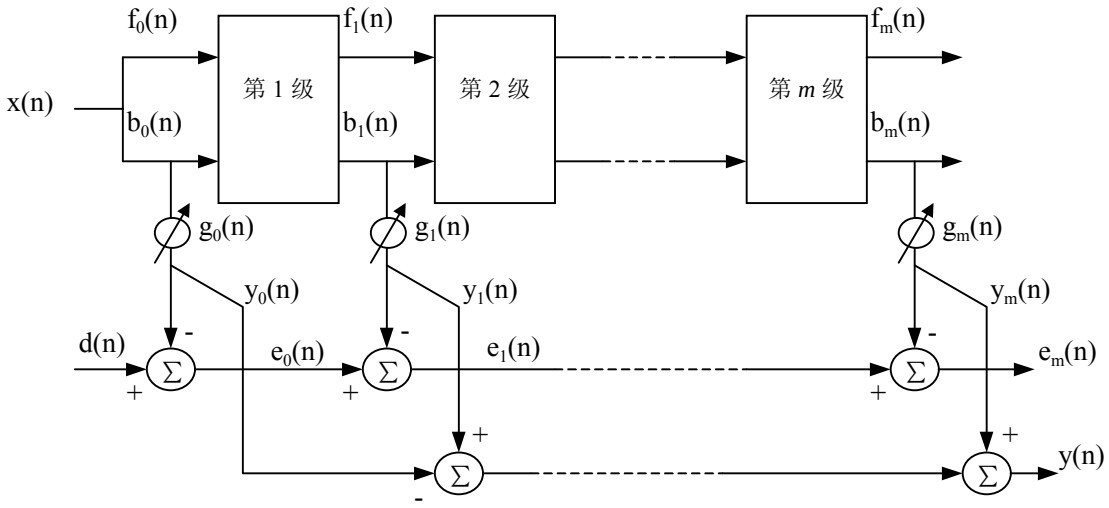


图 7-6 格形结构均衡器结构

在信道均衡器应用中采用格形结构自适应滤波器的结构框图如图 7-6 所示。这个图是通用形式，有两个输出，一个输出为 $e_m(n)$，等效于自适应系统中 $e(n)$ 输出；另一个输出为 $y(n)$，等效为自适应滤波器本身的输出。显然这种格形抽头结构完成两类最佳估计：一类是格形误差预测器，输出分别为前向预测误差 $f_m(n)$ 和后向预测误差 $b_m(n)$；一类是多路回归滤波器，其特性用下列方程表示

$$\begin{aligned} e_0(n) &= d(n) - b_0(n)g_0(n) \\ e_m(n) &= e_{m-1}(n) - b_{m-1}(n)g_{m-1}(n) \\ y(n) &= \sum_{m=0}^{M} g_m(n)b_m(n) \end{aligned} \qquad (7\text{-}9)$$

从上面算法可以看出格形抽头自适应滤波器的计算要复杂得多，但计算量的增加却带来很多好处。格形均衡器有两大优点：即数值稳定性好和收敛速度更快，而且格型均衡器的特殊结构允许进行最有效长度的动态调整。因而，当信道的时间扩散性不很明显时，可以只用少量级数实现；而当信道的时间扩散特性增强时，均衡器的级数可以由算法自动增加，并且不用暂停均衡器的操作。另外格形结构自适应滤波器还提供了误差预测值和可用作其它分析的有明显物理意义的反射系数最佳值。正是这些优点使格形结构得到了广泛应用。

7.3.2 自适应均衡器算法

本章的概述已指出，自适应均衡器除包括一个按照某种结构设计的滤波器，还有一套

自适应算法。自适应算法是根据某种判据来设计的。最常用自适应均衡器算法有迫零算法、LMS 算法和 RLS 算法及它们的各种改进算法。

一、迫零算法

我们知道，横向滤波器结构的自适应均衡器特性完全取决于各抽头系数，而抽头系数的确定则依据均衡的效果。为此，首先要建立度量均衡效果的标准。通常采用的度量标准为峰值畸变或均方畸变。峰值畸变的定义是

$$D=\frac{1}{y_0}\sum_{\substack{k=-\infty\\k\neq 0}}^{\infty}|y_k| \qquad (7\text{-}10)$$

其物理意义是冲激响应的所有抽样时刻码间串扰绝对值之和与 $k=0$ 时刻抽样值之比。码间串扰绝对值之和事实上反映了实际信息传输中某抽样时刻所受前后码元干扰的最大可能值，即峰值。显然，对于无码间串扰的冲激响应来说，$D=0$。以峰值畸变为准则时，选择抽头系数的原则应当是使均衡后的冲激响应的 D 最小。

理论分析证明，如果均衡前的峰值畸变小于 1（即眼图不闭合），则均衡后的最小峰值畸变必定发生在使 $y_k=0(k=\pm1,\pm2,\cdots\pm N)$ 的情形。这里，认为横向滤波器的抽头数为 $2N+1$ 个。为了确定迫使码间串扰 y_k 为零时的抽头系数，需要解 $2N+1$ 个联立方程。自动求解联立方程的最准确和最便于实现的方法是迭代法。数学推导表明，峰值畸变 D 是抽头系数的凸函数，因此调整抽头系数的任何迭代技术都能使峰值畸变达到最小。基于迫使码间串扰为零的均衡算法称为迫零算法。

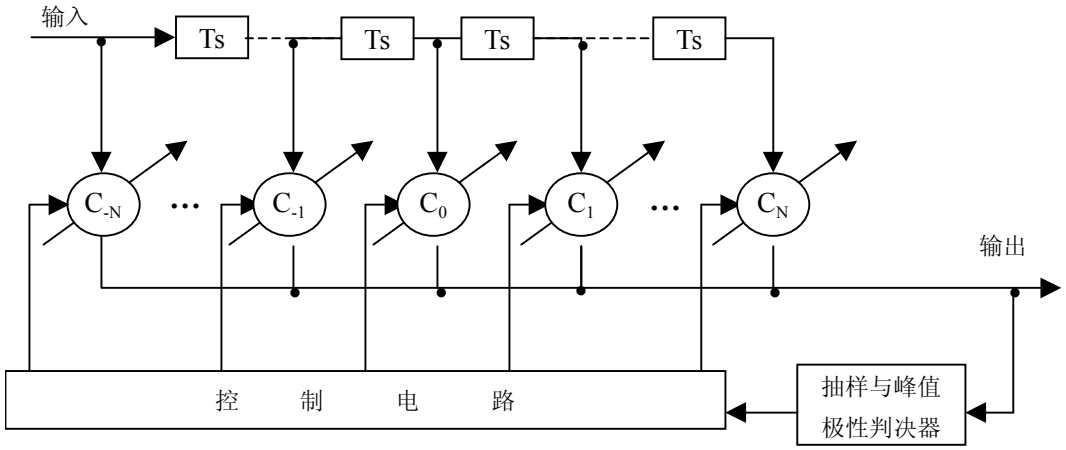


图 7-7 预置式自动均衡

迫零算法的具体实现方案可以有多种。一种最简单的方法是预置式自动均衡，原理方框图如图 7-7 所示。在预置式自动均衡器中，在传输实际信息前先发送低重复频率的单脉冲信号，按照均衡输出冲激响应中各抽样值的正、负极性（y_0除外），以反极性方向在原抽头第数上加一个固定的增量，即增减抽头系数一次。y_k 为正极性时，相应的 C_k 减小一

个增量，反之亦然。为了快速调整，通常对$2N+1$个抽头系数同时进行调整。控制电路的作用就是根据每次对码间串扰的判决结果，对抽头系数的增、减进行控制。可以预计，这种迭代法所能达到的均衡精度与增量大小有关，增量愈小精度愈高，但收敛时间就愈长。

在实际系统中，有时不允许在传输信息之前先进行预置式调整，即使允许预置式调整也不能保证信道在传输期间一成不变。为了在传输信息期间，利用包含在信号中的码间串扰信息自动调整抽头系数，就必须采用自适应均衡。自适应均衡时，不能直接采用各抽样值的极性作为控制信息，而必须从抽样值中提取误差信息（即偏离正常幅度取值的部分），用统计的方法确定误差的正、负极性，然后控制抽头系数的调整方向。不难理解，在采用迫零算法的自适应均衡中，如果初始眼图是闭合的，则误差信息就会发生错误，从而抽头系数调整也会产生错误，这可能导致均衡过程不收敛。

图 7-8 为三个抽头的自适应均衡器的原理方框图。图中并-串转换为$\log_2^K$位，K 为发送信号的电平数，而 A/D 变换器则为$n=\log_2^K+1$位。A/D 变换器的第一位输出码表示抽样值的极性，第 n 位则反映了误差信息。信号极性经移位寄存器与误差的极性用模 2 和求相关后送入可逆计数器进行统计计数器溢出或退尽时控制抽头系数的增、减。图中所示为四电平的系统。

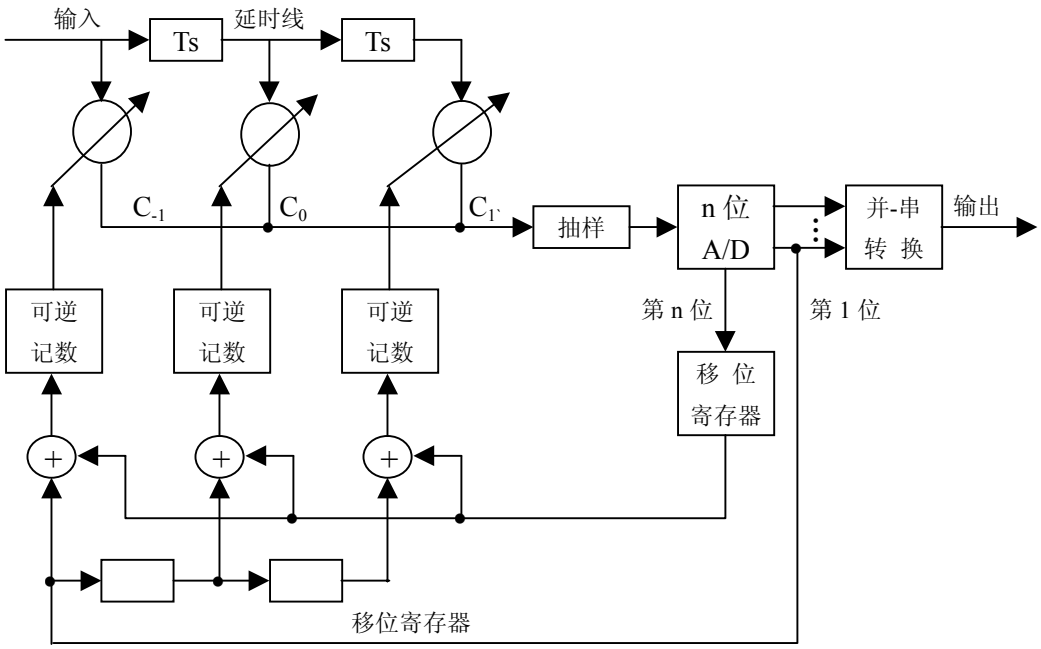


图 7-8　迫零算法自适应均衡器

二、基本 LMS 算法

LMS 算法的判据是最小均方误差，即理想信号 *d(n)*与滤波器输出 *y(n)*之差 *e(n)*的平方值的期望值最小，并且根据这个判据来修改权系数$w_i(n)$，由此产生的算法称为最小均方算法（LMS）。绝大多数对自适应滤波器的研究是基于由 Windrow 提出的 LMS 算法。这是

因为 LMS 算法的设计和实现都较为简单，因而在很多应用场合都非常适用。

令 N阶 FIR 滤波器的抽头系数为 $w_i(n)$，滤波器的输入和输出分别为 $x(n)$和 $y(n)$，则 FIR 横向滤波器方程可表示为

$$y(n)=\sum_{i=1}^{N} w_i(n)x(n-i) \tag{7-11}$$

令 $d(n)$ 代表“所期望的响应”，并定义误差信号

$$\begin{aligned} e(n) &= d(n)-y(n) \\ &= d(n)-\sum_{i=1}^{N} w_i(n)x(n-i) \end{aligned} \tag{7-12}$$

采用向量形式表示权系数及输入 $\mathbf{W}$ 和 $\mathbf{X}(n)$，可以将误差信号 $e(n)$ 写作

$$\begin{aligned} e(n) &= d(n)-\mathbf{W}^T\mathbf{X}(n) \\ &= d(n)-\mathbf{X}^T(n)\mathbf{W} \end{aligned} \tag{7-13}$$

误差平方为

$$e^2(n)=d^2(n)-2d(n)\mathbf{X}^T(n)\mathbf{W}+\mathbf{W}^T\mathbf{X}(n)\mathbf{X}^T(n)\mathbf{W} \tag{7-14}$$

上式两边取数学期望后，得均方误差

$$\mathrm{E}\left\{e^2(n)\right\}=\mathrm{E}\left\{d^2(n)\right\}-2\mathrm{E}\left\{d(n)\mathbf{X}^T(n)\right\}\mathbf{W}+\mathbf{W}^T\mathrm{E}\left\{\mathbf{X}(n)\mathbf{X}^T(n)\right\}\mathbf{W} \tag{7-15}$$

定义互相关函数向量 $\boldsymbol{R}_{Xd}^T$：

$$\mathbf{R}_{Xd}^T=\mathrm{E}\left\{d(n)\mathbf{X}^T(n)\right\} \tag{7-16}$$

和自相关函数矩阵

$$\mathbf{R}_{XX}=\mathrm{E}\left\{\mathbf{X}(n)\mathbf{X}^T(n)\right\} \tag{7-17}$$

则式（7-15）的均方误差可表述为

$$\mathrm{E}\left\{e^2(n)\right\}=\mathrm{E}\left\{d^2(n)\right\}-2\mathbf{R}_{Xd}^T\mathbf{W}+\mathbf{W}^T\mathbf{R}_{XX}\mathbf{W} \tag{7-18}$$

这表明，均方误差是权系数向量 $\mathbf{W}$ 的二次函数，它是一个中间向上凹的抛物形曲面，是具有唯一最小值的函数。调节权系数使均方误差为最小，相当于沿抛物形曲面下降找最小值。可以用梯度法来求该最小值。

将式(7-18)对权系数 $\mathbf{W}$ 求导数，得到均方误差函数的梯度

$$\nabla(n)=\nabla \mathrm{E}\left\{e^2(n)\right\}=\left[\frac{\partial \mathrm{E}\left\{e^2(n)\right\}}{\partial W_1},\cdots,\frac{\partial \mathrm{E}\left\{e^2(n)\right\}}{\partial W_N}\right]^T$$

$$=-2\boldsymbol{R}_{Xd}+2\boldsymbol{R}_{XX}\boldsymbol{W} \qquad (7\text{-}19)$$

令 $\nabla(n)=0$，即可求出最佳权系数向量

$$\mathbf{W}_{\text{opt}}=\mathbf{R}_{XX}^{-1}\mathbf{R}_{Xd} \qquad (7\text{-}20)$$

将 $\mathbf{W}_{\text{opt}}$ 代入式（7-18），得最小均方误差

$$\mathrm{E}\left\{e^2(n)\right\}_{\min}=\mathrm{E}\left\{d^2(n)\right\}-\mathbf{R}_{Xd}^T\mathbf{W}_{\text{opt}} \qquad (7\text{-}21)$$

利用式(7-21)求最佳权系数向量的精确解需要知道 $\mathbf{R}_{XX}$ 和 $\mathbf{R}_{Xd}$ 的先验统计知识，而且还需要进行矩阵求逆等运算。Widrow 和 Hoff 提出了一种在这些先验统计知识未知时求 $\mathbf{W}_{\text{opt}}$ 的近似值的方法，习惯上称之为 Widrow-Hoff LMS 算法。这种算法的根据是最优化方法中的最速下降法。根据最速下降法，“下一时刻”权系数向量 $\mathbf{W}(n+1)$ 应该等于“现时刻”权系数向量 $\mathbf{W}(n)$ 加上一个负均方误差梯度 $-\nabla(n)$ 的比例项，即

$$\mathbf{W}(n+1)=\mathbf{W}(n)-\mu\nabla(n) \qquad (7\text{-}22)$$

式中 μ 是一个控制收敛速度与稳定性的常数，称之为收敛因子。

不难看出，LMS 算法有两个关键：梯度 $\nabla(n)$ 的计算以及收敛因子 μ 的选择。

精确计算梯度 $\nabla(n)$ 是十分困难的。一种粗略的但是却十分有效的计算 $\nabla(n)$ 的近似方法是：直接取 $e^2(n)$ 作为均方误差 $\mathrm{E}\left\{e^2(n)\right\}$ 的估计值，即

$$\hat{\nabla}(n)=\nabla\left[e^2(n)\right]=2e(n)\nabla\left[e(n)\right] \qquad (7\text{-}23)$$

式中的 $\nabla\left[e(n)\right]$ 为

$$\nabla\left[e(n)\right]=\nabla\left[d(n)-\mathbf{W}^T(n)\mathbf{X}(n)\right]=-\mathbf{X}(n) \qquad (7\text{-}24)$$

将式(7-24)代入式(7-23)中，得到梯度估值

$$\hat{\nabla}(n) = -2e(n)\mathbf{X}(n) \tag{7-25}$$

于是，Widrow-Hoff LMS 算法最终为

$$\mathbf{W}(n+1) = \mathbf{W}(n) + 2\mu e(n)\mathbf{X}(n) \tag{7-26}$$

三、RLS 算法

上节所描述的 LMS 算法本质上是（随机）最陡下降算法，其中梯度向量的真值由数据直接得到的估值来近似。这种最陡下降算法的主要优点是计算简单，但计算简单付出的代价是收敛速度缓慢。我们下面介绍的最小二乘（LS）算法是一种快速收敛算法。该算法判决依据是直接处理接收数据，使其二次性能指数最小，而前面所述的 LMS 算法则是使平方误差的期望值最小。

我们希望设计出一自适应滤波器，通过调节滤波器参数 $\boldsymbol{W}_i$，使得基于过去的观测样本而得到的观测信号 $\hat{s}(n)$ 在某种意义上最逼近原信号 $s(n)$。此时，恢复误差

$$\eta(n) = s(n) - \mathbf{W}^T\mathbf{X}(n) \tag{7-27}$$

另一方面，我们可以将 $\mathbf{W}^T\mathbf{X}(n)$ 视作为 $x(n)$ 的一步预测。因此，可定义预测误差

$$e(n) = x(n) - \mathbf{W}^T\mathbf{X}(n) \tag{7-28}$$

我们设计自适应滤波器的目的自然是希望使恢复误差 $\eta(n)$ 最小。但是，由于真实信号 $s(n)$ 未知，故 $\eta(n)$ 是不可观测的或无法计算的。与此相反，预测误差 $e(n)$ 却是可观测的。由于恢复误差的关系为：

$$e(n) = \eta(n) + n(n) \tag{7-29}$$

而噪声序列 $n(n)$ 是独立的，因此，不可观测的恢复误差 $\eta(n)$ 的最小化等价于可观测的预测误差 $e(n)$ 的最小化。

具体地，我们来考虑

$$\varepsilon(n, \mathbf{W}) = \sum_{i=1}^{n} \lambda^{n-i} \left| e(i) \right|^2 \tag{7-30}$$

的最小化。式中，λ 叫作遗忘因子，通常取 $0 \le \lambda \le 1$。由

$$\frac{\partial \varepsilon(n,\mathbf{W})}{\partial \mathbf{W}} = \frac{\partial}{\partial \mathbf{W}} \sum_{i=1}^{n} \lambda^{n-i} \left[x(i) - \mathbf{W}^T \mathbf{X}(i) \right]^2$$

$$= -2 \sum_{i=1}^{n} \lambda^{n-i} \left[x(i) - \mathbf{W}^T \mathbf{X}(i) \right] \mathbf{X}(i)$$

$$=0$$

可得到等价关系式

$$\sum_{i=1}^{n} \lambda^{n-i} \mathbf{X}(i) \mathbf{X}^T(i) \mathbf{W} = \sum_{i=1}^{n} \lambda^{n-i} x(i) \mathbf{X}(i) \qquad (7\text{-}31)$$

若令

$$\mathbf{R}(n) = \sum_{i=1}^{n} \lambda^{n-i} \mathbf{X}(i) \mathbf{X}^T(i) \qquad (7\text{-}32)$$

$$\mathbf{U}(n) = \sum_{i=1}^{n} \lambda^{n-i} x(i) \mathbf{X}(i) \qquad (7\text{-}33)$$

则式（7-31）可简写为

$$\mathbf{R}(n)\mathbf{W}(n) = \mathbf{U}(n) \qquad (7\text{-}34)$$

假定 $\mathbf{R}(n)$ 是非奇异的，则

$$\mathbf{W}(n) = \mathbf{R}^{-1}(n)\mathbf{U}(n) \qquad (7\text{-}35)$$

这就是滤波器参数的公式，之所以记作 $\mathbf{W}(n)$，是因为 $\boldsymbol{W}$ 随时间而改变。式（7-33）叫做最佳滤波器系数的 Yule-Walker 方程。依据式(7-35)来调整滤波器的参数有两处不便。第一，需要矩阵求逆及矩阵乘法等运算，因而计算量大。第二，$\mathbf{W}(n)$ 与预测误差 $e(n)$ 之间也未建立任何关系，不能实现根据预测误差 $e(n)$ 来调整滤波器参数的要求。

注意，（非平稳或时变）预测误差 $e(n)$ 由

$$e(n) = x(n) - \mathbf{W}^T(n-1)\mathbf{X}(n) \qquad (7\text{-}36)$$

表示。利用此表达式，可以将式（7-33）的 $\mathbf{U}(n)$ 改写作

$$\mathbf{U}(n)=\sum_{i=1}^{n}\lambda^{n-i}\left[\mathbf{W}^T(i-1)\mathbf{X}(i)+e(i)\right]\mathbf{X}(i)$$

注意到 $\mathbf{W}^T(i-1)\mathbf{X}(i)\mathbf{X}(i)=\mathbf{X}(i)\mathbf{X}^T(i)\mathbf{W}(i-1)$ 和式（7-35），用 $\mathbf{R}^{-1}(n)$ 式乘上式后得

$$\mathbf{W}(n)=\mathbf{R}^{-1}(n)\sum_{i=1}^{n}\lambda^{n-i}\mathbf{X}(i)\mathbf{X}^T(i)\mathbf{W}(i-1)+\mathbf{R}^{-1}(n)\sum_{i=1}^{n}\lambda^{n-i}\mathbf{X}^T(i)e(i)$$

$$=\mathbf{W}_1(n)+\mathbf{W}_2(n) \tag{7-37}$$

为了简化第一项 $\mathbf{W}_1(n)$ 的表达，并建立 $\mathbf{W}(n)$ 与 $\mathbf{W}(n-1)$ 之间的关系，一种合理的想法是认为 $n-1$ 时刻及其以前时刻的滤波器参数相同，即

$$\mathbf{W}(0)=\mathbf{W}(1)=\cdots=\mathbf{W}(n-1)$$

这样，利用式(7-32)及上述假定，就有

$$\mathbf{W}_1(n)=\mathbf{R}^{-1}(n)\sum_{i=1}^{n}\lambda^{n-i}\mathbf{X}(i)\mathbf{X}^T(i)\mathbf{W}(n-1)=\mathbf{W}(n-1) \tag{7-38}$$

另一方面，为了简化 $\mathbf{W}_2(n)$ 的表达，一种合理的想法是：认为遗忘因子 $\lambda=0$。这相当于，只有本时刻的结果被记忆下来，而将以前各时刻的结果全部遗忘。从而，有下列简化结果：

$$\mathbf{W}_2(n)=\mathbf{R}^{-1}(n)\sum_{i=1}^{n}0^{n-i}\,\mathbf{X}^T(i)e(i)$$

$$=\mathbf{R}^{-1}(n)\mathbf{X}^T(n)e(n) \tag{7-39}$$

将式(7-38)和式（7-39）代入式（7-37），则得

$$\mathbf{W}(n)=\mathbf{W}(n-1)+\mathbf{R}^{-1}(n)\mathbf{X}^T(n)e(n) \tag{7-40}$$

式（7-40）描述了一个滤波器参数受其输入误差 $e(n)$ 控制的自适应滤波算法，被称为递推最小二乘（RLS）。在某些文献中，也被称为扩展最小二乘(extended least squares)或扩展矩阵方法(extended matrix method)等。

为了实现递推计算，还要解决逆矩阵 $\mathbf{R}^{-1}(n)$ 的递推计算问题。为此，我们先引入一个著名的结果——矩阵求逆引理。

引理 1（矩阵求逆引理）：若 $\boldsymbol{A}$ 是非奇异的，则

$$(\boldsymbol{A}+\boldsymbol{B}\boldsymbol{C}^T)^{-1}=\boldsymbol{A}^{-1}-\boldsymbol{A}^{-1}\boldsymbol{B}(\boldsymbol{I}+\boldsymbol{C}^T\boldsymbol{A}^{-1}\boldsymbol{B})^{-1}\boldsymbol{C}^T\boldsymbol{A}^{-1} \tag{7-41}$$

证明：只要用 $(\boldsymbol{A}+\boldsymbol{B}\boldsymbol{C}^T)$ 左乘上式右边，并验证其结果等于单位阵即可。此证明留给读者作练习。

由 $\mathbf{R}(n)$ 的定义式(7-32)，显然有

$$\mathbf{R}(n)=\mathbf{R}(n-1)+\mathbf{X}(n)\mathbf{X}^T(n)$$

对上式应用矩阵求逆引理，得

$$\mathbf{R}^{-1}(n)=\mathbf{R}^{-1}(n-1)-\frac{\mathbf{R}^{-1}(n-1)\mathbf{X}(n)\mathbf{X}^T(n)\mathbf{R}^{-1}(n-1)}{1+\mathbf{X}^T(n)\mathbf{R}^{-1}(n-1)\mathbf{X}(n)} \tag{7-42}$$

综合以上分析，递推最小二乘自适应算法如下所示。

算法（7-1）（基本 RLS 自适应算法）：

初始化：

$$\boldsymbol{W}(0)=\boldsymbol{0}$$

$$\boldsymbol{R}(0)=\boldsymbol{I}$$

for k=1 to n final do:

$$e(n)=x(n)-\mathbf{X}^T(n)\mathbf{W}(n-1)$$

$$\mathbf{R}^{-1}(n)=\mathbf{R}^{-1}(n-1)-\frac{\mathbf{R}^{-1}(n-1)\mathbf{X}(n)\mathbf{X}^T(n)\mathbf{R}^{-1}(n-1)}{1+\mathbf{X}^T(n)\mathbf{R}^{-1}(n-1)\mathbf{X}(n)}$$

$$\mathbf{W}(n)=\mathbf{W}(n-1)+\mathbf{R}^{-1}(n)\mathbf{X}^T(n)e(n)$$

7.4 自适应均衡器的 DSP 实现

下面我们主要以 LMS 算法为例，描述利用 MATLAB 和 DSP 实现横向滤波器型的自适应均衡器的方法。

7.4.1 LMS 算法的 MATLAB 实现

利用 MATLAB 工具可以对不同的信道模型进行仿真，并对各种结构形式、自适应算

法的均衡器在不同信道模型下的收敛速度和精度进行仿真，其结果对设计实用自适应均衡器具有极其重要的意义。

可以充分利用 MATLAB 提供的函数库，对 ISI 信道、均衡器伪随机训练码、信道加性高斯白噪声进行仿真，并利用其数据输入方便地进行均衡器阶数、ISI 信道参数、信道信噪比（SNR）、训练码序列长度、收敛因子 μ 进行设置，得到不同参数条件下的仿真结果，供设计均衡器时参考。

利用 MATLAB 仿真自适应均衡器模型如图 7-9 所示。其信道模型采用 ISI-AWGN 信道的等效模型，信道参数为 h=(0.28,1,0.28)。

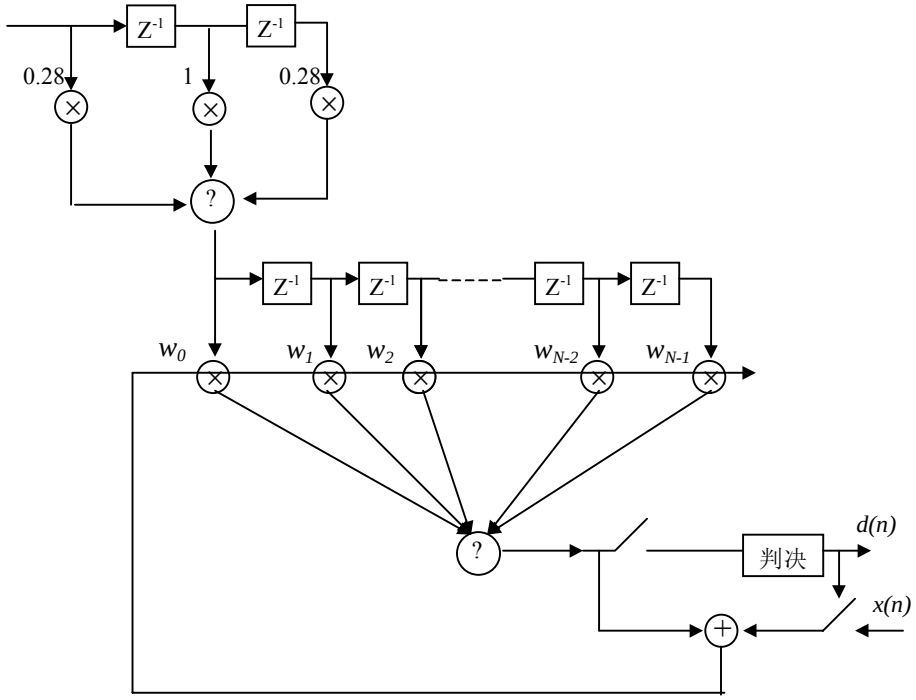


图 7-9 MATLAB 仿真信道模型

在仿真过程中主要用到 MATLAB 以下函数：

- RAND(N)：该函数产生一个 $N \times N$ 矩阵，其数值是在区间（0.0,1.0）之间均匀分布的随机数。
- SIGN（X）：对于每个数据 X，若 X 大于 0，则 SIGN（X）返回 1，若等于 0，则返回 0，若小于 0，则返回-1。
- RANDN（N）：该函数产生一个 $N \times N$ 矩阵，其数值服从均值为 0，方差为 1 的正态分布随机数。

- CONV（A，B）实现 A 矢量和 B 矢量的卷积，其结果矢量长度为 LENGTH(A)+LENGTH(B)-1。

【程序 7-1　横向结构 LMS 自适应均衡器的 MATLAB 实现】

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%   ISI-AWGN channel and LMS adaptive equalizer
%        using transversal structure and LMS algorithm
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%              63
%     y(n)=SUM w(k)*x(n-k)         k=0,1,2,……,63
%              k=0
%     e(n)=d(n)-y(n)
%
%     w(k+1)=w(k)+2 μ *e(n)*x(n)       k=0,1,2,……,63
%
%   where we use filter order=63 and mu=0.02
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%   input simulate parameters
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
isi=input('the chanel parameter(the typical vector:[_,1,_]):isi=');
order=input('order of the adaptive equalizer(usually odd):order=');
snr=input('SNR(dB):snr=');
len=input('the length of training sequence:len=' );
mu=input('the mu value(the typical value<0.1):mu=');%步长经验值.
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%仿真时采用的参数如下：
%isi=[0.28,1,0.28];         %ISI 信道参数
%order=63                          %滤波器阶数
%snr=30;                           %AWGN 信道信噪比
%len=1000;                         %训练序列长度
%mu=0.02;                          %调整步长
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
M=(order-1)/2;
N=len+length(isi)-1;
e=zeros(1,N);
error=e;
out=zeros(1,N);                              % FIR  滤波器输出
```

```
number=0;
for i=1:100
    x=sign(rand(1,len)-0.5);              %采用 PN 码作为训练序列
     noise=randn(1,N)/10.^(snr/10);       %加入 AWGN
     y=conv(isi,x)+noise;
     wk=zeros(1,order);
    for n=order:N-M+1                     %LMS 算法
          y1=y(n+M-1:-1:n-M-1);
          d1=wk*y1';
          e(n)=x(n-2)-d1;                 %时延
          wk=wk+mu*e(n)*y1;
          e(n)=10*log10(abs(e(n)));
    end
    error=error+e;
end
error=error(order:N-M+1)/100;             %误差
```

7.4.2 LMS 算法的 DSP 实现

绝大多数的均衡器是基于最新发展的DSP来设计的。基于DSP的自适应均衡器比用硬件实现的自适应均衡器有很多优点，其功率消耗以及体积更小、更容易实现。由于大多数DSP都有并行的硬件乘法器、流水结构以及快速的片内存储器，这些资源使自适应均衡器的实现更容易且更有效。下面我们给出横向滤波型结构LMS自适应均衡器的TMS320C54x实现的例子。

在 TMS320C54x 中，新增了一条 LMS 指令，该指执行 LMS 算法。该指令采用双数据存储器为操作数，X 和 Y，通过该指令可以实现 X 和 Y 相乘，其结果保存在 B 中，而同时，不改变 T 寄存器的内容，使 T 中可以始终保存着用以更新参数的误差值。利用 LMS 指令，可以方便地实现 LMS 算法。

【程序 7-2】横向结构 LMS 自适应均衡器的 C54x 实现

```
;***********************************************************
;***********************************************************
;   LMS    adaptive filter using transversal structure and LMS algorithm
;***********************************************************
;                63
;     y(n)=SUM w(k)*x(n-k)          k=0,1,2,……,63
;               k=0
;
;     e(n)=d(n)-y(n)
;
;     w(k)=w(k-1)+2 μ *e(n)*x(n-k)        k=0,1,2,……,63
```

```
;
;   where we use filter order=64 and mu=0.01
;************************************************************
; 自适应均衡器参数为
;   FILT_SIZE       滤波器阶数
;   mu              LMS 算法的
;   FRAME_SIZE  数据帧长度
;---------------------------------------
ADAPT_DP    .usect      "adpt_var",0
d_primary   .usect      "adapt_var",1           ;d(n)
d_output    .usect      "adapt_var",1           ;y(n)
d_error     .usect      "adapt_var",1           ;e(n)
d_mu        .usect      "adapt_var",1           ;mu
d_mu_e      .usect      "adapt_var",1           ;mu*e(n)
d_new_x     .usect      "adapt_var",1           ;新输入数据
d_adapt_count.usect     "adapt_var",1           ;计数器
hcoff       .usect      "bufferh",FILT_SIZE     ;模拟信道参数
wcoff       .usect      "bufferw",FILT_SIZE     ;自适应均衡器参数
xh          .usect      "bufferx",FILT_SIZE     ;模拟信道输入数据
xw          .usect      "bufferp",FILT_SIZE     ;自适应均衡器输入数据
;---------------------------------------
; 自适应均衡器初始化程序
;---------------------------------------
adapt_init:
        STM     #xw,AR1
        RPTZ    A,#FILT_SIZE-1
        STL     A,*AR1+

        STM     #d_primary,AR1
        RPTZ    A,#6
        STL     A,*AR1+

        STM     #hcoff,AR1
        RPT     #FILT_SIZE-1
        MVPD    #scoff,*AR1+
        LD      #ADAPT_DP
        ST      #K_mu,d_mu
        STM     #1,AR0
;---------------------------------------
```

```
; LMS 算法
;----------------------------------------
adapt_task
      STM     #FILT_SIZE,BK         ; 设置循环存储器大小
      STM     #hcoff,AR2
      ADDM    #1,d_adapt_count
      LD      *AR6+,A
      STM     #wcoff,AR4
      STL     A,d_new_x             ; 读进新数据
      LD      d_new_x,A
      STL     A,*AR3+0%
      RPTZ    A,# FILT_SIZE-1       ; 重复 128 次
      MAC     *AR2+0%,*AR3+0%,A ;
      STH     A,d_primary           ;计算 d(n)

;start simultaneous filtering and updating the adaptive filter here.
      LD      d_mu_e,T
      SUB     B,B
      STM     #(FILT_SIZE-2),BRC    ; 设置循环次数
      RPTBD   lms_end-1
      MPY     *AR5+0%,A             ;
      LMS     *AR4,*AR5             ;
      ST      A,*AR4                ; 更新滤波器系数
      MPY     *AR5+0%,A             ;
      LMS     *AR4,*AR5             ; B = 输出 y(n) 并更新滤波器系数

lms_end
      STH     A,*AR4          ; 滤波器最后参数
      MPY     *AR5,A          ; x(0)*h(0)
      MVKD    #d_new_x,*AR5   ; 读出新样点
      LMS     *AR4,*AR5+0%
      STH     B,d_output      ;存贮滤波器输出
      LD      d_primary,A
      SUB     d_output,A
      STL     A, d_error      ; e(n)=d(n)-y(n)
      LD      d_mu,T
      MPY     d_error,A       ; A=u*e
      STH     A,d_mu_e
      LD      d_error,A
```

```
STL     A, *AR7+
LD      #FRAME_SIZE,A  ;
SUB     d_adapt_count,A
BC      adapt_task,AGT          ;检查数据都已处理完否，如果未完成继续循环
RET
```

第八章 分集接收的 DSP 实现

在无线通信中，发射信号要经直射、反射、散射等多条传播路径到达接收端，这些多径信号相互叠加会形成衰落，其中快衰落的衰落深度可达 40dB，偶尔可达 80dB。衰落会严重影响通信质量（如会导致数字信号的高误码率等），为了减少其影响，可采用加大发射功率的办法，但实际上这种办法的代价太大，且会造成对其他电台的干扰。为此，作为一种优先的方案，人们往往采用信号处理技术来抗衰落，分集接收仅仅是其中的一种。

分集接收是利用信号和信道的性质，将接收到的多径信号分离成互不相关（独立的）的多径信号，然后将多径衰落信道分散的能量更有效地接收起来处理之后进行判决，从而达到抗衰落的目的。分集技术是一项研究利用信号的基本参量在时域、频域和空域中，如何分散开又如何收集起来的技术。分集技术早已在模拟无线通信中得到成功应用，近年来分集技术在数字无线通信中得到更加广泛的应用，如在移动通信中，基站广泛采用二重空间分集接收，CDMA 手机和基站中采用 RAKE（多径）接收机进行分集接收，来减小衰落的影响；在短波通信中，采用具有带内频率分集和编码分集的多路并传方法来降低系统的误码率；在对流层散射通信中，采用四重频率分集的时频调制来提高系统性能。

本章将介绍分集接收的基本概念与基本原理，并在此基础上，以 RAKE 接收和交织为例，讨论其设计及 MATLAB、DSP 实现问题。

8.1 分集技术

8.1.1 分集接收的基本概念

目前使用的典型抗衰落方法有信道编码、均衡、扩频和分集。其中差错控制通过增加信息的冗余度来纠正衰落引起的误码；均衡主要通过补偿信道衰落引起的畸变来减小衰落的影响；扩频是通过特殊的信号设计所带来的分离多径信号的能力，来消除引起衰落的多径信号干涉效应；而分集是有意识地分离多径信号并恰当合并以提高接收信号的信噪比来抗衰落。当然，为有效地抗衰落，实际运用中，往往是联合使用上述抗衰落方法，例如，分集与差错控制相结合，可降低传输错误率；而为减小衰落引起的信噪比波动对均衡器性能的影响，在均衡器前使用分集可减小信噪比波动，进而达到提高系统性能的目的。

分集是一种有效的通信接收方式，它能以较低的成本改善系统的性能。分集的概念可简单解释如下：如果一个无线路径经历深衰落，那么另一个相对独立的路径中可能仍保持着较强的信号。因此，一旦在多径信号中选择出两个或两个以上的信号，接收机的瞬时信噪比和平均信噪比就可得到改善，改善幅度通常可以达到 20 到 30dB。

分集的基本思想是将接收到的多径信号分离成不相关的（独立的）多路信号，然后把

这些多路信号分离信号的能量按一定的规则合并起来，使接收到的有用信号能量最大，进而提高接收信号的信噪比。因此，分集接收包括两个方面的内容：一是如何把接收的多径信号分离出来使其互不相关，二是将分离出来的多径信号恰当合并，以获得最大信噪比。

8.1.2 分集方式

分集分为宏观分集和微观分集两大类。宏观分集也称为多基站分集，其主要作用是抗慢衰落。例如，在移动通信系统中，把多个基站设置在不同的物理位置上（如蜂窝小区的对角线上），同时发射相同的信号，小区内的移动台选择其中最好的基站与之通信，以减小地形、地物及大气等对信号造成的慢衰落。

微观分集的主要作用是抗快衰落。理论与实验都证明，当信号在空间、频率及时间等方面分离时，都会呈现出互相独立的衰落特性，由此按路径分离的不同，微观分集可分为以下几种：

（1）空间分集

空间分集的依据是间隔相距达到一定程度不同接收地点收到信号的衰落独立性。空间分集的基本结构为发端使用一副天线发射，收端多部天线接收。

空间分集也称设备分集，已被广泛用于短波通信和移动通信，如二重空间分集的短波FSK 通信，较不分集时好 12dB，相当于发射功率增加 15 倍；移动通信蜂窝基站系统广泛采用两副接收天线来接收同一移动台发来的信号，以改善性能。

（2）频率分集。

由于频率间隔大于相关带宽的两个信号所遭受的衰落是不相关的，因此，可以用两个以上不同的频率传输同一信息，以实现频率分集。如在 GSM 系统中利用跳频来达到频率分集；在 IS-95CDMA 系统中，利用发射信号带宽（1.25MHz）远大于相关带宽（市区和郊区的相关带宽分别是 50kHz 和 250kHz）获得频率分集效果。

（3）时间分集

实践证明，快衰落除了具有空间和频率的独立性外，还具有时间的独立性，即同一信号在不同的时间区间多次重发，只要两次重发间隔足够大，那么各次发送的信号通过信道后，出现的衰落将是彼此独立的。时间分集主要用于在衰落信道中传输数字信号，如短波和移动通信系统中，用交织来改变原来的码元顺序供发端发射，以达到时间分集的目的。此外，时间分集也有利于克服短波信道和移动信道中由于多普勒效应引起的信号衰落现象。

（4）极化分集

由于两个不同极化的电磁波具有独立的衰落特性，所以发端和收端可以用两个位置很近但为不同极化方向的天线发送和接收信号，以获得分集效果。

极化分集可以看作空间分集的一种特殊情况，其优点是设备的结构紧凑，节省空间，缺点是由于发射功率要分配到两副天线上，因此有 3dB 损失。

（5）角度分集

角度分集的依据是从不同入射角的两个信号所受到的衰落将是彼此独立的特性，其基本结构是接收端使用多个锐方向性接收天线来接收不同方向来的信号分量。

角度分集也可看作空间分集的一种特殊情况，显然，它在较高频率时比较容易实现。

（6）路径分集

路径分集的依据是来自两个不同路径信号的时延大于某一值时，这两个衰落信号可看作互不相关的特性。

一般系统中的多径信号很难分离，但在扩频通信系统中，存在多径信号的内在可分离特性，即具有抗多径干扰的能力。当然，若不采用路径分集时，则多径干扰被当作噪声处理；若采用路径分集，则将把多径干扰变害为利，如使用 RAKE 接收可将多个路径来的同一码序列的波形适当合并，获得高性能的信号。

上述分集有时又分为显分集和隐分集两类，其依据是由分集作用是否隐含在被传输的信号之中。例如，交织编码、路径分集等方式属于隐分集，而空间分集、角度分集等属于显分集。由于隐分集主要利用信号设计技术，可以大大降低实现成本，所以是发展最快，最有前途的一种分集方式。

8.1.3 分集合并技术

分集接收时，从接收端得到的 L 个不同的独立的信号，可以通过不同形式的合并技术来获得合并增益。从接收链路看，可以在中频和射频上进行，也可以在基带上进行。

合并时采用的准则和方式有多种形式，下面予以介绍：

一、信号合并准则

1. 合并信号的表达式

设分集重数为 L，则合并的信号 S 可以表达为

$$S=a_1S_1+a_2S_2+\cdots+a_LS_L \tag{8-1}$$

其中 a_i 为加权系数，s_i 为第 i 个分离的独立信号，$i=1，2，3，\cdots，L$。选择不同的加权系数就形成了不同的合并方法。

2. 信号合并准则

（1）最大信噪比准则

该准则最早用于模拟信号的合并，但应用于数字信号合并时应注意，在频率选择性衰落信道中，最大信噪比准则并不一定是最佳的。

（2）眼图最大张开准则

该准则适用于数字信号的合并。眼图张开最大，表征码间干扰最小，因此，在频率选择性衰落信道中，眼图最大张开度准则是一种最佳的信号合并准则。

（3）误码率最小准则

数字信号合并的最终结果是希望误符号率达到最小。因此，误符号率最小准则就是数字信号的最佳合并准则。

二、最大信噪比准则下的信号合并方法

1. 选择性合并

选择性合并方法是在多支路接收信号中,选取信噪比最高的支路信号作为输出信号，其原理框图如图（8-1）所示。

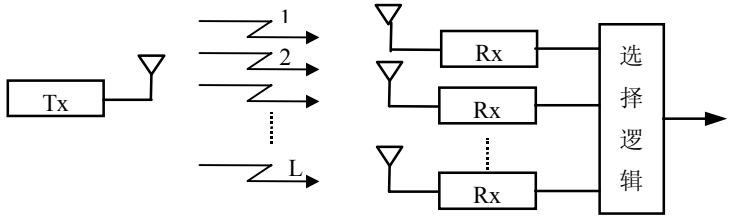


图 8-1 选择性合并原理框图

接收端是 $i=1，2，\cdots，L$ 的 L 个分集支路接收机，利用选择逻辑选择其中具有最大基带信噪比的某一路基带作为输出。

若每一支路的平均信噪比为 $\overline{SNR}$，则选择合并后的平均输出信噪比为

$$\overline{SNR}_B = \overline{SNR} \sum_{k=1}^{L} 1/k \tag{8-2}$$

其中，L 为合并路径数目，或分集重数。

由式（8-2）得合并处理增益

$$G_s = \frac{\overline{SNR_S}}{\overline{SNR}} = \sum_{k=1}^{L} \frac{1}{k} \tag{8-3}$$

2. 最大比值合并

最大比值合并原理框图如图 8-2 所示。每一路有一个加权，加权的权重依各支路信噪比来分配，信噪比大的支路权重大，信噪比小的支路权重小。由式（8-1），合并输出信号的信噪比可表示为

$$SNR = \sum_{i=1}^{L} a_i SNR_i \tag{8-4}$$

其中 a_i 为第 i 支路的加权系数；SNR_i 为第 i 支路输出信噪比。

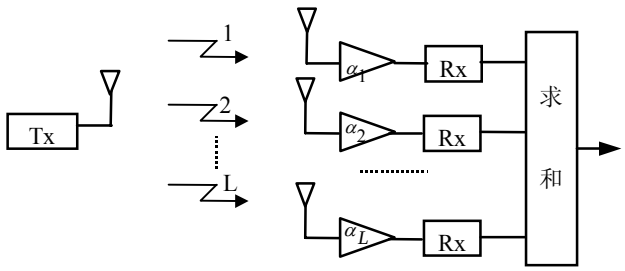


图 8-2 最大比值合并原理框图

可以证明，最大比值合并的处理增益为：

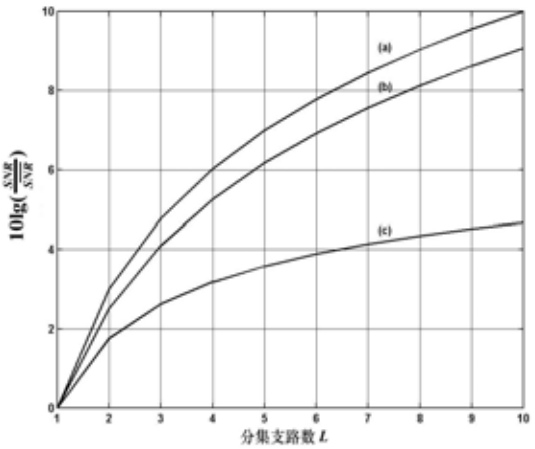
$$G_m=L \tag{8-5}$$

3. 等增益合并

在上述最大比值合并中，取a_i =1（i=1，2，…，L。）即为等增益合并。等增益合并性能仅次于最大比值合并，当 L 较大时，其性能比最大比值合并相关不多，约 1dB 左右。等增益合并实现比较容易，其设备也简单。

4. 最大信噪比准则下几种合并法的性能比较

图 8-3 给出了三种合并方式平均信噪比的改善程度，其中曲线（c）为选择式合并；（a）为最大比值合并；(b)为等增益合并。由图可见，最大比值合并和等增益合并使平均信噪比获得较大的改善。



(a)最大比值合并　(b)等增益合并　(c)选择式合并

图 8-3 三种合并方式平均信噪比的改善程度示意图

8.2 RAKE 接收机的设计基础

8.2.1 多径信号的分离与合并

一、多径分集的特点

通常接收的多径信号时延差很小且是随机的，叠加后的多径信号一般很难分离。所以，在一般的分集中，需要建立多个独立路径信号，在接收端按最佳合并准则进行接收。而多径分集是通过发端的特定信号设计来达到接收端接收多径信号的分离，它是扩频系统所特有的。

多径分集不同于时域均衡，时域均衡多用于信号不可分离多径的情况，它主要用于解决符号（码间）干扰问题，而多径分集用于信号可分离多径的情况，其目的在于减少时延扩展引起的符号干扰。

二、多径信号的分离与合并

多径信号分离的基础是采用直接扩频信号。要求直扩系统的时间（T）带宽（W）积远大于 1，即 $TW>>1$。对于带宽为 W 的系统，所能分离的最小路径时延差为 $1/W$。对于直扩序列的码片（chip）宽度为 T_C 的系统，所能分离的最小路径时延差为 T_C，并且要求所采用的直扩序列信号的自相关性和互相关性要好。

多径信号的合并所要解决的问题是如何调整各独立路径的时延，按一定的统计规律组合成一路信号。序列的多径合并准则有：①第一路径准则（EP）②最强路径准则（LP）③检测后积分准则（PDI）④等增益合并准则（EGC）⑤最大比值合并准则（MRC）⑥自适应合并准则。具体采用何种准则依具体应用情况而定。

8.2.2 RAKE 接收机的工作原理

RAKE 接收机的简单原理是对每个路径使用一个相关接收机，各相关接收机与被接收信号的一个延迟形式相关，然后对每个相关器的输出进行进行加权，并把加权后的输出相加合成一个输出，以提供优于单路相关器的信号检测，然后在此基础上进行解调和判决。由于这种接收机收集所有接收路径上的信号，其作用与农用多齿草耙（英文名“RAKE”）的作用相似，故称为 RAKE 接收机。

RAKE 接收技术是 CDMA 蜂窝移动通信系统中的关键技术之一，在 CDMA 蜂窝移动通信系统中，信道带宽远远大于信道的相关带宽，这样，在无线信道传输中出现的时延扩展，可以被看作只是被传信号的再次传送，如果这些多径信号相互间的延时超过了一个码片的长度，那么他们就可看作是非相关的。

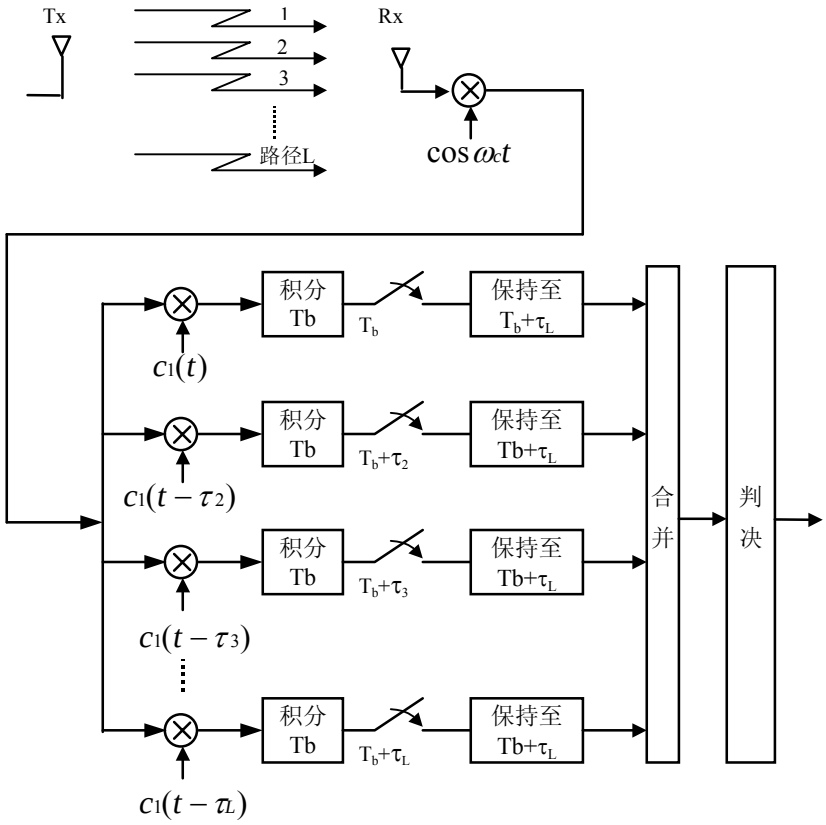


图 8-4 简化的 RAKE 接收机组成

假设发端从 T_X 发出的信号经 L 条路径到达接收天线 R_X。 路径 1 距离最短，传输时延也最小，依次是第 2 路径、第三路径…、时延时间最长的是第 L 条路径。通过电路测定各条路径的相对时延差，以第 1 条路径为基准时，第二条相对于第一条路径的相对时延差为 τ_2，第三条相对于第一条路径相对时延差为 τ_3…、第 L 条路径相对于第一条路径相对时延差为 τ_L，且有 $\tau_L > \tau_{L-1} > \cdots > \tau_3 > \tau_2$（ $\tau_1 = 0$）。

接收端通过解调后，送入 L 个并行相关器（例如，在 IS-95CDMA 系统中，基站接收机 $L=4$，移动台接收机 $L=3$），图中为用户 1，即使用 PN 码 $c_1(t)$，通过位同步，各个相关器的本地码分别为 $c_1(t)$、$c_1(t-\tau_2)$、$c_1(t-\tau_3)$ …$c_1(t-\tau_L)$。经过解扩加入积分器，每次积分时间为 T_b，每一支路在 T_b 末尾进入电平保持电路，保持直到 $T_b+\tau_L$，这样 L 个相关器于 $T_b+\tau_L$ 的时刻输出，通过加权再相加合并，经判决器产生数据输出。

显然，如果接收机只有一个单独的相关器，一旦接收机中使用的单个相关器输出被衰落扰乱时，接收机就不可能校正此值，从而导致判决器大量误判。而在 RAKE 接收机中，若一个相关器的输出被衰落扰乱了，还可以有其他相关器支路作出补救，并且通过改变被扰乱之路的权重，还可以消除此路信号的负面影响。由于 RAKE 接收机提供了对 L 路信号良好的统计判决，因而它能有效抗多径衰落。

假设 L 个相关器的输出分别为 Z_i，如图 8-4 所示，分集合并后的输出为

$$Z'(t)=\sum_{i=1}^{L}a_i(t)Z_i(t) \qquad (8\text{-}6)$$

加权系数 $a_i(t)$ 根据对应的信号 $Z_i(t)$的能量在 M 个相关器输出信号的总能量所占比重来选择，即

$$a_i(t)=\frac{Z^2{}_i(t)}{\sum_{l=1}^{L}Z_l{}^2(t)}, \qquad i=1,\cdots,2 \qquad (8\text{-}7)$$

当相关器 i 的输出被衰落扰乱时，其输出 Z_i 的能量就小，对应的加权系数 $a_i(t)$ 自动取小，从而使得该分量在式（8-6）的作用减小，结果达到抑制多径衰落的效果。

8.2.3 RAKE 接收机的设计考虑

一、性能比较

如果有足够多的可分离的多径数，则支路相等的最大比值合并 RAKE 接收机的性能优于等增益合并的 RAKE 接收机，但是当只有一路较强的多径信号或大部分多径信号的时延位于一个扩频码元的宽度 T_C 之内时，即出现图 8-5 所示的信道冲激响应时，等增益合并可能不如选择性合并。原因是，此时等增益合并 RAKE 接收机只有一条支路接收到较强的信号，而其它支路只收到很少的信号能量，降低了等增益合并 RAKE 接收机的信噪比。

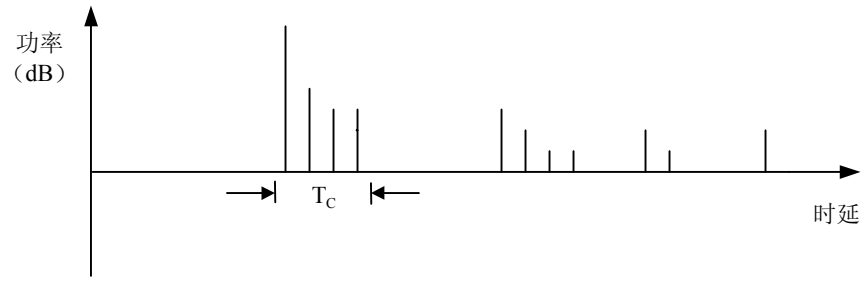


图 8-5 信道的冲击响应

根据扩频带宽的选择，多径环境下可能有几路到几十路可分离的多径信号，有的多径信号只包含很少的信号能量，所以，RAKE 接收机不需要分集接收所有的多径信号。为此，除了根据信道的特性，选择适当的 RAKE 支路外，还可以在 RAKE 接收机的每个支路设置一个门限，当信号电平低于门限值时将该支路关闭，以防止信噪比很低的分集支路对 RAKE 接收机的影响。

二、RAKE 接收机支路数的选择

RAKE 接收机支路数的选择要根据信道特性和复杂度来考虑。除此之外，即使有足够多的可分离的多径信号，RAKE 接收机的支路数增加到一定程度，性能改善还不明显。

对于 IS-95 CDMA 系统，多径特性的现场测试表明，不同路径数的概率分布是：3 路径为 20%，2 路径为 50%，1 路径为 28%。这说明，RAKE 接收机至少要具有处理 3 个路径的分集能力，考虑到具体实现的复杂度，IS-95 CDMA 基站使用 4 重分集，移动台采用 3 重分集。

三、扩频带宽的选择

采用 RAKE 接收机的扩频系统带宽的选择要考虑到多径环境的多径时延、接收机的复杂度及同步的实现。

扩频带宽（系统带宽）为 W 的扩频信号的时间分辨率为 $1/W$，一般的扩频接收机只有一个相关器，接收时间窗 $T_W=1/W$ 内的多径信号能量，时间窗 T_W 外的多径信号则被抑制掉了。这种单一相关器的接收机只接收到一小部分有用信号的能量，并未充分利用有用信号，而等增益合并和最大比值合并的 RAKE 接收就可以接收到时间窗 $T_W=L/W$ 内的多径信号（L 为 RAKE 接收机的支路数），既利用了更多的有用信号能量。如果增加扩频带宽，可分离的多径数增加，就需要相应增加 RAKE 接收机的支路数，否则未接收的多径信号会形成干扰。如果多径信道的最大时延为 3μs（室内信道的典型值），用 IS-95 CDMA 的 1.25MHz 扩频带宽，时间分辨率约 1μs，用 3 个支路的等增益合并或最大比值合并 RAKE 接收机就几乎可以接收到所有的信号能量，若用 10MHz 的扩频带宽、时间分辨率为 0.1μs，用 3 个支路的等增益合并或最大比值合并 RAKE 接收机只能收到时间窗 $T_W=0.3\mu s$（<<3μs）内的多径信号能量，收到的信号能量只占总能量的一小部分，同时未接收到的多径信号成为干扰信号。因此，对于采用 RAKE 接收机的扩频系统的扩频带宽的选择要根据实际情况而定。对于微小区或市内通信环境，多径时延比较小，需要用宽带扩频信号，以保证有足够多的可分离的多径信号，实现有效地 RAKE 接收。对于市区环境的小区，多径时延较大，如果用宽带扩频信号，则需要多支路数的 RAKE 接收机，硬件复杂度高，综合各种因素，IS-95 CDMA 采用 1.25MHz 扩频带宽，这样在复杂度不高的情况下，RAKE 接收机仍有较好的性能。

8.2.4 RAKE 接收的的几种方式

下面以差分 PSK（DPSK）为例说明 RAKE 接收的的几种方式。

一、检测后积分（PDI，Post Detection Integration）多径接收机

多径时延未知时，最简单的多径分集方式是采用检波，后积分的方法既在接收机检测器后面设置一个积分时间等于多径扩展 τ_M 秒的积分器，如图 8-6 所示：

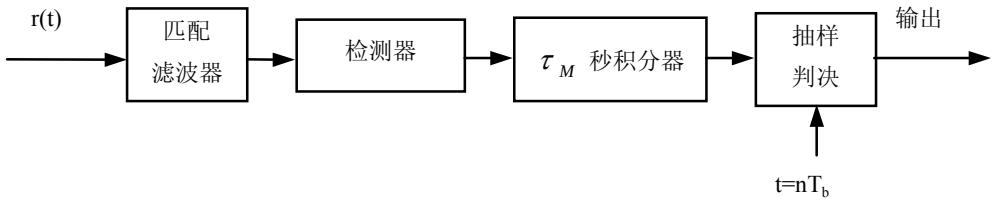


图 8-6 PDI 接收机框图

PDI 接收机的优点是实现起来比较简单，但存在一个约束条件，即 τ_M 不能大于码元宽度 T_b，否则多径分两会落到相邻码元中而造成码间干扰。

PDI 接收机的主要缺点是没有利用多径的参量信息，而简单的积分不可避免的要把 τ_M 秒范围中的干扰与噪声包含进去，因而影响多径分集效果。为了克服这一不足，可对多个峰值进行筛选，把那些幅度明显大于噪声背景的多径分量取出，在对他们进行延时和相位校正，使之在某一时刻对齐，并按一定规则合并。

二、具有信道参数测量的 DPSK –RAKE 接收机

为了获得信道参数，可采用探测信号，探测信号可以是专门的信号（训练序列），也可以使数据信号本身，接收机通过接收探测信号来估计信道参数 $\{\hat{a}_k\}$ 和 $\{\hat{\tau}_k\}$,图 8-7 给出了具有信道参量的 RAKE 接收机的结构。

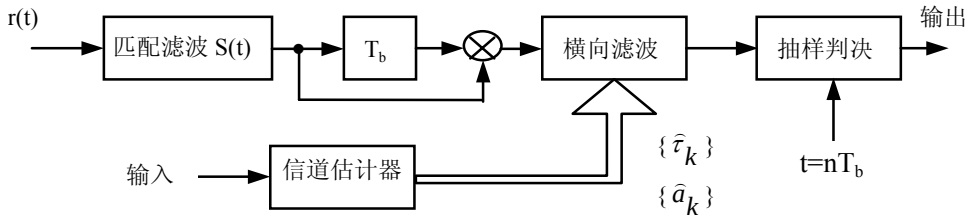


图 8-7 具有信道估计的 DPSK-RAKE 接收机结构

三、具有信道参数测量的 DPSK –DRAKE 接收机

当 RAKE 接收机采用等增益合并准则时，则称之为 DPSK –DRAKE 接收机，其结构图如图 8-8 所示。图中的估值由信道估计器获得，并用来控制横向滤波器的抽头时延。横向滤波器采用相同的抽头加权系数，即 $\hat{a}_k = 1$。

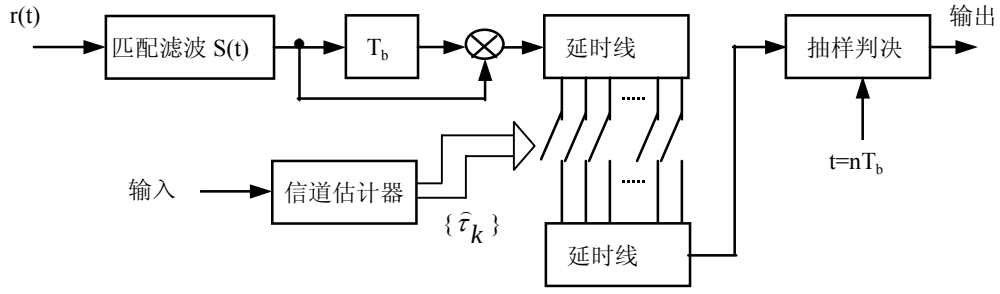


图 8-8 具有信道参数测量的 DPSK –DRAKE 接收机

四、并行相关 RAKE 接收机

并行相关 RAKE 接收机的原理如图 8-9 所示，图中的搜索器的作用是搜索所有多径，估计出多径的相位、到达时刻和强度参数，并从中选出三路最强的多径信号供相关器作相关处理，然后再合并。

对于 IS-95CDMA 系统，基站中的 RAKE 接收机有 4 个并行相关器和 2 个搜索相关器

组成，基站接收机无法得到多径信号的相位信息，一般采用非相关最大比值合并准则；而移动台中的 RAKE 接收机有 3 个并行相关器和一个搜索相关器组成，它可利用基站发送的导频信号估计出多径信号的相位、到达时刻和强度参数。

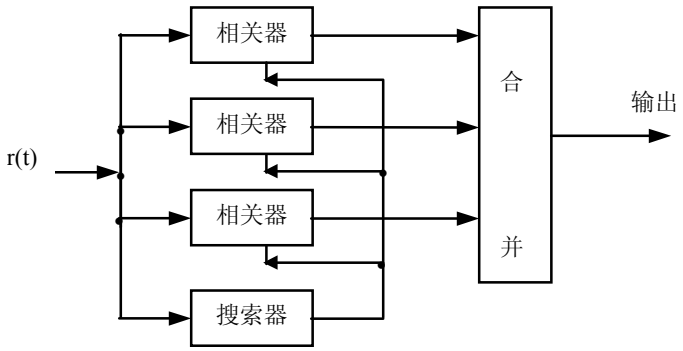


图 8-9 并行相关 RAKE 接收机

8.3 RAKE 接收机的 DSP 实现

RAKE 接收的原理比较简单，但应用于各种不同场合后，为了获得高性能，往往与其它技术综合，形成了以 RAKE 接收为核心的不同接收机结构 ，如在 IS－95CDMA 基站中，其 RAKE 接收机是由相关解调器、路径合并器、解交织器和比特译码器等部分组成的，相应的实现一般用 Qualcomm 公司的 ASIC 芯片或 FPGA 芯片实现。本节以适用于多径慢衰落信道的 RAKE 接收机为例，分析其性能及 MATLAB、DSP 实现。

8.3.1 RAKE 接收机性能的理论分析

下面以小区内反向链路的 RAKE 接收机为例进行分析。设小区内的所有用户都与一个基站通信，基站用 RAKE 接收机分别对每个用户分集接收，如图 8－10 所示。

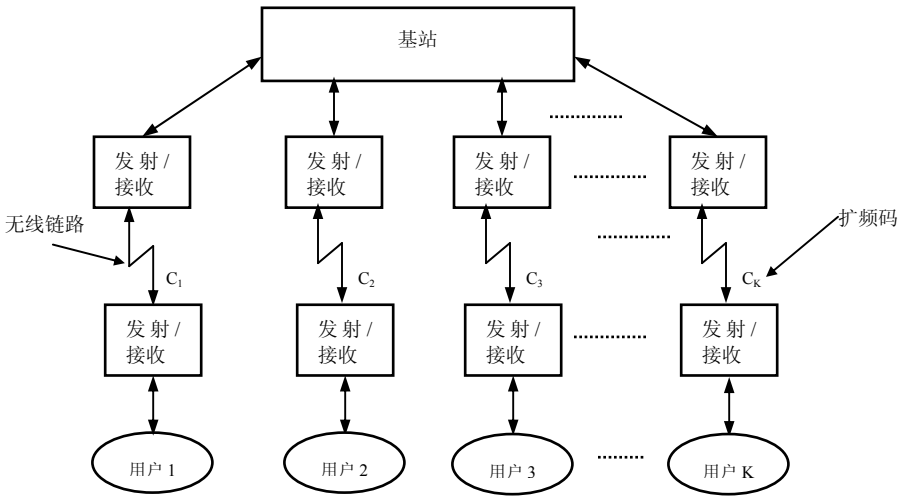


图 8-10　基站与移动台之间的连接示意图

多径信号相对时延估计的 RAKE 接收机的实现结构图由图 8-4 所示，由图可见在多径信号到达时刻分别对各路多径信号分集合并。

设小区内有 K 个同时工作的用户，BPSK 调制的第 k 个用户的发射信号为：

$$S_k(t)=Ac_k(t)b_k(t)\cos(\omega_c t+\phi_k) \qquad (8\text{-}8)$$

式中ω_C是载波频率，A 是信号幅度，ϕ_k是信号相位，$b_k(t)$是第 k 个用户的数据信号，它为脉宽等于 T_b的矩形脉冲序列，幅度等概率取值±1。$c_k(t)$为取值±1 的扩频码序列，码长为 L_C，T_C是扩频码的码元宽度，且满足 $T_b=L_CT_C$ 。

设用户和基站之间的信道是慢变的瑞利多径衰落信道，即在两个相邻数据比特间隔内，与信道相联系的随机参数没有很大变化，此时，信道的复低通响应可写为：

$$h_k(\tau)=\sum_{l=1}^{L_p} a_{lk}\delta(t-\tau_{lk})\exp(j\varphi_{lk}) \qquad (8\text{-}9)$$

式中 a_{lk}, τ_{lk}和φ_{lk}分别表示第 k 个用户，第 l 条路径的损耗、时延和相位，δ（t）是单位冲激响应函数。多径数目 L_p由具体系统确定，它小于或等于系统最大可分离的多径数目。

设多径相位φ_{lk}为［0，2π］内均匀分布的独立随机变量；多径时延　　τ_{lk}也是独立的，且服从［0，T_b］内的均匀分布；a_{lk}是独立的瑞利分布的随机变量，其分布概率密度函数为：

$$f_a=\begin{cases}\dfrac{y}{\beta}\exp\left(\dfrac{-y^2}{2\beta}\right) & y\geq 0\\ 0 & y\leq 0\end{cases} \qquad (8\text{-}10)$$

令φ_k=0（不影响分析结果），由式（8-8）和式（8-9）卷积积分得出基站的接收信号：

$$r(t)=A\sum_{l=1}^{L_p} a_{l1}c_1(t-\tau_{l1})b_1(t-\tau_{l1})\cos(\omega_c t+\varphi_{l1})+$$
$$A\sum_{l=1}^{L_p}\sum_{k=2}^{K} a_{lk}c_k(t-\tau_{lk})b_k(t-\tau_{lk})\cos(\omega_c t+\varphi_{lk})+n(t) \qquad (8\text{-}11)$$

式中 n(t)是功率谱密度为 N_0/2W/HZ 的高斯白噪声。

设对应于第一个用户的 RAKE 接收机对多径信号的相关解调，则 RAKE 接收机的第 j 条支路的判决变量为：

$$\xi_j = \int_0^{Tb} r(t)a_1(t-\tau_{j1})\cos(\omega_c t+\varphi_{j1})dt \qquad (8\text{-}12)$$

如果 RAKE 接收机有 L 个分集支路（ $L \le L_p$ ），则有 L 个判决变量 ξ_j（j=1, 2, …, L）。将式（8-11）代入式（8-12）得：

$$\begin{aligned}\xi_j = {} & b_1(0)\frac{AT}{2}a_{j1} + \frac{A}{2}\sum_{\substack{l=1\\ l\neq j}}^{L_p} a_{l1}\cos(\varphi_{l1}-\varphi_{j1})\cdot\int_0^{Tb} c_1(t-\tau_{l1})b_1(t-\tau_{l1}) \\ & \cdot c_1(t-\tau_{j1})dt + \frac{A}{2}\sum_{l=1}^{L_p}\sum_{k=2}^{K} a_{lk}\cos(\varphi_{lk}-\varphi_{j1})\cdot\int_0^{Tb} c_k(t-\tau_{lk})b_k(t-\tau_{lk}) \\ & \cdot c_1(t-\tau_{j1})dt + \int_0^{Tb} n(t)c_1(t-\tau_{j1})\cos(\omega_c t+\varphi_{j1})dt\end{aligned} \qquad (8\text{-}13)$$

式中 $b_1(0)$时要判决的信息数据比特，且假定每个用户与基站间都有 L_p 条路径。

为简化上述表述，定义如下物理变量：

$$R_{k1}(\tau) = \int_0^{\tau} c_k(t-\tau)c_1(t)dt \qquad (8\text{-}14)$$

$$\hat{R}_{k1}(\tau) = \int_{\tau}^{Tb} c_k(t-\tau)c_1(t)dt \qquad (8\text{-}15)$$

$$v(\tau_{j1}) = \int_0^{Tb} n(t)c_1(t-\tau_{j1})\cos(\omega_c t+\varphi_{j1})dt \qquad (8\text{-}16)$$

将以上三式代入式（8-13）得

$$\begin{aligned}\xi_j = {} & b_1(0)\frac{AT}{2}a_{j1} + \frac{A}{2}\sum_{\substack{l=1\\ l\neq j}}^{L_p} a_{l1}\cos(\varphi_{l1}-\varphi_{j1})\cdot[b_1(-1)R_{11}(t_{l1}) + b_1(0)\hat{R}_{11}(t_{lk})] \\ & + \frac{A}{2}\sum_{l=1}^{L_p}\sum_{k=2}^{K} a_{lk}\cos(\varphi_{lk}-\varphi_{j1})\cdot[b_k(-1)R_{k1}(t_{lk}) + b_k(0)\hat{R}_{k1}(t_{lk})] + v\end{aligned} \qquad (8\text{-}17)$$

式中 $t_{lk}=\tau_{lk}-\tau_{j1}$

式（8-17）中第 1 项是所需信号，第 2 项是由于自相关旁瓣引起的自干扰，第 3 项是

其它 K-1 个用户的多址干扰，最后一项是高斯噪声。

式（8-17）中的多址干扰项的计算比较困难，为估算系统的性能，这里将所有多址干扰近似为高斯白噪声。这样，对固定的 $a_1=a_{j1}$，所需信号功率为 $a_1^2A^2T_b^2/4$，最后一项是均值为零，方差为 $N_0T_b/4$ 的高斯随机变量，相应有[26]：

所需信号功率=$\left(\frac{AT_b}{2}\right)^2 a_1^2$

干扰功率=$(L_pK-1)\left(\frac{AT_b}{2}\right)^2 e^2E\left(a^2\right)/2$

噪声功率=$N_0T_b/4$

上面干扰功率中的 a 是除了 a_1 之外的任一个瑞利分布的随机变量 a_{lk}，e^2 为：

$$e^2=E\left\{\frac{[\alpha_{-1}R(t_1)+\alpha_0\hat{R}(t_1)]^2}{T^2}\right\} \qquad (8\text{-}18)$$

对于 Gold 码，$e^2=2/(3N)$。

在假定多址干扰为高斯噪声的情况下，可推出在 a_1 上的条件误码率为：

$$P(e|a_1)=\frac{1}{2}erfc(\sqrt{\eta}) \qquad (8\text{-}19)$$

式（8-19）中η为

$$\eta=\frac{2a_1^2E_b}{(L_pK-1)e^2E_bE(a^2)+N_0} \qquad (8\text{-}20)$$

式中，E_b=（A^2T_b）/ 2。若各条路径的增益平方的均值相等，则η的平均值η_0 为：

$$\eta_0=\frac{2\overline{E_b}}{\lambda e^2\overline{E_b}+N_0} \qquad (8\text{-}21)$$

式中，$\overline{E_b}=E(\alpha^2)E_b$，λ=$KL_p$-1。

式（8-19）所表示的误码率的平均值为[15]：

$$P(e)=P(\eta_0)=\frac{1}{2}\left\{1-\sqrt{\frac{\eta_0}{1+\eta_0}}\right\} \qquad (8\text{-}22)$$

相应的最大比值 RAKE 接收机的平均误码率为：

$$P_e = [P(\eta_0)]^L \sum_{k=0}^{L-1} \binom{L-1+k}{k} [1-P(\eta_0)]^k \qquad (8\text{-}23)$$

8.3.2 RAKE 接收机的 MATLAB 实现

一、系统描述

仿真中的 CDMA 系统仅涉及扩频调制、多径衰落信道、扩频解调模块，不包含信道编／解码、交织等部分，也即考虑 CDMA 系统扩频调制解调级别上的 RAKE 接收机的误比特性能。系统采用等效低通表示方式，其结构框图如下图所示：

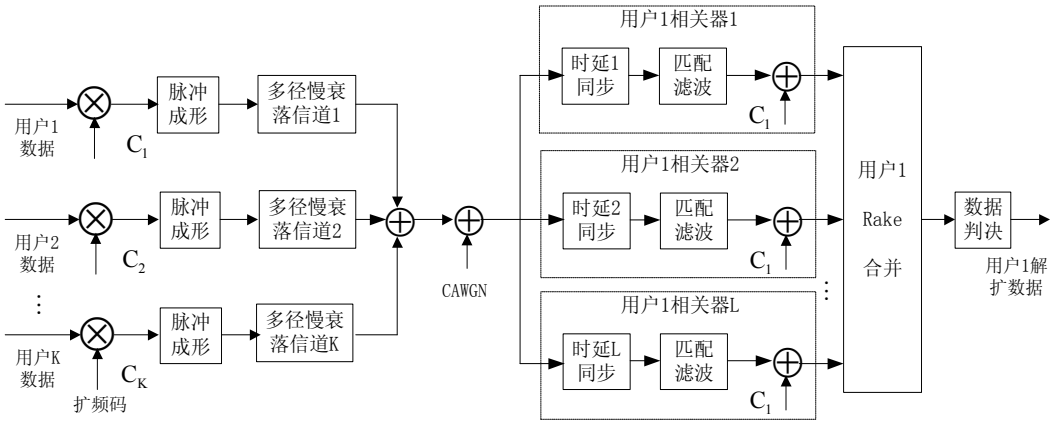


图 8-11 异步 DS-CDMA 的系统结构框图

1．系统参数描述

用户数：K=1；

用户扩频码：GOLD 码；

用户扩频码周期：可选，所附程序中取 $L_c = 31$；

成型滤波器/匹配滤波器：矩形波（$h(t) = \begin{cases} 1, & 0 \le t \le T_c \\ 0, & else \end{cases}$）；

信道模型：抽头延迟线模型；

信道噪声：加性高斯白噪声（AWGN）；

RAKE 合并：最大比值合并（MRC）。

设用户和基站之间的信道是慢变的瑞利多径衰落信道。多径数 L 为固定量或 $[1, L_{\max}]$

之间变化的随机变量。多径时延也是独立的，且服从$[0,T_b]$内的均匀分布。多径相位为$[0,2\pi]$内均匀分布的独立随机变量。在假设 RAKE 接收机中的信道估计单元对时延和相位的估计都是准确的情况下，我们可以仅考虑加性高斯噪声和瑞利衰落对 RAKE 接收机接收性能的影响。

2．噪声的产生

由式（8-21）知，η_0 和 N_0 是一一对应的关系，仿真中为了计算方便我们令 $A=1$，$T_b=1$，$E(\alpha^2)=1$，则信道中的高斯白噪声的单边功率谱密度为：

$$N_0=\frac{1}{\eta_0}-\frac{1}{2}\lambda e^2 \qquad (8\text{-}24)$$

在接收端，噪声 $n(t)$ 与载波 $\cos(\omega_c t+\varphi)$ 相乘，其单边功率谱密度变为 $N_0/2$，双边功率谱密度即为 $N_0/4$。再经过匹配滤波器，噪声均值仍为零但方差会发生改变：

$$\begin{aligned}\sigma_n^2&=\frac{1}{2\pi}\int_{-\infty}^{+\infty}(N_0/4)\left|H(j\omega)\right|^2 d\omega\\&=\int_{-\infty}^{\infty}(N_0/4)\left|h(t)\right|^2 dt\\&=N_0T_c/4\end{aligned} \qquad (8\text{-}25)$$

其中匹配滤波器的等效带宽为 $B=T_c/2$。仿真中，我们让信号在经过匹配滤波器之后再叠加上均值为零、方差为 σ_n^2 的高斯噪声以实现噪声对 RAKE 接收机的影响。

3． 瑞利衰落的产生

在前面计算噪声的功率谱密度时我们令 $E(\alpha^2)=1$，因为 α 是服从瑞利分布的，其均值和方差分别为 $\sqrt{\frac{\pi}{2}}\beta$ 和 $\frac{4-\pi}{2}\beta^2$，所以可以推出瑞利衰落参数 $\beta=1/\sqrt{2}$。我们用参数为 β 的 MATLAB 命令 raylrnd 即可生成所需的瑞利衰落变量，并将其作为乘性因子乘到 BPSK 信号上。

二、仿真结果和分析

图 8-12 为 RAKE 接收机误码率的理论曲线图，图 8-13 为仿真曲线图，其中横坐标为信噪比，系指信号功率与噪声和干扰功率之比。比较仿真图和理论图中的曲线我们可以看出，仿真的误码率曲线与理论值符合得比较好。

由图 8-12 和图 8-13 可以看出， RAKE 分集接收能有效地减少多径衰落的影响，降低

误比特率；单用户检测中分集重数 L 不超过路径总数时，分集重数 L 越大越好。

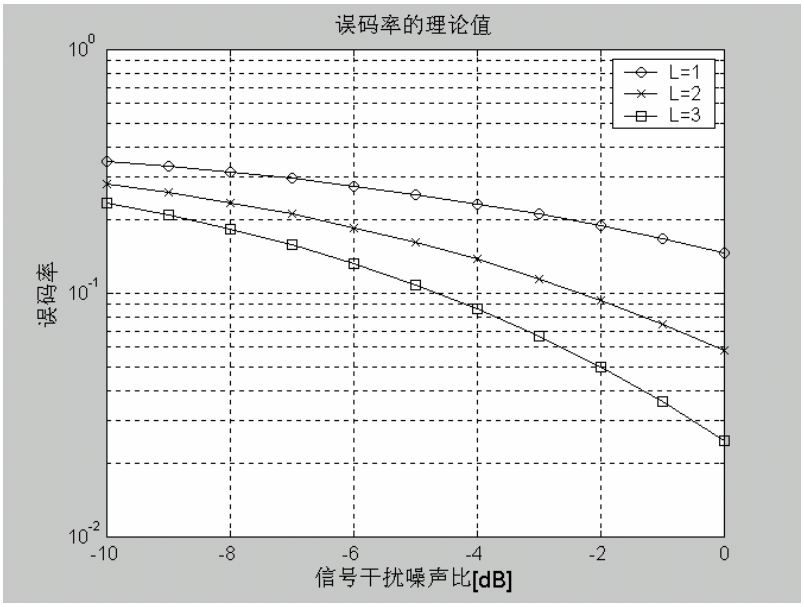


图 8-12 RAKE 接收机误码率的理论曲线图

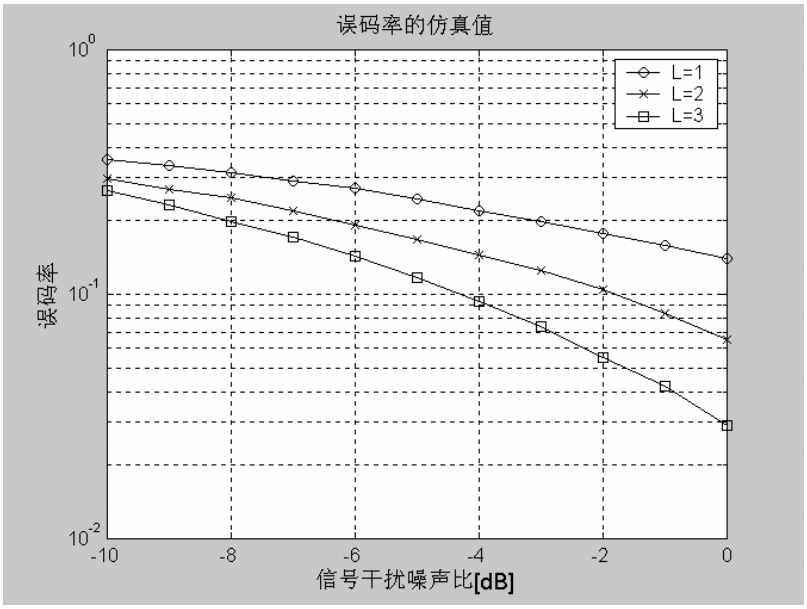


图 8-13 RAKE 接收机误码率的仿真曲线图

三、MATLAB 程序说明

MATLAB 源程序包含四个编写的 m 文件：

①．主程序：用于仿真 RAKE 接收机的误比特率。

②．子程序 cdma_codes.m ：用于产生用户扩频序列，可以生成 Gold 码、随机码和正

交 Walsh 序列。

③．子程序 Goldlow.m ：周期小于等于 127 的扩频序列生成程序。

④．子程序 Periodshift.m ：矩阵循环平移函数

具体程序清单如下：

Matlab **仿真主程序**

```
%                                          CDMA Rake接收机误码率仿真
%    系统描述
%    异步DS-CDMA系统
%    多径慢衰落信道
%    矩形码片成型滤波器
%    Rake接收: 最大比值合并
clear;clc;%close all;
timeofprog=cputime;

K=1; %用户数
Lc=31;%扩频码长度
T=1;Tc=T/Lc;%信号比特和扩频码周期
code_type=2;   %0 =  随机  1 = hadamard码  2 = gold码
code_seed=rand('state');
codes=(cdma_codes(K,0,code_type,Lc,code_seed))';
Q=10;%  矩形码片成型滤波器过抽样因子
Mpmax=5;%最大多径数
Mps=Mpmax*ones(K,1);%每个用户的多径数

%随机产生的时延，每条路径可分离
while (1)
    TausOri=rand(K,Mpmax)*Lc;
    TausOri=sort(TausOri,2);
    Tausdelta=TausOri(:,2:Mpmax)-TausOri(:,1:Mpmax-1);
    TuasSTD=sum(sum(Tausdelta>ones(K,Mpmax-1),1),2);
    if TuasSTD==K*(Mpmax-1)
        break
    end
end
Taus=floor(TausOri*Q)/Q;
TausSam=Taus*Q;%将时延转换为相应的抽样数

%仿真数据存储
```

```
filename='CDMARakedatapp.m';
fid=fopen(filename,'a+');
fprintf(fid,'%s ','???SimuPara: ');
SimuPara=[Lc,Taus];
for dataID=1:length(SimuPara)
      fprintf(fid,'%f    ',SimuPara(dataID));
end
fprintf(fid,'\n');
fclose(fid);

%设置信噪比SNRdBs
SNRdBs=-10:0;
nSNRdBs=length(SNRdBs);
for SNRID=1:nSNRdBs
      SNRdB=SNRdBs(SNRID);
      %AWGN方差的产生
      N0=1/(10^(SNRdB/10))-(K*Mps(K)-1)*1/(3*N);
      %AWGN的单边功率谱密度
     Bandwidth=Tc/2; %匹配滤波器等效带宽
     sigma2=N0*Bandwidth/2 %经过匹配滤波器后AWGN的方差
      sigma=sqrt(sigma2);

      nbitsAll=0; %发送总比特数
     nerrorbits=zeros(Mps(1),1); %错误比特数
      times=0; %循环次数
   while (1)
      N=1000;%一次发送的比特数
      bits=2*rem(unidrnd(2,K,N),2)-1;%BPSK信号
      Signal=zeros(1,(N+1)*Lc*Q);%不包括AWGN
      for k=1:K
          bitsChip=kron(bits(k,:),codes(k,:));%bits -->chips
          ChipsSam=kron(bitsChip,ones(1,Q));%chips --> samples
          for pathID=1:Mps(k)
               Fadings(pathID,:)=kron(raylrnd(1/(sqrt(2)),1,N),ones(1,Lc*Q));
               %产生瑞利衰落振幅
               TauSam=round(TausSam(k,pathID));
               Signal(TauSam+1:TauSam+N*Lc*Q)=Signal(TauSam+1:TauSam+N*Lc*Q)+...
                                                    Fadings(pathID,:).*ChipsSam;
          end
```

```
    end

    % 第一个用户的Rake接收
    % 假设时延已经准确估计
    TausSamEst=TausSam;
    Rakebranch=zeros(Mps(1),N);
    for pathID=1:Mps(1)
        TauSam=round(TausSamEst(1,pathID));

RakeChipMF=1/2*sum(reshape(Signal(TauSam+1:TauSam+N*Lc*Q),Q,N*Lc),1)*(Tc/Q);
        %解调后的信号(振幅为解调前的1/2)再经过匹配滤波器
        Noise=sigma*randn(1,N*Lc);
        RakeChipMF=RakeChipMF+Noise;%加上噪声
        Rakebranch(pathID,:)=codes(1,:)*reshape(RakeChipMF,Lc,N)*Tc;
        %解扩
    end
    %最大比值合并
    Rakebranchpower=Rakebranch.^2;
    for pathID=1:Mps(K)
        power=sum(Rakebranchpower(1:pathID,:),1);
        for temp=1:pathID
            MRCPara(temp,:)=Rakebranchpower(temp,:)./power;
            %每条路径的最大比值合并系数
        end
        orderID=pathID;
        RakeMRC(orderID,:)=sum(MRCPara(1:pathID,:).*Rakebranch(1:pathID,:),1);
    end
    %误码率计算
    RakeRcvSign=sign(RakeMRC);
    nbitsAll=nbitsAll+N;
    nerrorbits=nerrorbits+sum(abs(sign(RakeRcvSign-kron(bits(1,:),ones(Mps(K),1)))),2);
    BER=nerrorbits/nbitsAll;
    [SNRdB,nbitsAll/N,nerrorbits']
    %输出数据存储
    fid=fopen(filename,'a+');
    outdata=[SNRdB,nbitsAll,nerrorbits',BER'];
    for dataID=1:length(outdata)
        fprintf(fid,'%f    ',outdata(dataID));
    end
```

```
        fprintf(fid,'\n');
        fclose(fid);
      times=times+1;
      %跳出WHILE循环的条件
      if nerrorbits(Mps(1))>=100 | times>=10
            break
        end
    end
end
timeofprog=cputime-timeofprog
display('Program exit!!!');
```

gold序列产生器

```
% L=15,31,63,127
function [codes]=goldLow(L,Numcodes)
n=log2(L+1);
if n==4
    f=[1,0,0,1;1,1,0,0];%2*4%生成多项式系数
elseif n==5
    f=[1,0,0,1,0;1,1,1,1,0;1,1,0,1,1;1,0,1,0,0;1,0,1,1,1;1,1,1,0,1];%6*5
elseif n==6
    f=[1,0,0,0,0,1;1,1,0,0,1,1;1,0,1,1,0,1;...
            1,1,0,0,0,0;1,1,1,0,0,1;1,1,0,1,1,0]; %6*6
elseif n==7
    f=[1,1,0,0,1,0,1;1,1,1,0,1,1,1;1,0,0,0,0,0,1;1,0,1,0,0,1,1;...
            1,1,1,0,0,1,0;1,1,1,1,0,1,1;1,1,0,0,0,0,0;1,1,0,1,0,0,1];%8*7
else
    display('error! L is an invalid number');
    break;
end
fsize=size(f);
nrow=fsize(1);
m=ones(nrow,L+n-1);
for i=1:L-1
    mf=m(:,i:i+n-1);
    m(:,i+n)=diag(rem(mf*f',2));
end

pairs=(nrow-1)*nrow/2;
```

```
Sumcodes=pairs*L+nrow;
if Numcodes>Sumcodes
    display('error! The needed number of codes is too large. Propose to increase L!');
    break;
end
c=zeros(Numcodes,L);
Sum=0;
for ii=1:nrow-1
    for jj=ii+1:nrow
        for j=1:L
            Sum=Sum+1;
            c(Sum,:)=m(ii,1:L)+Periodshift(m(jj,1:L),0,j-1);
            if Sum==Numcodes
                c=rem(c,2);
                codes=2*c(1:Numcodes,:)-1;
                break;
            end
        end
    if Sum==Numcodes break; end
    end
    if Sum==Numcodes break; end
end
if Sum<Numcodes
    c(Sumcodes-nrow+1:Sumcodes,:)=m(:,1:L);
    c=rem(c,2);
    codes=2*c(1:Numcodes,:)-1;
end
codes=codes';
```

CDMA扩频码产生器

```
%       G : 扩频增益
%       K_in : 小区内用户数
%       K_out : 小区外用户数
%       code_type : 0 = random 1 = hadamard 2 = gold
%       code_seed : seed for random gen of codes
function [C]=cdma_codes(K_in,K_out,code_type,G,code_seed)
K=K_in+K_out;
c=zeros(G,K);
switch code_type
```

```
    case 0,     %random
        rand('state',code_seed);
        c=2*round(rand(G,K))-1;
    case 1,     %hadamard
        if rem(G,4)
           error('Spreading gain must be modulo 4 for hadamard codes');
        else
           codes=hadamard(G);
           if K_in<=G
              c=codes(:,1:K_in);
           else
           for i=1:floor(K_in/G)
                 c(:,(i-1)*G+1:i*G)=codes(:,1:G)
              end;
              c(:,floor(K_in/G)*G+1:K_in)=codes(:,1:K_in-floor(K_in/G)*G);
           end;
           rand('state',code_seed);
           c=[c codes(:,ceil(G*rand(1,K_out)))];
        end;
    case 2,     %gold
        if G<=127
            codes=goldlow(G,K_in+K_out);
        else
           n=log2(G+1);
           codes=gold(n,G);
        end;
        if K_in<=G
            c=codes(:,1:K_in);
        else
            for i=1:floor(K_in/G)
                c(:,(i-1)*G+1:i*G)=codes(:,1:G)
            end;
            c(:,floor(K_in/G)*G+1:K_in)=codes(:,1:K_in-floor(K_in/G)*G);
        end;
       rand('state',code_seed);
       c=[c codes(:,ceil(G*rand(1,K_out)))];
     otherwise
        error(['Code_type ', num2str(code_type),' is invalid']);
end;
```

```
for i=1:K
    C(:,i)=c(:,i);%/norm(c(:,i));
end
```

矩阵循环移位函数

```
function f=Periodshift(x,row,column)
%row—下移, column—右移
xsize=size(x);
f=x;
remrow=rem(row,xsize(1));
remcolumn=rem(column,xsize(2));
if   remrow>0
    f(1:remrow,:)=x(xsize(1)-remrow+1:xsize(1),:);
    f(remrow+1:xsize(1),:)=x(1:xsize(1)-remrow,:);
elseif remrow<0
    f(xsize(1)+remrow+1:xsize(1),:)=x(1:-remrow,:);
    f(1:xsize(1)+remrow,:)=x(-remrow+1:xsize(1),:);
end
x=f;
if remcolumn>0
    f(:,1:remcolumn)=x(:,xsize(2)-remcolumn+1:xsize(2));
    f(:,remcolumn+1:xsize(2))=x(:,1:xsize(2)-remcolumn);
elseif remcolumn<0
    f(:,xsize(2)+remcolumn+1:xsize(2))=x(:,1:-remcolumn);
    f(:,1:xsize(2)+remcolumn)=x(:,-remcolumn+1:xsize(2));
end
```

8.3.3 RAKE 接收的 DSP 实现

如前所述，在实际应用中，RAKE 接收往往与其它技术综合，形成了以 RAKE 接收为核心的复杂接收机结构。此时，考虑到实时性，该接收机要用专用集成电路实现。

下面给出了图 8-4 所示的 RAKE 接收机的 DSPC54 实现程序，其中接收机采用长度为 31 的 Gold 码作为扩频码，不考虑载波调制，完全同步，进行 4 路径分集的等增益合并。

```
*******************************************************************************************
                              [8.2 C54 实现传统 RAKE 接收机的程序]
*******************************************************************************************
.INCLUDE              "init_54x.ASM"
```

```
REAL    EQU   100H
IN      EQU   0H
OUT     EQU   1H

MAIN    STM   #REAL,AR3
        NOP
        NOP

        LD    IN,A              ；输入数据
        STL   A,*AR3
        STM   #REAL+30,AR3
        NOP
        RPTZ   A,#30
        MAC   *AR3-,#PN,A       ；相关器 1
        NOP
        STM   #REAL+31,AR3
        NOP
        RPTZ  A,#30
        MAC   *AR3-,#PN,A       ；相关器 2
        NOP
        STM   #REAL+32,AR3
        RPTZ  A,#30
        MAC   *AR3-,#PN,A       ；相关器 3
        NOP
        STM   #REAL+33,AR3
        RPTZ  A,#30
        MACD  *AR3-,#PN,A       ；相关器 4
        NOP
        STH   A,OUT             ；等增益合并

        STM   #REAL+3,AR3       ；数据移位
        RPT   #3
        MAC   *AR3-,#PN,A
        NOP

        B   MAIN

PN      .WORD  7FFFH,7FFFH,7FFFH,7FFFH,7FFFH,7FFFH,7FFFH,7FFFH   ；GOLD 码
```

```
.WORD   8001H,7FFFH,7FFFH,8001H,7FFFH,8001H,7FFFH,7FFFH
.WORD   8001H,7FFFH,7FFFH,8001H,8001H,8001H,8001H,7FFFH
.WORD   8001H,7FFFH,8001H,7FFFH,8001H,8001H,7FFFH
```

8.4 交织器的 DSP 实现

交织可以在不附加任何开销的情况下，使数字通信系统获得时间分集。因此，在第二代数字蜂窝移动通信中，交织成为了极其有用的一项技术。

交织器的作用是把码字顺序相关的比特（称为源比特）非相关化。如图 8-14 所示，假定某消息由若干帧组成，每帧含 3 比特。交织时，分别由 3 个消息分组的第 1 比特组成新的第 1 帧，第 2 比特组成新的第 2 帧，第 3 比特组成新的第 3 帧，然后依次传送。在传输期间，帧 2 丢失，若没有交织，则会丢失某一整个消息分组，而采用交织后，仅每个消息分组的第 2 比特丢失，则利用纠错编码技术可以恢复出全部分组中的消息。可见，图 8-13 所示的交织就是把码字的 i 个比特分散到 j 个帧中（即不同的时间段中），以改变源比特的邻近相关特性，显然帧值 j 越大，传输特性越好，但传输时延也越大。

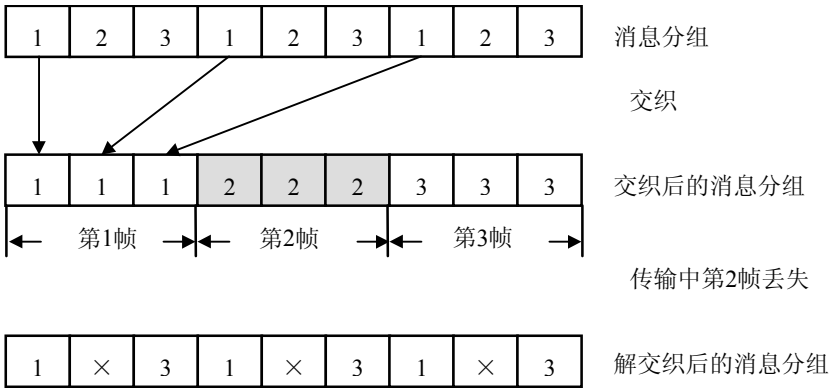


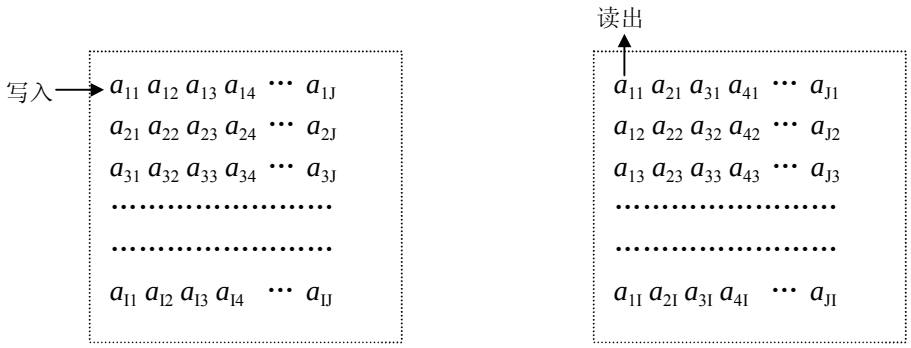
图 8-14 交织器的基本原理

信道编码（纠错编码）仅在检测和校正单个差错和不太长的差错串时才有效，难以抗突发持续较长的深衰落和突发干扰。而使用交织可使数据经无线信道后发生的差错串长度变短，但无法纠错。所以，交织器一般与信道编码联合使用。显而易见，若交织器设计良好，则错误将会随机分布，用编码技术就更容易纠正。

最常用的交织是块交织，它在数据分块或分帧的情况下使用。例如在 GSM 系统中，块交织器的输入码流是 20ms 的帧，每帧 456 比特。两个话音帧进行块间交织，每两帧（40ms）共 912 比特按每行 8 位写入，共写入 114 行，记 8×114=912 比特。输出时按列读出，每次读出 114 比特，恰好相当于 GSM 话音帧的一个 TDMA 时隙。也就是说，将 912 个符号交织后分散到 8 个 TDMA 帧的时隙中来传输。若传输中受到突发性干扰，丢失一个时隙的脉冲串，经过解交织后，则将突发差错变成随机差错，随后用纠错编码技术纠错。本节主要讨论块交织的实现方法及 MATLAB 实现、DSP 实现。

8.4.1 块交织的实现方法

一个（I，J）的块交织器可以看成是一个 I 行 J 列的存储矩阵。数据{a_{11} a_{12} a_{13} a_{14} … a_{1J} a_{21} a_{22} a_{23} a_{24} … a_{2J} a_{31} a_{32} a_{33} a_{34} …a_{3J} ……a_{I1} a_{I2} a_{I3} a_{I4} … a_{IJ}}按行写入，按列读出，如图 8-15 所示。连续的数据处理要有两个矩阵，其中一个用于数据写入，另一个用于数据读出。解交织也要有两个矩阵，用于反交织处理。



（a）按行写　　（b）按列读

图 8-15 （I，J）块交织器的读写示意图

一、块交织器性能的主要参数

交织器的性能主要由最小间隔 F、延时 D 和存储单元数 N 三个参数描述，其中最小间隔 F 是指突发连续错误分布的最小距离，它一般由突发长度确定；延时 D 表示交织和解交织时所带来的额外处理时间；存储单元数 N 表示交织过程所需的存放数据的单元数目。对于图 8-15 所示的块交织器，有：

$$F = \begin{cases} I & H \le J \\ 1 & H > J \end{cases} \tag{8-26}$$

其中 H 为突发错误的长度。S=1 对应于突发长度与序列长度相等的情况，此时无论如何排列，错误长度总是相互紧挨着的。

设交织和解交织过程中，每个元素所需的额外读/写操作时间为一个单位时间，则交织延时在发送端是 IJ，在接收端也是 IJ。所以总延时为：

$$D=2IJ \tag{8-27}$$

为了保证连续的操作，需要两个矩阵。设一个矩阵元素需要一个存储单元，则所需的存储单元数目为：

$$N=2IJ \tag{8-28}$$

显然，最小间隔 F 越大越好，延时 D 和存储单元数 N 越小越好。但实际上，总是受到一定的限制，不同系统有不同的要求。例如，对于无线话音通信，交织器的延时应限制在 40ms 以内，否则，计入传播、信道编码、语音编码等延时后，恢复出的语音人耳听起来不舒服。

二、块交织器的实现算法

为了增加随机性，实用中有时采用与图 8-15 不同的读出方法，即非传统的交织方法来

实现交织，如 IS-95 系统中的同步信道使用了“比特反转”方法。一般而言，非传统的交织中的延时和存储要求与图 8-15 所示的传统交织器的延时和存储要求相同，但最小间隔性能得到改善。非传统的交织有多种方法，一种块交织的具体实现算法如下：

$$i(B,j) = c(n,k); \quad k=0,1,...,B_s-1;\ n=0,1,...,N,N+1,...;$$

$$B=B_0+(M/2) * n+(k \bmod (M/2));$$

$$j=4 * (2 * (X \bmod (B_s/M)) + ((k \bmod M)/(M/2))),$$

$$X=a * (k/M) – b * (k \bmod M);$$

其中n是需要进行交织的数据块序号，B是交织完成后待发数据的子块序号，B_0是交织完成后待发数据的子块序号的初始偏移量。M是子块个数，当M=8时，每增加1与此相对应B将增加8。k是待交织数据块内比特的序号，a和b是优化交织增益的参数，mod()表示求模运算。

假定交织参数B_s =328，B_0 =0，M=8，a=8，b=6，则每来一块待交织数据，经交织后都会产生8个子块。由此可将这8个子块作为一个整体考虑，按照子块序号0~7顺序排列，如图8-16表示。

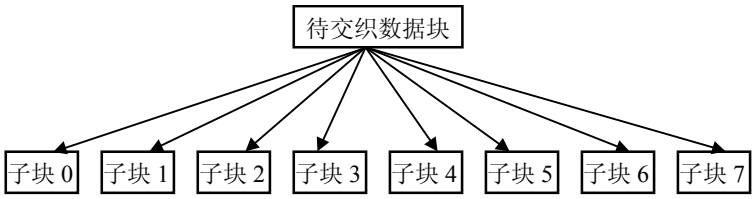


图8-16 交织块之间的关系

每 1 子块存放的数据是 328/8=41 比特，当将第 0 子块的数据标识为比特 0~比特 40，第 1 子块的数据标识为比特 41~比特 81，依次类推，第 7 子块的数据标识为比特 287~比特 327 后，我们可以将这些子块作为一个整体考虑。若交织后的第 0 比特对应交织前的第“w_0”比特，第 1 比特对应交织前的“w_1”比特”，……，则通过编程计算得出的表格如图 8-17 所示。

w_0	w_1	……	w_{327}

图 8-17 交织前后数据比特的对应表格

该表格在内存中共占用 328 字。

8.4.2 块交织的 MATLAB 实现

一、MATLAB程序

根据块交织器的实现算法编写的MATLAB程序如下，其中输入数据和输出数据分别用41行8列的矩阵表示。

****

[8.3 MATLAB 实现数据交织的程序**]**

```
**********************************************************************************
%交织

Bs=328;                    %每个数据块的数据长度
M=8;                       %子块个数
a=8;b=6;                   %交织增益优化参数
n=1;                       %需要交织的数据块序号（个数）
k=zeros(1,Bs);
j=zeros(1,Bs);
B=zeros(1,8);
for i=1:328                                    %生成输入数据
    k(i)=i;
    Bo=0;                                      %子块序号的初始偏移量
    B(i)=Bo+(M/2)*n+mod(k(i),(M/2));
    X=a*(k(i)/M)-b*mod(k(i),M);
    j(i)=4*(2*mod(X,(Bs/M))+(mod(k(i),M)/(M/2)));    %位置标签
end
[Y,I]=sort(j);        %输出顺序
%将 k、j、I 的值以 41*8 的矩阵形式表示
X=reshape(k,8,41)';          %输入数据矩阵
Y=reshape(j,8,41)';          %输入数据在交织后的矩阵中的位置矩阵
Z=reshape(I,8,41)';          %输出矩阵
```

二、仿真结果

该交织器的输入数据、输出数据用矩阵表示分别如下所示。

输入矩阵 X =

1	2	3	4	5	6	7	8
9	10	11	12	13	14	15	16
17	18	19	20	21	22	23	24
25	26	27	28	29	30	31	32
33	34	35	36	37	38	39	40
41	42	43	44	45	46	47	48
49	50	51	52	53	54	55	56
57	58	59	60	61	62	63	64
65	66	67	68	69	70	71	72
73	74	75	76	77	78	79	80
81	82	83	84	85	86	87	88
89	90	91	92	93	94	95	96
97	98	99	100	101	102	103	104

105 106 107 108 109 110 111 112
113 114 115 116 117 118 119 120
121 122 123 124 125 126 127 128
129 130 131 132 133 134 135 136
137 138 139 140 141 142 143 144
145 146 147 148 149 150 151 152
153 154 155 156 157 158 159 160
161 162 163 164 165 166 167 168
169 170 171 172 173 174 175 176
177 178 179 180 181 182 183 184
185 186 187 188 189 190 191 192
193 194 195 196 197 198 199 200
201 202 203 204 205 206 207 208
209 210 211 212 213 214 215 216
217 218 219 220 221 222 223 224
225 226 227 228 229 230 231 232
233 234 235 236 237 238 239 240
241 242 243 244 245 246 247 248
249 250 251 252 253 254 255 256
257 258 259 260 261 262 263 264
265 266 267 268 269 270 271 272
273 274 275 276 277 278 279 280
281 282 283 284 285 286 287 288
289 290 291 292 293 294 295 296
297 298 299 300 301 302 303 304
305 306 307 308 309 310 311 312
313 314 315 316 317 318 319 320
321 322 323 324 325 326 327 328

输出矩阵 Z =

328 129 258 59 188 317 118 247
288 89 218 19 148 277 78 207
248 49 178 307 108 237 38 167
208 9 138 267 68 197 326 127
168 297 98 227 28 157 286 87
128 257 58 187 316 117 246 47
88 217 18 147 276 77 206 7
48 177 306 107 236 37 166 295
8 137 266 67 196 325 126 255

296	97	226	27	156	285	86	215
256	57	186	315	116	245	46	175
216	17	146	275	76	205	6	135
176	305	106	235	36	165	294	95
136	265	66	195	324	125	254	55
96	225	26	155	284	85	214	15
56	185	314	115	244	45	174	303
16	145	274	75	204	5	134	263
304	105	234	35	164	293	94	223
264	65	194	323	124	253	54	183
224	25	154	283	84	213	14	143
184	313	114	243	44	173	302	103
144	273	74	203	4	133	262	63
104	233	34	163	292	93	222	23
64	193	322	123	252	53	182	311
24	153	282	83	212	13	142	271
312	113	242	43	172	301	102	231
272	73	202	3	132	261	62	191
232	33	162	291	92	221	22	151
192	321	122	251	52	181	310	111
152	281	82	211	12	141	270	71
112	241	42	171	300	101	230	31
72	201	2	131	260	61	190	319
32	161	290	91	220	21	150	279
320	121	250	51	180	309	110	239
280	81	210	11	140	269	70	199
240	41	170	299	100	229	30	159
200	1	130	259	60	189	318	119
160	289	90	219	20	149	278	79
120	249	50	179	308	109	238	39
80	209	10	139	268	69	198	327
40	169	298	99	228	29	158	287

8.4.3 块交织的 DSP 实现

具体实现时，首先用高级编程语言计算得出图 8-17 所示的表格，然后再用 DSP 汇编语言实现。当采用矩阵方式表示时，用 C（或 MATLAB）程序计算得出每个输入矩阵元素在新矩阵中的位置标号，结果如下：

Y =

290	251	212	173	134	95	56	65
26	315	276	237	198	159	120	129
90	51	12	301	262	223	184	193
154	115	76	37	326	287	248	257
218	179	140	101	62	23	312	321
282	243	204	165	126	87	48	57
18	307	268	229	190	151	112	121
82	43	4	293	254	215	176	185
146	107	68	29	318	279	240	249
210	171	132	93	54	15	304	313
274	235	196	157	118	79	40	49
10	299	260	221	182	143	104	113
74	35	324	285	246	207	168	177
138	99	60	21	310	271	232	241
202	163	124	85	46	7	296	305
266	227	188	149	110	71	32	41
2	291	252	213	174	135	96	105
66	27	316	277	238	199	160	169
130	91	52	13	302	263	224	233
194	155	116	77	38	327	288	297
258	219	180	141	102	63	24	33
322	283	244	205	166	127	88	97
58	19	308	269	230	191	152	161
122	83	44	5	294	255	216	225
186	147	108	69	30	319	280	289
250	211	172	133	94	55	16	25
314	275	236	197	158	119	80	89
50	11	300	261	222	183	144	153
114	75	36	325	286	247	208	217
178	139	100	61	22	311	272	281
242	203	164	125	86	47	8	17
306	267	228	189	150	111	72	81
42	3	292	253	214	175	136	145
106	67	28	317	278	239	200	209
170	131	92	53	14	303	264	273
234	195	156	117	78	39	328	9
298	259	220	181	142	103	64	73
34	323	284	245	206	167	128	137

98	59	20	309	270	231	192	201
162	123	84	45	6	295	256	265
226	187	148	109	70	31	320	1

将 Y 矩阵排序即可得到输出矩阵 Z，也就是所需的交织表。例如输入矩阵 X 中的 1 号数据，由 Y 矩阵可知它在交织后的位置是 290，所以在输出矩阵 Z 中第 290 个元素是 1 号。依此原理编制的实现交织器交织过程的 C54 程序如下：

```
-----------------------------------------------------------------
[8.4  交织器交织过程的 DSP 实现程序]
;----------------------------------------------------------------;
;输入参数：                                                       ;
;ar0 -->指向输入缓冲起始地址的指针，也就是待交织的数据的首地址   ;
;ar1 -->指向交织表起始地址的指针                                  ;
;ar2 -->指向输出缓冲起始地址的指针，也就是交织完的数据的地址     ;
;ar3 -->待交织数据的长度，以比特表示                             ;
;----------------------------------------------------------------
;uninter_word -->待交织的数据的长度，以字表示                    ;
;inter_bit      -->交织数据的比特位置                            ;
;loc_bit         -->待交织数据的比特位置                         ;
;loc_word        -->待交织数据的字的位置                         ;
;----------------------------------------------------------------;

          .def  inter_R

inter_R:  ldm  ar3,a                  ;待交织数据的长度
     sftl  a,-4                  ;长度/16 得到待交织数据所占的字数

     stl   a,uninter_word        ;存储待交织数据的长度（字的个数）
     stm   #15,BRC               ;循环的次数为 16
outstart:  st    #0000h,inter_bit       ;字内比特起始位置为 0
inerstart: rptb i_inner-1             ;循环开始
     ld    *ar1,-4,a         ;由交织表的数据得到此输出比特在待交织数据
                             ;中的 WORD 位置
     stl   a,loc_word        ;将此 WORD 位置保存起来
     ld    #15,b             ;由交织表的数据得到此输出比特在待交织数据
     and   *ar1+,b           ;中的 BIT 位置
     stl   b,loc_bit               ;将此 BIT 位置保存起来
     ld    #1,a
     rsbx c                        ;清 C
```

```
        rpt   loc_bit
        rol   a                 ;将 1<<loc_bit+1
        ror   a
        stlm a,ar4              ;保存 A 中的数据
        ldm  ar0,b                  ;保存 AR0 中的数据
        ldm  ar0,a              ;在输入缓冲区中寻找要交织的 BIT 的字的位置
        add  loc_word,a             ;ar0+loc_word
        stlm a,ar0
        ldm  ar4,a
        and  *ar0,a             ;a 中的第 loc_bit 个比特就是需要
                        ;交织到此比特位置的数据
        stlm b,ar0
             rsbx c
        rpt   loc_bit
        ror   a                     ;数据右移到 A 的最低位中
        rol   a
        rsbx c
        rpt   inter_bit
        rol   a                 ;A<<inter_bit,inter_bit 是输出缓冲当前字中
        ror   a                  ;要处理的比特位置

        ssbx c                      ;置 C
        ld    #0ffffh,16,b
        or    #0fffeh,b
        rpt   inter_bit
        rol   b
        ror   b
        and  *ar2,b             ;清除输出缓冲此比特位置原有的数据
        or    a,b               ;将 A 中的数据通过“或”操作写入输出 B 中
        stl   b,*ar2            ;将 B 中的数据写入输出缓冲区中
        addm      #0001h,inter_bit    ;输出缓冲中待操作的比特位置加 1
        nop
i_inner    mar  *ar2+                ;输出缓冲中待操作的字位置加 1
           ld     uninter_word,a
        sub    #1,a             ;待交织数据的字的长度减 1
        stl    a,uninter_word
        and    #0ffffh,a                ;判断整字的待交织数据是否交织完
        bc     outstart,aneq       ;没有交织完则继续交织，否则去判断剩下
                ;的数据够不够一个字
```

```
    ldm  ar3,a              ;如果待交织数据长度除以 16 所得余不为 0
    and  #15,a              ;的话，还要继续处理最后一个字的数据
    bc   endprog,aeq
    sub  #1,a
    stlm a,brc
    stm  #0000h,ar3
    B    inerstart
endprog: ret
```

程序中 ar0 指向待交织数据的输入缓冲地址，ar1 指向交织表，ar2 指向完成交织的数据的地址。ar1 每次加 1，对应于 ar2 所指字的比特位置 inter_bit 也加 1，指向的内容是 input buffer 内的地址偏移量，此偏移量指向的比特就是需要交织到 ar2 指向字的 inter_bit 比特。程序的主要结构相当于有两层循环，在外层循环中指针 ar2 每次加 1，对应内层循环执行 16 次，inter_bit 是 ar2 所指字内的比特位置在内层循环中每次加 1，回到外层循环后 inter_bit 被清零。

去交织是交织的逆过程，需要使用相同的交织表，程序的结构与上述的描述大致相同，但比特搬移方向相反，具体编程时可将上述程序略加修改即可。

第九章 软件无线电的 DSP 技术

软件无线电是在 20 世纪 90 年代初期提出来的一种新的无线通信系统体系结构。它的基本思想是以开放的、可扩展的、结构最简的硬件为通用平台，把尽可能多的通信功能用可升级、可替换的软件来实现。

9.1 软件无线电概述

软件无线电最初起源于军事通信。无线通信在现代军事通信中占据着极其重要的位置。当代无线通信系统很多，例如，卫星通信系统、蜂窝移动通信系统、无线寻呼系统、短波通信系统、微波通信系统，等等。各种无线通信系统的调制方式也很多，有 AM、FM、LSB、USB、ISB、FSK、PSK、MSK、GMSK、QAM，等等。其多址方式有：时分多址（TDMA）、频分多址（FDMA）和码分多址（CDMA）等。各种通信系统由于自身的特点而应用于不同场合。短波电台适合远距离传输，其所需的发射功率不大，传输的“中继系统”—电离层不会被摧毁；卫星通信能传播高质量信息，所能提供的频带很宽；微波通信抗干扰能力强，适合大容量的数据传输，但只能点与点之间传输。军用电台往往是根据某种特定的用途而设计的，功能单一，有些电台的基本结构相似，而信号特征差异很大。例如，工作的频段不同，有的在 HF 频段，有的在 VHF、UHF 频段，调制方式不同，波形结构不同，通信协议不同，数字信息的编码方式、加密方式不同，等等。电台之间的这些差异极大地限制了不同电台之间的互连互通。而且，由于不同频段的电台只能满足某些特定的要求，无法满足部队各种各样的军事需求，给协同作战带来了困难。

在格林纳达冲突中，美军各种通信设备的不兼容性暴露无遗。在海湾战争中，由于盟军联合作战，需要及时有效的通信联络，这大大增强了通信保障的复杂程度，不得不借助许多额外的无线电台，才得以保证高效的通信联络。

在民用通信中也存在互通性问题。如现有的陆地移动通信系统是以 AMPS、TACS、J-TACS、N-AMPS 标准为代表的第一代模拟移动通信系统和以 GSM 及 DCS1800/PCS1900、PDC、IS54、IS-95 标准为代表的第二代数字移动通信系统构成的。这些通信体系的制式、频率各不相同，不能互通、兼容，给跨国经商、旅游等活动带来极大的不便。这些问题在未来的无线通信领域也依然存在。例如，第三代移动通信系统标准就有好几种（如 W-CDMA、CDMA2000，TD-CDMA、SCDMA 等）。

为了解决互通性问题，各国军方进行了积极探索，努力使不同设备既能满足互通的要求，又能满足抗干扰、保密性好的要求；既能使通信设备跟上无线电飞速发展的步伐，又能延长设备的使用寿命。软件无线电为这些问题的解决提供了一种新的思路。

1992 年 5 月，在美国电信系统会议(IEEE National Telesystems Conference)上，MITRE 公司的 Joe Mitola 首次明确提出了软件无线电的概念，软件无线电的核心是将宽带 A/D、D/A 尽可能地靠近天线，用软件实现尽可能多的无线电功能。软件无线电里程碑性的文章

发表于 1995 年 5 月，Mitola 在文章中回顾了软件无线电不断发展的概念、结构、技术上所面临的挑战以及软件无线电所能带来的经济效益。此篇文章与其他一些文章一起构成了第一期关于软件无线电的专刊，包括软件无线电的许多键技术的研究成果，如高速 A/D/A 转换、DSP 处理的瓶颈问题，智能天线技术等等。在此专刊发表之后商用的软件无线电研究迅速升温。

1998 年 6 月，欧洲委员会与 SDR 论坛(Software-Defined Radio Forum，其正式名称为 MMITS 论坛：Modular Multifunction Transfer System Forum)联合主办了第一届软件无线电国际研讨会，会上指出可以使一个移动终端的结构适应全球由软件定义的移动通信标准。第一届亚洲研讨会也于 1998 年 4 月由日本的 Kieo 大学主办。

由于技术的变化和应用的扩展，有关软件无线电的概念、结构、实现、用途等都在发展之中，目前还很难给出一个严格而全面的定义。其中心思想是：构造一个具有标准化、模块化的通用硬件平台，使 A/D 和 D/A 尽量地接近天线，并通过软件加载实现各种无线通信功能的一种开放式体系结构。

软件无线电的主要特点可以归纳如下：

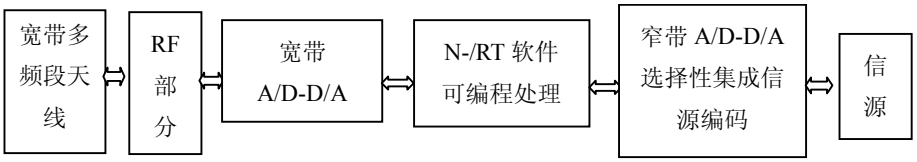
（1） **具有很强的灵活性** 软件无线电可以通过增加软件模块，很容易增加新的功能。可以与其他任何电台进行通信，并可以作为其他电台的射频中继。可以通过无线加载来改变软件模块或更新软件。

（2） **具有较强的开放性** 软件无线电由于采用了标准化、模块化结构，其硬件可以随着器件和技术的发展而更新或扩展，软件也可以随需要而不断升级。

软件无线电这一概念一经提出，就得到了全世界无线电领域的广泛关注。

9.2 软件无线电的基本结构

软件无线电的基本思想是以一个通用、标准、模块化的硬件平台为依托，用软件编程来实现无线电台的各种功能，从基于硬件、面向用途的电台设计方法中解放出来。功能的软件化实现势必要求减少功能单一，灵活差的硬件电路，尤其是减少模拟环节，把数字化处理(A/D 和 D/A 变换)尽量靠近天线。软件无线电强调体系结构的开放性和全面可编程性，通过软件的更新硬件的配置结构，实现新的功能。理想的软件无线电的组成结构如图 9-1 所示。



N-/RT=准实时和实时

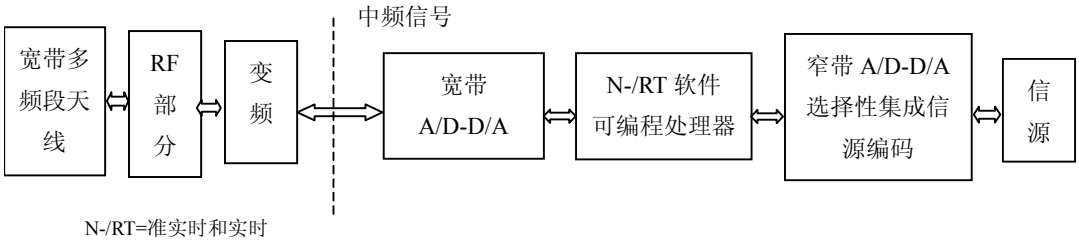
信源包括窄带业务与未来的宽带业务

图 9-1 理想软件无线电结构

软件无线电主要由天线、射频前端、宽带 A/D-D/A 转换器、通用和专用数字信号处理

器以及各种软件组成。

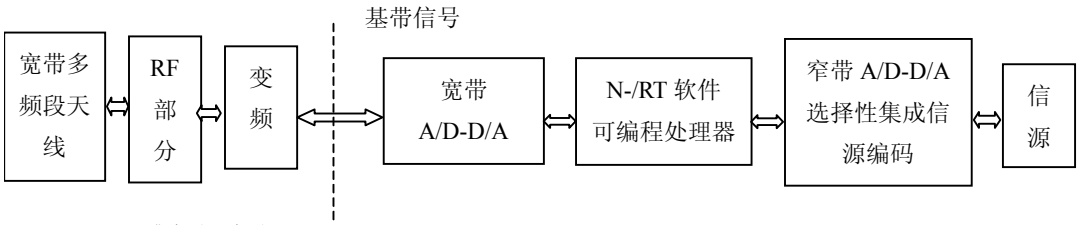
随着无线通信频段的升高，由于受硬件器件的制约，实现理想的软件无线电将更加困难。在实现理想软件无线电的过程中，将有两种演进结构如图 9-2、 图 9-3 所示。



N-/RT=准实时和实时

信源包括窄带业务与未来的宽带业务

图 9-2 中频数字化软件无线电结构



N-/RT=准实时和实时

信源包括窄带业务与未来的宽带业务

图 9-3 基带数字化软件无线电结构

目前 LF，HF 频段可以实现理想的软件无线电结构，而 VHF，UHF 以及更高的频段只能实现基带数字化或中频数字化。但随着硬件器件水平的提高，高频段的无线通信系统将逐步由基带数字化演进到理想的软件无线电结构。

软件无线电的硬件具有开放性，其硬件必将采用总线式的结构。总线式的软件无线电结构的发展是以美国的易通话（SPEAKeasy）计划为代表。易通话计划是下一代军用无线电的一项分阶段的技术基础研究与开发计划，该计划由空军罗姆实验室管理，三军和国防高级研究计划局资助。易通话是一项多期、联合军种技术计划，其目的之一是验证可编程波形、多频段、多工作方式具有开放结构的无线电台方案，之二是发展一种通用的软件结构，便于新波形的增加。

易通话第一阶段研究是从 1992 年到 1995 年，其频率范围从 2M~2GHz，覆盖了 HF、VHF、UHF 频段。这么宽的频带不能用一个宽带射频信道来实现，按目前可用技术，须分成三个频带。低频段是 2~30MHz，中频段是 30~400MHz，高频段是 400M~2GHz。实际可行演示中，仅仅实现了中频段。功能结构简图如图 9-4 所示。

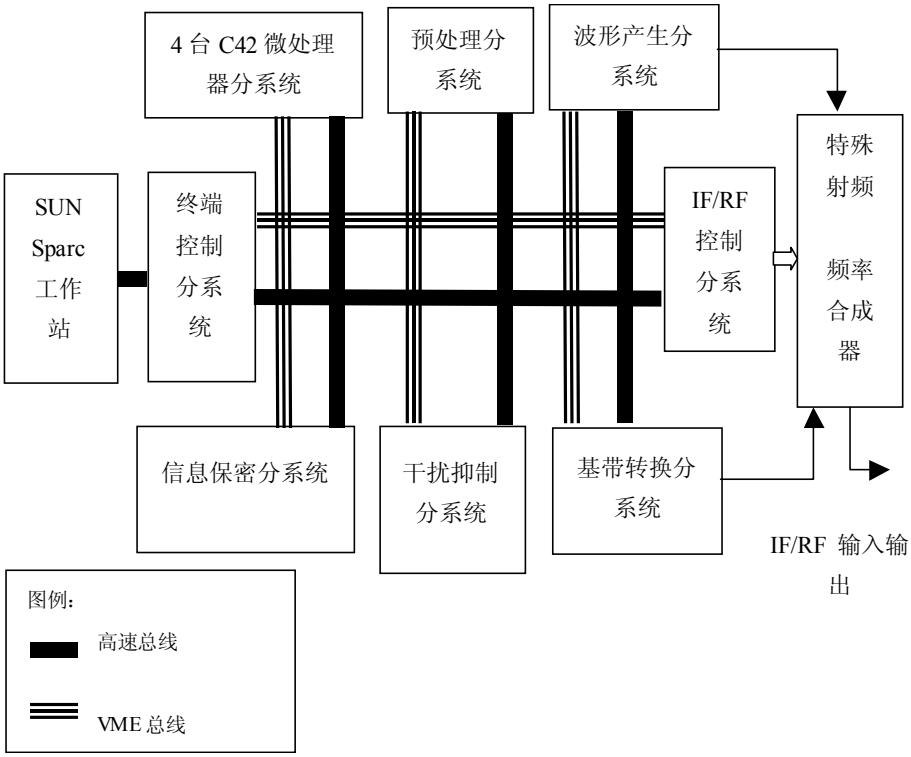


图 9-4　SPEAKEASY 第一期功能简图

易通话第一期工程高级开发模型由 VME 总线提供控制，由高速环总线提供数据传输。1995 年易通话电台参与了联合士兵协同演习（JWID-95），在演习中展示了良好的与标准电台互连操作的性能、网关功能及可编程能力。

易通话第一阶段计划获得了成功，而且也使易通话成为软件无线电在军事领域成功应用的典范。易通话第二阶段计划从 1995 年开始，其目标不仅仅是验证可编程波形、多频段、多工作方式的无线电台方案，更主要是研究为整个无线通信系统提供一个从输入输出到射频的开放的模块化可编程结构。为了降低无线通信系统生命周期更新的费用，第二阶段侧重设计商业成品的模块化与标准化。目前刚刚完成的第二阶段的易通话电台 70%是由商业成品模块组成，采用 PCI 总线，用 windows 95 界面的便携电脑作为用户界面，其工作频段为 20~400MHz，其功能方框图如图 9-4。由图 9-4 可知，易通话 II 由射频模块、调制解调模块、信息安全模块、网际互联模块、人机界面模块、控制模块、PCI 总线组成。PCI 总线由 PCI—I 与 PCI—II 两部分组成，这两部分是通过信息安全模块相连的，PCI—I 总线上的数据是不加密的，PCI—II 总线上的数据是经过加密（CP）模块处理的。在 1997 年 TF-XXI-AWE 演习中，模块化易通话被成功应用。易通话 II 展示了一个开放的、模块化的、标准的、可编程的系统结构，其优越性是明显的。

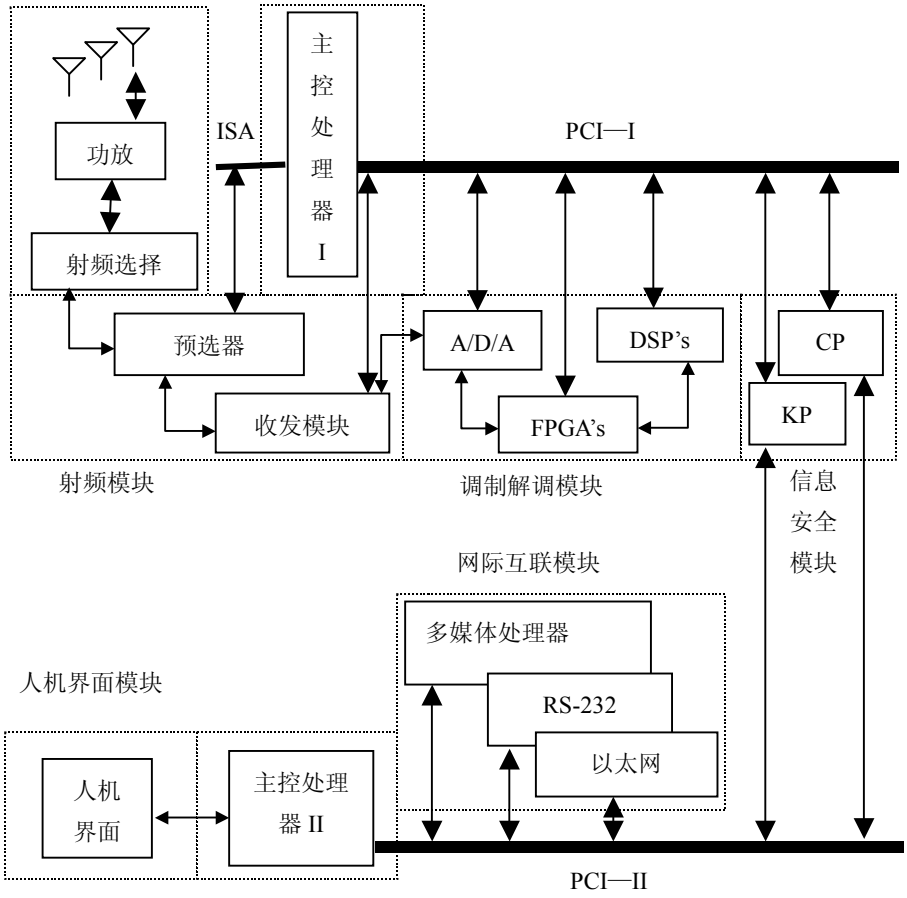


图 9-4 第二期 SPEAKeasy 功能简图

9.3 软件无线电的关键技术

软件无线电是近几年发展 新兴技术，对它的研究还处于起步阶段，许多技术问题需要解决。其中的关键技术有以下几个方面。

9.3.1 开放结构

文献[18]指出近几年的无线产业的增长主要应归功于采用了开放式标准。因此对军用软件无线电的系统结构的研究是很重要的。

开放的系统结构具有以下特征：

1、接口与协议是明确定义与广泛应用的；

2、使用的标准是标准化组织定义的或商业市场广泛采用的；

3、接口与协议是适应未来需要的。

开放系统结构的研究首先需要进行合适的模块划分，易通话 II 将系统划分为射频模

块、调制解调模块、信息安全模块、网际互联模块、人机界面模块与控制模块；其次接口需要进行明确定义，易通话模块是基于总线的。

当前，众多提出的软件无线电结构，缺乏通用的设计方法，许多实际设备只能适用于某些特定系统。弗吉尼亚工学院的 Srikathyayani Srikanteswara 提出一种标准化结构：分层无线电结构。该结构能够实现软件无线电的所有特性，并提供一种修改和更新系统的方法。

一、软件无线电硬件标准化

该结构采用硬件分页来实现。硬件分页（Hardware Paging）是指硬件模块在系统中进行页面调入和页面调出处理。分页的概念是从虚拟存储系统中引入的，类似于在虚拟存储系统中具有一个虚拟地址并且可以在实存储器和辅助存储器之间作为一个单位来传送的一种长度固定的(数据)块。分层结构有利于基于数据流的处理，总线既用来处理数据，也用来对信息进行编程。数据的前端添加字头，用来指示数据包的特性。基于数据流的处理简化了模块之间的接口，可以很方便的添加或替换模块。

分层无线电按照功能可以划分为三层，每一层对数据包字头添加或修改，将封装后的数据传递到下一层。当处理结束后，信息以相反的顺序传递。分层结构具体定义为：软件无线电接口（SRI）层，配置层和处理层。在各层中都有应用软件驻留，提供用户接口。三层结构采用基于数据流的处理方法实现。其分层结构如图 9-5 所示。

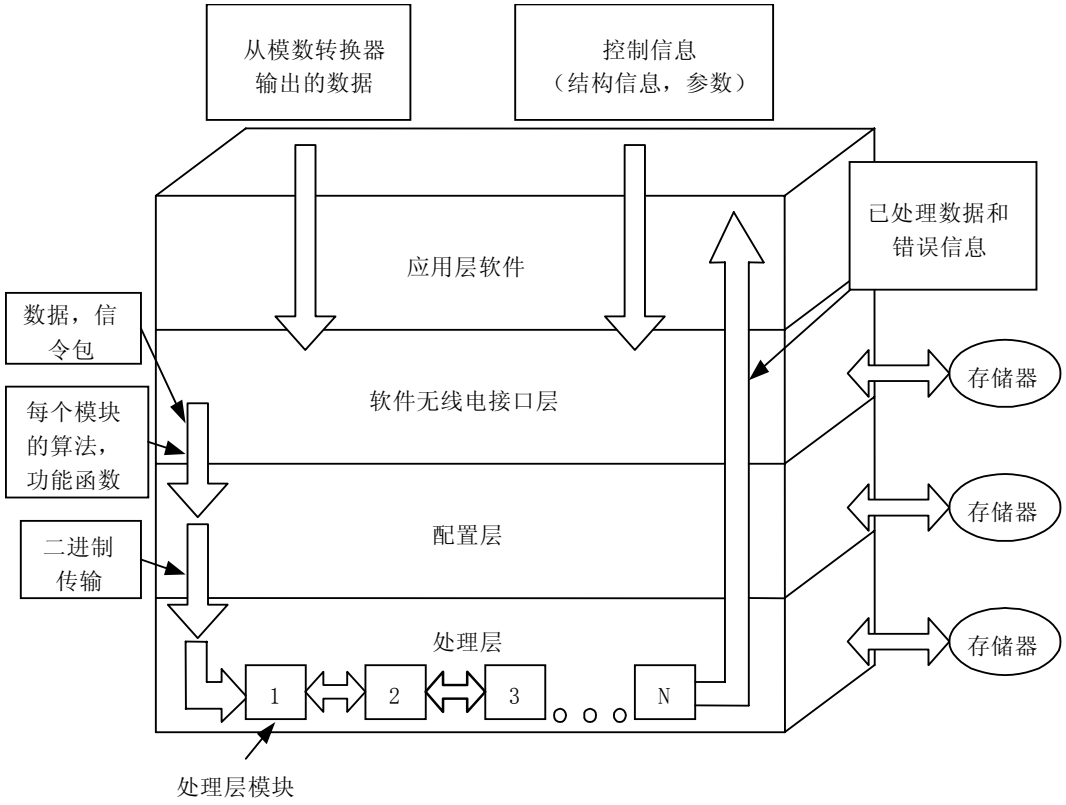


图 9-5 分层结构

1. 各层功能定义

SRI 层负责提供无线电硬件与外界的接口，调整送入和送出无线电系统的信息资源分

配，将各种需求按优先级排队，对送入信息进行打包封装，然后送到下一层。送入的信息包括数据、要求建立新系统的编程信息或对现有系统的修改信息等。

在本地存储器中，SRI 层设置了对各种无线电配置的系统级描述。每一个系统级描述包含实现系统所需的算法清单，这些算法是按照一定顺序排列的。事实上，SRI 层并不包含配置硬件所需的比特信息，它只包含算法所需的代码以及算法之间的组合次序。

在配置层的存储器中，存有设定处理层硬件组合需要的比特信息。配置层从 SRI 层发送的编程数据包中提取配置比特信息。

处理层包含一系列可配置模块，称为处理模块。该层是执行数据操作的模块，用来实现软件无线电的实际功能。每一个处理模块分为静态区和动态可配置区，如图 9-6 所示。通过由配置层发送的编程数据包，静态区对动态区按实际需要进行配置。

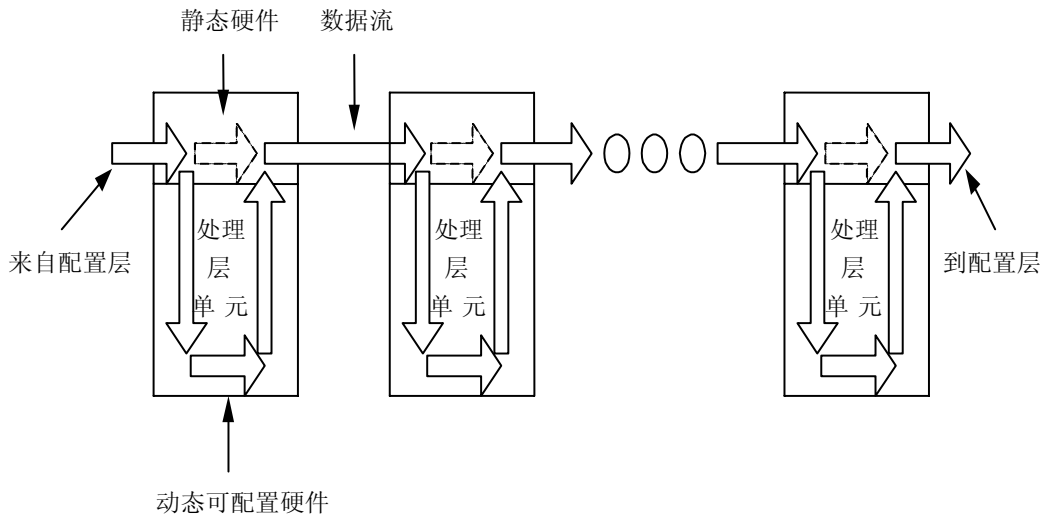


图 9-6 处理层的处理单元

一旦系统配置完毕，数据在 SRI 层进行编程封装，然后送到下一层，进行后续处理。当整个过程结束后，数据由配置层送回到 SRI 层，同时对数据进行拆分，最后输出到主控计算机上。

2. 各层之间数据流处理

该软件无线电结构建立在基于数据流处理的基础上。数据流是一个已知长度的数据包，包含配置信息或待处理数据。这些数据包在 SRI 层封装成形，添加字头信息，然后在低层对字头进行解释编译。

基于数据流处理提供了一种深度的管道操作，可以最大限度利用处理速率，同时保证一定的灵活性。整个算法相当于一个数据流，该数据流被逐渐分解为更小计算原语（对应处理模块），每一个处理模块执行完自身特定操作，就将数据和控制信息传递到下一个模块。整个过程如图 9-7 所示，类似于一套流水线操作。

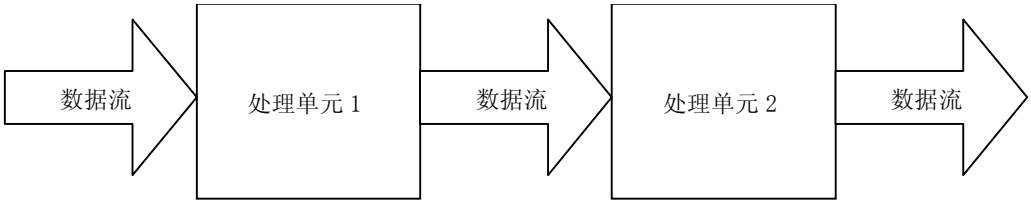


图 9-7 基于数据流的处理

由上可以看出，分层软件无线电结构是一种开放式标准结构。模块之间的接口十分简单，每个模块可以独立的接收和发送数据包，并进行打包或拆分。该分层结构对于目前的 FPGA 能够实现部分可配置，对于未来的计算机平台能够实现完全可配置。

二、软件无线电软件标准化

1. 结构特点

软件无线电的核心思想是“重配置性”（Re-configurable），主要体现在软件的可重用性，这是整个系统的一个重要指标及研究方向。目前国外正在研究软件无线电系统中软件的即插即用（Plug and Play）技术，并提出了基于 Java/CORBA 制定的软件无线电软件协议和相应技术标准。该标准化软件体系具有以下特点：

（1）分层的软件结构，把应用与底层硬件相分离；

（2）使用 CORBA 提供了一个分布式处理环境，保障实现软件的可移植性、可重用性和可扩展性；

（3）最大地利用商品标准和产品。

该软件体系中定义了一个运行环境 OE，该运行环境由总线层、操作系统、核心框架服务和基本应用、CORBA 中件服务和基本应用组成。实现具体应用的软件包括调制解调（负责各种信号处理）、链路网络协议、安全应用等。其软件体系结构如图 9-8 所示。

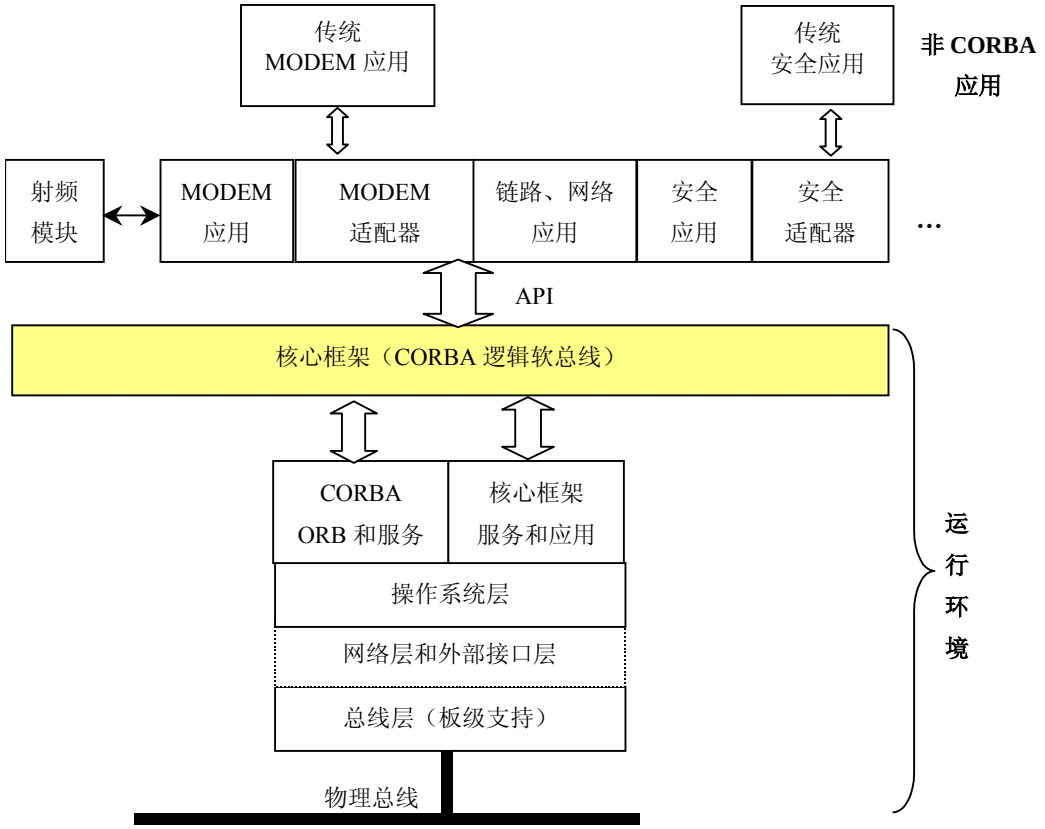


图 9-8　软件体系结构

2. 网络和外部接口层

该软件体系可采用现有商业化的组件来支持多个独特的外部接口和网络接口，可能的外部和网络物理接口包括 RS232、RS422、RS423、RS485、以太和 802.x。为了支持这些接口，可以采用各种低层网络协议，包括 PPP、SLIP、LAPx 和其它协议。

软件无线电的网络功能部分也在操作系统层实现。

3. 操作系统层

操作系统层提供一个实时的嵌入操作系统功能，提供对应用的多线程支持。体系规范要求为操作系统业务提供一个标准的操作系统接口，以便于应用的可移植性。

POSIX（可移植操作系统接口）是一个公认的工业标准。规范中的操作系统是 POSIX 兼容的。现有的实时操作系统很多是符合 POSIX 的，如 LynxOS、VxWorks 等。POSIX 程序语言可以是 C／C＋＋、Ada 或 Java。

4. CORBA 控件

CORBA 是核心框架中提供分布处理环境的消息传输技术。CORBA 协议提供了消息调度来处理传输消息所需的封装和握手。CORBA 是一种跨平台的框架，在使用分布处理时，可用来标准化客户/服务器操作。

分布处理是模块化软件无线电的一个基本特征，基于如下原因，该体系采用 CORBA 作为中件服务：

（1）CORBA 是一个已被广泛采用的开放的工业标准，现已存在许多实用的 CORBA 产品。
（2）CORBA 采用接口描述语言 IDL 来定义分布对象间的接口，通过特定语言的 IDL 编绎器形成头文件、开发包，使代码更可靠、易于维护。
（3）CORBA 通过对象请求代理（ORB）软件总线抽象化了硬件物理总线，从而使应用可运行在不同的物理总线上。
（4）CORBA 不依赖处理器类型、总线类型和操作系统，可跨平台通信。
（5）在 CORBA 中，客户软件和服务软件可采用不同的语言编写。
（6）已有商业产品提供对 TI 公司 DSP 的 RPC 支持，CORBA 从而可与 DSP 通信。

5. 核心框架

核心框架是软件体系中最重要的。它是开放的应用层接口和业务的基本集合，为应用开发者提供了底层硬件软件的抽象接口。它包括：
（1）基本应用接口：所有软件应用可使用。
（2）框架控制接口：提供系统控制。
（3）框架业务接口：支持核心和非核心应用（系统资源管理、注册、文件系统及管理等系统功能）。

6. 应用层

应用层执行用户通信功能，它包括调制解调级的数字信号处理、链路级的协议处理、网络级的协议处理、互连网络级的路由、外部 I/O 访问、安全和嵌入应用。应用需要使用核心框架的接口和服务。应用层可受限地对操作系统直接访问。在低于应用层的层中实现的网络功能，如商业化的 IP 网络层，由于其存在于操作系统的内核中，不再受限于体系规范中的约束。

7. 适配器 Adapter

适配器用于支持无 CORBA 能力的采用。适配器在一个实现中提供了无 CORBA 能力的组件或设备。适配器的概念是基于工业界公认的适配器设计模式。适配器对支持非 CORBA 的调制解调器、安全和主机处理采用特别有用。

9.3.2 宽带/多频段天线与 RF 模块

这是软件无线电不可替代的硬件出入口。 软件无线电对这部分的要求包括：天线能覆盖所有的工作频段，能用程序控制的方法对功能及参数进行设置。实现的技术包括：组合式多频段天线及智能化天线技术；模块化、通用化收发双工技术；多倍频程宽带低噪音放在大器方案等。

RF 部分包括预放大、功率输出。在软件无线电中，某些 RF 转换问题更加突出，其中包括对放大器的线性及接入频段的效率需求。

9.3.3 模数转换部分

软件无线电体系结构的一个重要特点是将 A/D 和 D/A 尽量靠近射频前端。为减少模

拟环节，在较高中频，乃至对射频信号直接进行数字化。这就要求 A/D 器件具有适中的采样速率和很高的工作带宽。为适应错综复杂的电磁环境，A/D 器件除了要有高速度、大带宽外，同时还需要大动态范围。软件无线电系统设计时，选择什么样的模数器件是很重要的，其依据是如下模数转换器的性能指标：

- 信噪比（SNR）

对于一满量程的正弦信号，SNR 可以准确表示为：

$$SNR = 6.02n + 1.76dB + 10\lg[f_s/2B]$$

式中，f_s 为采样率，B 为模拟信号带宽。式右边的第三项也称为处理增益，是一个正值，它表示信号带宽与 $0.5f_s$ 相差的程度所增加的信噪比。可以看出提高采样率，或者降低模拟信号带宽都可以改善模数转换器的信噪比。

- 转换灵敏度

假设一个 A/D 器件的输入电压范围为（－V，V），转换位数为 n，即它有 2^n 个量化电平，则它的量化电平为

$$\Delta V = 2V/2^n$$

ΔV 也可以称之为转换灵敏度。A/D 转换的位数越多，器件的电压输入范围越小，它的转换灵敏度越高。

- 无杂散动态范围（SFDR）

无杂散动态范围是指在第一奈奎斯特区内测得信号幅度的有效值与最大杂散分量有效值之比的分贝数。SFDR 允许人们评价一个 ADC 能够在非常大的信号存在的情况下，检测到有用小信号的能力，或者是评价一个 DAC 在输出有用信号时，对邻道造成的寄生干扰。是软件无线电中的一个重要指标。寄生信号是由于 A/D 的非线性引起的。

- 差分非线性

差分非线性误差(DNL)主要由于模数转换器本身的电路结构和制造工艺等原因，引起在量程中某些点的量化电压和标准的量化电压不一致而造成的。积分非线性误差（INL）是指 A/D 模拟前端、采样保持器的传递函数的非线性所造成的。INL 引起的各失真分量的幅度随输入信号的幅度变化。

- 互调失真

互调失真当两个正弦信号同时输入时，由于器件的非线性，将会产生许多失真产物。由于二阶互调产物容易被滤除。除非另有说明，一般情况下双音互调失真是指三阶产物引起的失真。

- 总谐波失真

由于 A/D 器件的非线性，使其输出的频谱中出现许多输出信号的高次谐波，这些高次谐波分量称为谐波失真分量。

- 全功率模拟输入带宽

当使用 ADC 做带通采样时，进入 ADC 的最大输入频率大于采样频率的一半，此时全功率模拟输入带宽是一个重要的指标。通常对全功率模拟输入带宽的定义是从 0 频率到某一个频率，在这个频率上，ADC 的输出比最大输出降低 3dB。

9.3.4 高速数字信号处理器

数字信号处理器是整个软件无线电方案的灵魂和核心所在。软件无线电的灵活性、开放性、兼容性等特点主要是通过以数字信号处理器为中心的通用硬件平台及 DSP 软件来实现的。从前端接收下来的信号，或将功放发射出去的信号都要经过数字信号处理器的处理：或进行频谱分析、信号解调、信号类型识别，或进行信号的数字上下变频，或进行各种各样的数字调制、数字滤波、比特流的编码、译码、同步信号的获取等等。软件无线电中的数字信号处理器除了能适应运算处理的高速度、高精度、大动态范围、大运算量外，它还应具有高效率的结构和指令集、较大的内存容量、较低的功耗等特点。

DSP 是软件无线电的核心，它要完成全部基带处理功能，如信号检测、同步获取、解调等等基本功能，还要完成加密、纠错、均衡、信号环境评估、信道接入控制、网络管理等功能。目前 DSP 无论在功能还是性能上，还不能完全满足软件无线电的要求。例如对一个带宽为 12.5MHz 通信信号，以 30MHz 的采样率进行采样，ADC 输出数据用一个性能良好的 FIR/IIR 滤波器进行滤波，假使对每个采样值运算 100 次的话，其运算量在 3000MIPS，再加上额外开销、控制的处理负荷，所要求的处理能力是相当高的。而目前处理速度最快的单片 DSP 芯片是 C62 的速率为 1600MIPS，而 C67 的速率为 1000MIPS，因而很难用单片 DSP 直接处理宽带射频或中频信号。由于实际通信往往是窄带的，其信号带宽为几十或几百 kHz。因此可以用多抽样率信号处理技术对采样信号进行预处理后，再用 DSP 来完成各种功能。这种预处理芯片就是所谓的数字下变频器。

9.3.5 软件无线电中的数字上/下变频器

数字变频技术是软件无线电的核心技术之一。

数字变频器的组成与模拟变频器类似，包括数字混频器、数字控制振荡器和低通滤波器三部分组成。但也存在区别，数字变频采用的是正交混频。上/下变频器的原理图如图 9-9 所示。

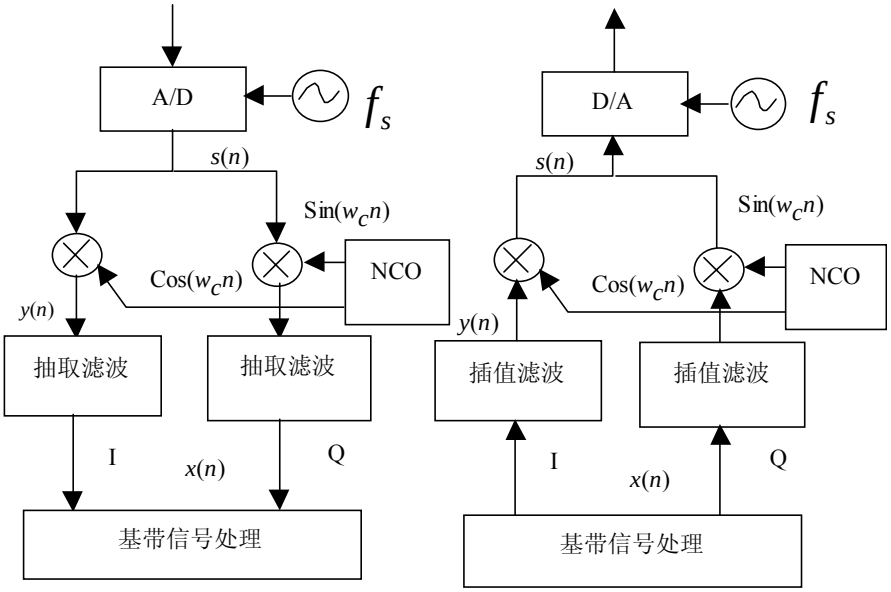


图 9-9 数字变频结构方框图（左图为数字下变频，右图为数字上变频）

在模拟变频中，混频器的非线性和模拟本地振荡器的频率稳定度、边带、相位噪声、温度漂移、转换速率等都是人们最关心和难以彻底解决的问题。这些问题在数字变频中是不存在的，频率步进、频率间隔等也具有理想的性能，另外，数字变频器的控制和修改比较容易等特点也是模拟变频所无法比拟的。影响数字变频性能的主要因数有两个：一个是表示数字本振、输入信号以及混频乘法运算的样本数值的有限字长所引起的误差；二是数字本振相位的分辨率不够宽而引起数字本振样本数值的近似取值。

自从 GrayChip 公司推出第一枚单信道数字下变频专门硬件芯片以来，市场上出现了多种多样的品种，其中最著名、应用最广泛的是美国的 Harris 公司（1999 年更名为 Intersil）和 Gray-Chip 公司，另外，还有 Analog Devices 公司与 Stanford Telecom 公司等。它们的主要产品的型号有：Harris 公司的 HSP50016、HSP50214 系列和最近推出的 HSP50216；Gray-chip 公司的 GC1011 系列、GC1012 系列、GC4014、GC4016；AD 公司的 AD6620、AD6224；Stanford Telecom 公司的 STL2130。数字上变频器件主要有 Harris 公司的 HSP50215、HSP50415；Gray-chip 公司的 GC4114；AD 公司的 AD9857 等。

9.4 软件无线电中的数字变频及 DSP 实现

由于数模转换器与 DSP 芯片技术的发展，数字变频开始替代了模拟混频，形成了变频数字信号处理。采用数字变频主要具有以下的优点：

- 载频与数字滤波器系数具有可编程性；
- 数字混频不存在非线性失真，因而互调小；
- 数字滤波频响特性好；
- 系统造价低。

数字变频结构如图 9-9 所示。变频数字信号处理包括多抽样率变换、正交混频与采样技术。

本节主要是介绍研究与 DSP 相关的正交混频与多抽样率变换技术及 DSP 实现。

9.4.1 正交变频及 DSP 实现

一、正交数字变频原理：

正交数字变频包括两个部分：一是乘法器、二是数字控制振荡器。

正交数字变频是将数字化后的信号分成两个信号，一个信号乘以 $\cos(w_0 n)$，下变频到零中心频率上，形成与原信号相位相同的信号，另一个信号乘以 $\sin(w_0 n)$，下变频到零中心频率上，形成与原信号正交的相位成份，如图 9-10 所示，其数学表达式为：

$$y(n) = s(n)e^{-j2\pi f_0 nT_s}$$

$$= s(n)\{\cos(2\pi f_0 nT_s) - j\sin(2\pi f_0 nT_s)\} \qquad (9\text{-}1)$$

其中 f_c 是载频，T_s 是采样间隔。

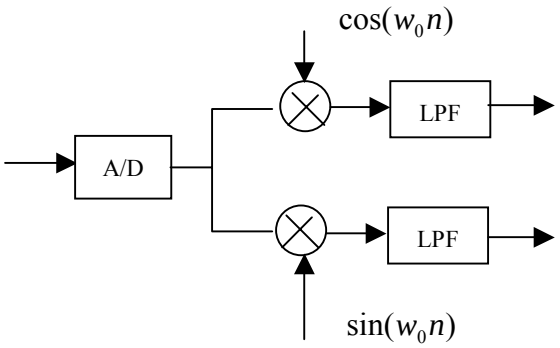


图 9-10　正交数字变频

正交数字变频中的正弦和余弦波是由数字控制振荡器产生的。

数字控制振荡器(NCO)

NCO 的目标就是产生一个理想的正弦和余弦波，更确切地说是产生一个可变频率的正弦波样本，如式(9-2)

$$S(n) = \cos(2\pi \bullet \frac{f_{LO}}{f_S} \bullet n) \qquad (n = 0,1,2,\cdots) \qquad (9\text{-}2)$$

式中 f_{LO} 为本地振荡频率；f_s 为正交数字变频输入信号的采样频率。

正弦样本可以用实时计算的方法产生，但这只适用于信号采样频率很低的情况。在软件无线电超高速的信号采亲频率的情况下，NCO 实时计算的方法是不可能实现的。此时，NCO 产生正弦样本的最有效。最简便的方法就是查表法，即事先根据各个 NCO 正弦波相位计算好相位的正弦值，并按相位角度作为地址存储该相位的正弦值数据，在每向输入一个待下变频的信号采样样本时，NCO 就增加一个 $2\pi\frac{f_{LO}}{f_S}$ 相位增量，然后，按照 $\sum_{i=0}^{n}2\pi\times\frac{f_{LO}}{f_S}$ 相位累加角度作为地址，检查该地址上的数值并输出到数字混频器，与信号样本相乘，乘积样本再经过低通滤波器输出，这样就完成了数字下变频。数字控制本地振荡器和数字混频器的功能框图如图 9-11 所示。

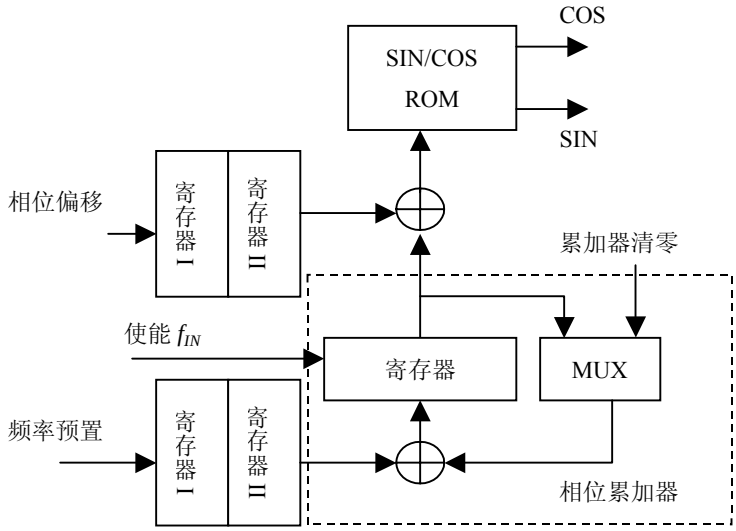


图 9-11 数控振荡器结构示意图

从上图可以看出，NCO 主要由三部分组成，包括相位累加器，相位加法器及 SIN/COS 表只读存储器组成。相位累加器的作用就是将数字本振频率转换成相位，每来一个时钟脉冲，相位在原来的基础上增加一个相位增量，相位加法器的功能是设置一定的初始相位以满足某些应用的需要。相位的正弦值　用查正弦表求得，也就是说，相位角度ϕ(0~2π)与其正弦值　表存在一一对应关系：ϕ－TAB(ϕ)，TAB(ϕ)表示以ϕ为地址，该地址上的内容数据。其实，只要保持一一对应关系查正弦表的地址不一定要真正的相位值，即若有 F－ϕ，则 F－TAB(ϕ)，这种对应关系的可传递性是毋庸置疑的，如式(9-3)所示。

$$\phi = 2\pi\times\frac{f_{LO}}{f_S}\times n \tag{9-3}$$

如果

$$F = \frac{f_{LO}}{f_S} \times 2^{n_b} \times n \tag{9-4}$$

式中 n_b 为表示相位二进制数据的位数，则 F－TAB(ϕ)。这种对应关系需要转换的原因是：由于实际的相位角度ϕ取值一般不是整数，这样相位角度直接用二进制数表示且作为查正弦表的地址是很复杂而且不确切的。用上式表示相位角度的好处是：相位被放大了 2^{n_b} 倍，使得相位的分辨率和本振频率的分辨率都大大增加了。

相位分辨率：

$$\Delta\phi = \frac{2\pi}{2^{n_b}}$$

频率分辨率为：

$$\Delta f_{LO} = \frac{f_S}{2^{n_b}}$$

若表示相位的二进制数据的位数 n_b 为 32 位，样本速率 f_s 为 65MHz，则相位分辨率和本振频率的分辨率分别为 0.000 000 084 度和 0.015 134Hz。具有这样的精度与分辨率对于模拟下变频来说是不可想象的。

相位的正弦值数据的位数取决于相位数据的位数，即前者必须能表示相位变化时，其相位正弦值的变化的最小值。我们知道在相位变化最小变化值的正弦值的最小变化值 发生在π/2±$\Delta\phi$ ~π/2 之间。这里，我们讨论相位在π/2±$\Delta\phi$ ~π/2 取值点的正弦值之差，即相位的正弦值数据的位数至少能表示该正弦值之差：

$$[\sin(\pi/2) - \sin(\pi/2 - \Delta\phi)] \bullet 2^{n_{bS}} \geq 1 \tag{9-5}$$

$$2^{n_{bS}} \geq \frac{1}{1-\cos(\Delta\phi)} \tag{9-6}$$

$$n_{bS} \geq \log_2[\frac{1}{1-\cos(\Delta\phi)}] \tag{9-7}$$

$$n_{bS} \geq \log_2[\frac{1}{1-\cos(\frac{2\pi}{2^{n_b}})}] \tag{9-8}$$

式中， n_b 为表示相位的正弦值二进制数据位数。

例如，当 n_b ＝16 时，由式计算得 $n_{bS} \geq 28$ 位。当相位的正弦值数据的位数小于式计

算值时，那么，相位的分辨率将发生“钝化”。

下面将类似推导在相位发生最小变化值时，能表示其正弦值的最大变化值的正弦值数据位数。这种情况发生在 0~$\pm\Delta\phi$、$\pi\sim\pi\pm\Delta\phi$ 取值点处。能表示 0 至 $\Delta\phi$ 取值点的正弦值之差的正弦值的数据位数的极限值：

$$[\sin(\Delta\phi)-\sin(0)]\bullet 2^{n_{bS}}\geq 1 \tag{9-9}$$

$$n_{bS}\geq \log_2[\frac{1}{\sin(\frac{2\pi}{2^{n_b}})}] \tag{9-10}$$

当 $n_b=16$ 时，由式计算得 $n_{bS}\geq 14$ 位。当相位的正弦值数据的位数小于式计算值时，那么，相位的分辨率将达不到 $\Delta\phi$。

NCO 产生的本振信号输入到数字混频器与输入的信号进行混频。数字混频器就是一个乘法器。信号经过混频后，输出到低通滤波器以滤除倍频分量和带外信号，然后进行抽取处理。

二、正交数字变频的 DSP 实现

[程序 9.1] TMS320C5410 映像寄存器初始化参数设置程序，在本章的后续程序中的所调用的映像寄存器的初始化参数设置程序均是此程序。

Init_54x.asm

```
*************************************************************************
*           for TMS302C5410 Register MAP
*************************************************************************
*************************************************************************
*                           ST0
*   ---------------------------------------------------
*   |15         13| 12    | 11 | 10    | 9    | 8             0|
*   ---------------------------------------------------
*   |     ARP    | TC |    C | OVA|OVB|       DP      |
*   ---------------------------------------------------
*************************************************************************
K_ARP               .set       000b<<13            ;ARP -> 000b -111b
K_TC              .set       1b<<12              ;TC=1 at reset
K_C                 .set       1b<<11              ;C=1 at reset
K_OVA               .set       1b<<10              ;OVA=0 at reset set OVA
K_OVB               .set       1b<< 9              ;OVB=0 at reset set OVB
K_DP                .set       00000000<<0         ;DP=000B at reset
```

```
K_ST0                .set       K_ARP|K_TC|K_C|K_OVA|K_OVB|K_DP
*************************************************************************
*                          ST1
*   ---------------------------------------------------------------------------
*   | 15    | 14   | 13 | 12   | 11   | 10   | 9    | 8   | 7   | 6    |   5    | 4       0|
*   ---------------------------------------------------------------------------
*   |BRAF|CPL | XF | HM |INTM| 0    |OVM|SXM|C16|FRCT|CMPT|    ASM |
*   ---------------------------------------------------------------------------
*************************************************************************
K_BRAF               .set       0b<<15              ;BRAF = 0 at reset
K_CPL                .set       0b<<14              ;CPL = 0    at reset
K_XF                 .set       1b<<13              ;XF = 1 at reset
K_HM                 .set       0b<<12              ;HM= 0    at reset
K_INTM               .set       1b<<11              ;INTM
K_ST1_RESR           .set       0b<<10              ;reserved
K_OVM                .set       1b<<9               ;OVM = 0 at reset
K_SXM                .set       1b<<8               ;SXM = 0 at reset
K_C16                .set       1b<<7               ;c16 = 0 at reset
K_FRCT               .set       0b<<6               ;frct = 0 at reset    set FRCT
K_CMPT               .set       0b<<5               ;CMPT =0 at reset
K_ASM                .set       00000B<<00          ;ASM =0 at reset

K_ST1_HIGH                                                                  .set
K_BRAF|K_CPL|K_XF|K_HM|K_INTM|K_ST1_RESR|K_OVM|K_SXM
K_ST1_LOW            .set       K_C16|K_FRCT|K_CMPT|K_ASM
K_ST1                .set       K_ST1_HIGH|K_ST1_LOW

*************************************************************************
*                          PMST
*   ---------------------------------------------------------------------
*   | 15              7|   6    | 5      | 4    |   3    |    2      | 1       0|
*   ---------------------------------------------------------------------
*   |       IPTR       |MP/MC|OVLY|AVIS|DROM|CLKOFF|Reserved|
*   ---------------------------------------------------------------------
*************************************************************************
K_IPTR               .set       001000000b<<7     ;111111111b at reset
K_MP_MC              .set       1b<<06             ;1 at reset
K_OVLY               .set       1b<<05             ;OVLY=0 at reset
K_AVIS               .set       0b<<04             ;AVIS=0 at reset
```

```
K_DROM              .set      1b<<03              ;DROM=0 at reset
K_CLKOFF            .set      0b<<02              ;CLROFF =0 at reset
K_PMST_RESR         .set      00b<<00             ;reserved for 548
K_PMST              .set
K_IPTR|K_MP_MC|K_OVLY|K_AVIS|K_DROM|K_CLKOFF|K_PMST_RESR

*************************************************************************
*                          SWWSR
*   ------------------------------------------------------
*   |     15    |14    12|11      9 | 8      6| 5       3 | 2       0|
*   ------------------------------------------------------
*   |Reserved| I/O    | Data    | Data |Program |Program|
*   ------------------------------------------------------
*************************************************************************
K_SWWSR_IO          .set      7000H
*************************************************************************
*                          BSCR
*   ---------------------------------------------------------------------
*   | 15                 12|    11    | 10               2|     1    |     0       |
*   ---------------------------------------------------
*   |     BNKCMP         |PS-DS|    Reserved       |    BH    |    EXIO |
*   ---------------------------------------------------------------------
*
*************************************************************************
K_BNKCMP            .set      0000b<<12           ;bank size = 64k
K_PS_DS             .set      0b<<11              ;1 at reset
K_BSCR_RESR         .set      000000000<<2        ;reserved
K_BH                .set      0b<<1               ;BH    =0 at reset
K_EXIO              .set      0b<<0               ;EXIO=0 at reset
K_BSCR              .set    K_BNKCMP|K_PS_DS|K_BSCR_RESR|K_BH|K_EXIO

IMR                 .set      00h       ;Interrupt mask register
IFR                 .set      01h       ;Interrupt flag register

ST0                 .set      06h       ;Status register 0
ST1                 .set      07h       ;Status register 1
AL                  .set      08h
AH                  .set      09h
AG                  .set      0Ah
```

```
BL              .set    0Bh
BH              .set    0Ch
BG              .set    0Dh
T               .set    0Eh     ;Temporary register
TRN             .set    0Fh     ;Transition register
;AR0            .set    10h
;AR1            .set    11h
;AR2            .set    12h
;AR3            .set    13h
;AR4            .set    14h
;AR5            .set    15h
;AR6            .set    16h
;AR7            .set    17h
;SP             .set    18h
BK              .set    19h     ;Circular-buffer size register
BRC             .set    1Ah     ;Block-repeat counter
RSA             .set    1Bh     ;Block-repeat start address
REA             .set    1Ch     ;Block-repeat end address
PMST            .set    1Dh     ;Processor mode status register
XPC             .set    1Eh     ;Program counter extension register

DRR20           .set    20h     ;McBSP0 data receive register  2
DRR10           .set    21h     ;McBSP0 data receive register  1
DXR20           .set    22h     ;McBSP0 data transmit register  2
DXR10           .set    23h     ;McBSP0 data transmit register  1

TIM             .set    24h     ;Time register
PRD             .set    25h     ;Time period counter
TCR             .set    26h     ;Time control register

SWWSR           .set    28h     ;Foftware wait-start register
BSCR            .set    29h     ;Bank_switching control register

SWCR            .set    2Bh     ;Software wait-start control register
HPIC            .set    24h     ;HPI control register

DRR22           .set    30h     ;McBSP2 data receive register  2
DRR12           .set    31h     ;McBSP2 data receive register  1
DXR22           .set    32h     ;McBSP2 data transmit register  2
```

```
DXR12          .set    33h     ;McBSP2 data transmit register   1
SPSA2          .set    34h     ;McBSP2 serial port sub-bank address register
SPSD2          .set    35h     ;McBSP2 serial port sub-bank data register

SPSA0          .set    38h     ;McBSP0 serial port sub-bank address register
SPSD0          .set    39h     ;McBSP0 serial port sub-bank data register

DRR21          .set    40h     ;McBSP1 data receive register   2
DRR11          .set    41h     ;McBSP1 data receive register   1
DXR21          .set    42h     ;McBSP1 data transmit register   2
DXR11          .set    43h     ;McBSP1 data transmit register   1

SPSA1          .set    48h     ;McBSP1 serial port sub-bank address register
SPSD1          .set    49h     ;McBSP1 serial port sub-bank data register

DMPREC         .set    54h     ;DMA channel priority and enable control
DMSA           .set    55h     ;DMA sub-bank-address register
DMSDI          .set    56h     ;DMA sub-bank data register with
DMSDN          .set    57h     ;DMA sub-bank data register

CLKMD          .set    58h     ;Clock mode register

;sub_address
SPCR1          .set    00h
SPCR2          .set    01h
RCR1           .set    02h
RCR2           .set    03h
XCR1           .set    04h
XCR2           .set    05h
SRGR1          .set    06h
SRGR2          .set    07h
MCR1           .set    08h
MCR2           .set    09h
RCERA          .set    0Ah
RCERB          .set    0Bh
XCERA          .set    0Ch
XCERB          .set    0Dh
PCR            .set    0Eh
```

```
K_DP1          .SET       500H/128
K_AD                .SET       500H

.TEXT
START   B    CONVERT_INIT
        NOP
        NOP
        SPACE   7CH*16
CONVERT_INIT:
        STM       #6FH,SP
        RPT       #1000
        NOP
        STM       #K_SWWSR_IO,SWWSR
        STM       #1,SWCR
        STM       #K_BSCR,BSCR
        STM       #K_ST0,ST0
        STM       #K_ST1,ST1
        STM       #K_PMST,PMST
        RPT       #1000
        NOP

        STM       #0b,CLKMD
CLKMAIN    LDM      CLKMD,A
        AND            #01b,A
        BC  CLKMAIN,ANEQ
        STM       #1001000101001111B,CLKMD          ;9.216*8
                  ;1001~~~~~~~~~~~~                 PLLMUL=10
                  ;~~~~0~~~~~~~~~~~                 PLLDIV
                  ;~~~~~00101001~~~                 PLLCOUNT
                  ;~~~~~~~~~~~~~~1~~                PLLON/OFF
                  ;~~~~~~~~~~~~~~~1~                PLLNDIV
                  ;~~~~~~~~~~~~~~~~1                PLLSTATUS

        STM       #60H,AR0
        RPTZ      A,#20H
        STL       A,*AR0+
        STM       #7FH,AR0
        RPTZ      A,#800H
        STL A,*AR0+
```

```
LD      #K_DP1,DP
RPT #07FFFH
NOP

STM     #0000000000000001B,IMR
            ;~~1~~~~~~~~~~~~~~      DMAC5
        ;~~~1~~~~~~~~~~~~~      DMAC4
        ;~~~~1~~~~~~~~~~~~      BXINT1  DMAC3
        ;~~~~~1~~~~~~~~~~~      BRINT1  DMAC2
        ;~~~~~~1~~~~~~~~~~      HPINT
        ;~~~~~~~1~~~~~~~~~      INT3
        ;~~~~~~~~1~~~~~~~~      BXINT2  DAMC1
        ;~~~~~~~~~1~~~~~~~      BRINT2  DMAC0
        ;~~~~~~~~~~1~~~~~~      BXINT0
        ;~~~~~~~~~~~~1~~~       TINT
        ;~~~~~~~~~~~~~~1~~      INT2
        ;~~~~~~~~~~~~~~~1~      INT1
        ;~~~~~~~~~~~~~~~~1      INT0

RSBX    INTM
B       MAIN
```

[程序 9.2]该程序是针对基于 DSP 的低中频下变频的数字化实现，其采样速率为 80kHz，中频为 25kHz。

Converter.asm

```
INCLUDE         "init_54x.ASM"
*SIN25K
SINSTP          .SET    0H
SIN25P          .SET    1H
SINNB0          .SET    2H
SINNB1          .SET    3H
SININT          .SET    4H
SINTLP          .SET    5H
SINOUT      .SET        6H
COS25P      .SET        7H
COSOUT      .SET        8H
*MAIN
TEMPM       .SET        9H
INPHASE     .SET        0AH
```

```
QUAD        .SET        0BH
MUOUT       .SET        19H
MIXDIN      .SET        1BH

K_IF80      .SET        25/80*128     ;步进相位步长

MAIN        IDLE    #1                              ;等待下一个中频采样数据输入
        CALL    SIN25K      ;产生 25kHz 的中频正弦余弦信号子程序
        STM     #MIXDIN+K_AD,AR3 ;数字正交混频
        STM     #COSOUT+K_AD,AR4
        MPY     *AR4,*AR3,A
        STH A,INPHASE       ;同相分量
            STM     #SINOUT+K_AD,AR4
        MPY     *AR4,*AR3,A
        STH A,QUAD          ;正交分量
        B   MAIN
SIN25K:         ST  #K_IF80,SINSTP      ;设置中频步长
SIN2GO      LD  SIN25P,A            ;SIN25P 存着数字压控振荡器上一时刻的相位
            ADD     SINSTP,A
                AND     #07FH,A
                STL     A,SIN25P        ;叠代出这一次相位的地址

                ADD     #SINTAB,A       ;读出这次正弦值
                READA   SINOUT

        LD  SIN25P,A            ;叠代出这次的余弦值的地址
        ADD     #32,A
        AND     #7FH,A
        READA   COSOUT

SINRET          RET
**
SINTAB
.word   07FFFH,07FD8H,07F61H,07E9CH,07D89H,07C29H,07A7CH,07884H
.word   07641H,073B5H,070E2H,06DC9H,06A6DH,066CFH,062F1H,05ED7H
.word   05A82H,055F5H,05133H,04C3FH,0471CH,041CEH,03C56H,036BAH
.word   030FBH,02B1FH,02528H,01F1AH,018F9H,012C8H,00C8CH,00648H
.word   00000H,0F9B8H,0F374H,0ED38H,0E707H,0E0E6H,0DAD8H,0D4E1H
.word   0CF05H,0C946H,0C3AAH,0BE32H,0B8E4H,0B3C1H,0AECDH,0AA0BH
```

```
.word   0A57EH,0A129H,09D0FH,09931H,09593H,09237H,08F1EH,08C4BH
.word   089BFH,0877CH,08584H,083D7H,08277H,08164H,0809FH,08028H
.word   08001H,08028H,0809FH,08164H,08277H,083D7H,08584H,0877CH
.word   089BFH,08C4BH,08F1EH,09237H,09593H,09931H,09D0FH,0A129H
.word   0A57EH,0AA0BH,0AECDH,0B3C1H,0B8E4H,0BE32H,0C3AAH,0C946H
.word   0CF05H,0D4E1H,0DAD8H,0E0E6H,0E707H,0ED38H,0F374H,0F9B8H
.word   00000H,00648H,00C8CH,012C8H,018F9H,01F1AH,02528H,02B1FH
.word   030FBH,036BAH,03C56H,041CEH,0471CH,04C3FH,05133H,055F5H
.word   05A82H,05ED7H,062F1H,066CFH,06A6DH,06DC9H,070E2H,073B5H
.word   07641H,07884H,07A7CH,07C29H,07D89H,07E9CH,07F61H,07FD8H
.word   07FFFH
.END
```

9.4.2 多抽样率信号处理及 DSP 实现

随着采样率的提高带来的另外一个问题就是采样后的数据流速率很高，导致后续的信号处理速度跟不上，特别是对有些同步解调算法，其计算量大，如果数据吞吐率太高是很难满足实时性要求的，所以很有必要对 A/D 后的数据流进行降速处理。信号经正交混频后，便将信号由射频或中频变换到基带。基带信号的最高有用频率远远小于变频之前的信号。对这样的基带信号进行降速处理或称为二次采样是完全有可能的。多速率信号处理技术为这种降速处理的实现提供了理论依据。多抽样率是指在一个系统中有两个或两个以上的抽样率。抽样率变换是指用数字的方法将一个信号的抽样率从一个给定的频率 $F=\frac{1}{T}$ 变换为一个不同的频率 $F'=\frac{1}{T'}$。本节将介绍多速率信号处理的一些基本概念和基本理论，其中最重要也是最基本的理论是抽取和内插，深入理解和掌握抽取和内插理论对软件无线电的研究和各种商品化数字上下变频的应用开发都是至关重要的。最后给出了 DSP 的实现。

一、抽取

抽取是指新的抽样率比原来的抽样率低的变换。现我们考虑降低整数 M 倍的过程，M 为抽取因子。

假定待处理的信号为 $X(n_1T_1)$，经 D 倍抽取后，信号为 $Y(n_2T_2)$。抽样处理框图如图 9-12(ａ)所示，我们将其等效为图 9-12(b)，即先令 $X(n_1T_1)$乘以一个序列$\lambda(n_1T_1)$，其定义如下：

$$\lambda(n_1T_1)=\begin{cases}1, n_1=0,\pm D,\pm 2D,\cdots\\ 0, \text{其他}\end{cases} \tag{9-11}$$

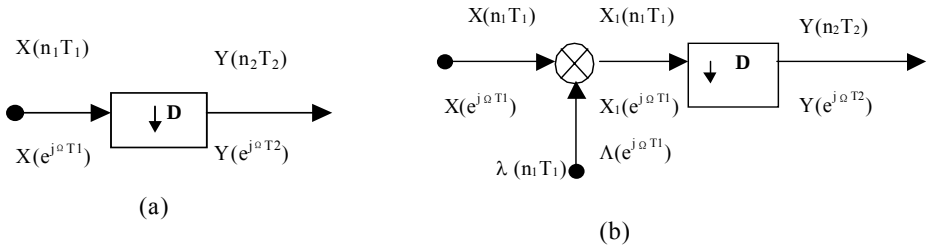


图 9-12　数字信号的直接抽取及其等效框图

$\lambda(n_1T_1)$的离散付立叶级数为

$$\Lambda(k\varphi)=\sum_{n_1=0}^{D-1}\lambda(n_1T_1)*\exp(-j(2\pi kn_1/D)),0\le k\le D-1 \tag{9-12}$$

其中，$\varphi=2\pi/DT_1$，$\Lambda(k\varphi)$的周期为 $D\varphi$。

从上面我们可以看出

$$\Lambda(k\varphi)=1，\qquad 0\le k\le D-1$$

于是有

$$\lambda(n_1T_1)=\sum_{k=0}^{D-1}\exp(j2\pi kn_1/D)$$

将上式代入式(9-12)后得

$$X_1(n_1T_1)=\sum_{k=0}^{D-1}X(n_1T_1)*\exp(j2\pi kn_1/D) \tag{9-13}$$

对 $X_1(n_1T_1)$进行抽取，每隔（D-1）点取一个抽样值，即所取的抽样点均在$\lambda(n_1T_1)$为 1 的点上，将这样的抽取结果作为 $Y（n_2T_2）$。显然这样得到的结果与直接对 $X（n_1T_1）$进行抽样所得到的结果是等效的。于是我们就可求得 $Y(e^{j\omega})$与 $X(e^{j\omega})$的关系如下：

$$\begin{aligned}Y(e^{jw})&=\sum_{n_1=-\infty}^{\div\infty}X_1(n_1T_1)e^{-jw\,n_1}\\&=\sum_{n_1=-\infty}^{+\infty}\left[\frac{1}{D}\sum_{k=0}^{D-1}X(n_1T_1)*\exp(j2\pi kn_1/D)\right]e^{-jw\,n_1}\\&=\frac{1}{D}\sum_{k=0}^{D-1}X(e^{jw}\,e^{-j2\pi k/D})\end{aligned}$$

从而可以看出，$Y(e^{j\omega})$是 $X(e^{j\omega})$的混迭样本。为了使抽取之后信号的频谱不致混迭，须使 $X(e^{j\Omega T_1})$中的最高频率成分Ωc 满足：

$$\Omega c\le(1/2D)\,\Omega_{SA1}=1/2\,\Omega_{SA2}$$

其中，Ω_{SA1}为 $X(e^{j\Omega T_1})$的周期，Ω_{SA2}为 $Y(e^{j\Omega T_2})$的周期。

抽取原理图如图 9-13 所示。

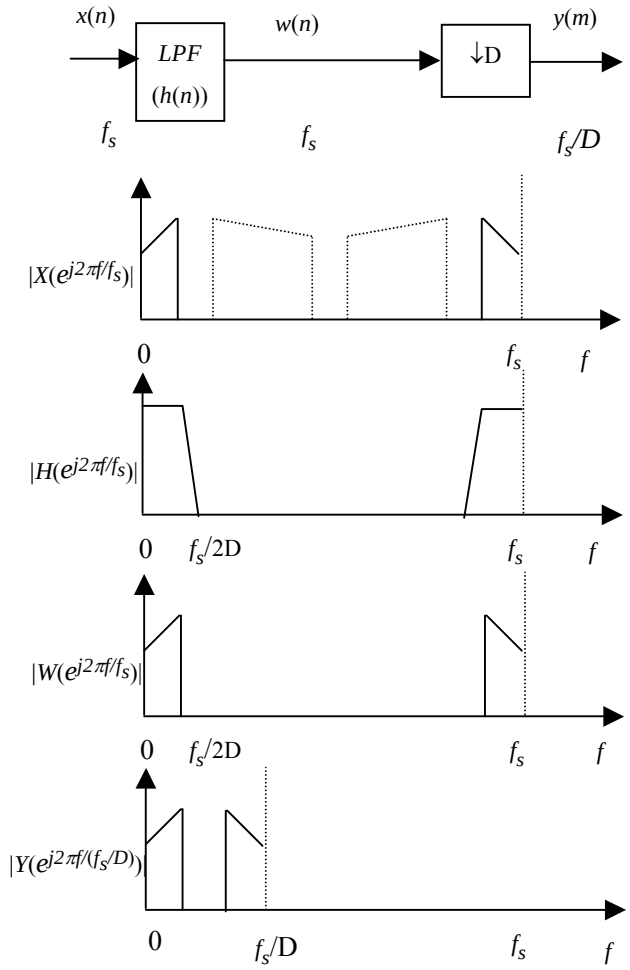


图 9-13 整数 D 倍抽取的框图与典型频谱

从时域来讲，抽取是每隔 D-1 个点，抽取一个点就行了。实际并不是这么简单。通过图 9-13 可以看到，经 f_s 采样后，除了有用信号外（实线表示），还有带外无用信号。若不经过任何处理，进行抽取则会产生频谱混迭，以致于无用信号落到有用信号的频带内，故须增加低通滤波。

抽取滤波的 DSP 实现如下：

[程序 9.3]该程序是针对基于 DSP 的低中频下变频的数字化实现，其采样速率为 80kHz，中频为 25kHz。抽样率为 5。

Decimate.asm

```
        .INCLUDE        "init_54x.ASM"
IDBASE  .SET    12H
QDBASE.SET      13H
IDAR2   .SET    14H
```

```
QDAR2  .SET     15H
IDECIM .SET     30H
IDECI1  .SET    31H
IDECI2  .SET    32H
IDECI3  .SET    33H
IDECI4  .SET    34H
QDECIM.SET      35H
QDECI1 .SET     36H
QDECI2 .SET     37H
QDECI3 .SET     38H
QDECI4 .SET     39H
K_DECLG    .SET     80
INDECIM    .SET     100H
QUDECIM    .SET     180H

MAIN   ST   #INDECIM,IDAR2     ; 设置下变频同相分量存储初始地址
       ST   #QUDECIM,QDAR2     ;设置下变频正交分量存储初始地址
       STM      #INDECIM,AR2
LOOP   IDLE     #1                  ; 等待下一个中频采样数据输入
       STM      #K_DECLG,BK              ; 设置循环寻址块的大小,即滤波器的阶
数大小
       STM      #1,AR0
       STM      #DECIMTAB,AR3     ; 抽取滤波器系数表地址
          MVDK  IDAR2,AR2          ; 中断率是按低频 16kHz 采样来的,故一个中断将
                    有 5 个
                                   ; 来数
       MVKD   #IDECIM+K_AD,*AR2+%      ; 自正交混频的数据,这几条程序
                                              ; 完成滤波数据的输入.
       MVKD   #IDECI1+K_AD,*AR2+%
       MVKD   #IDECI2+K_AD,*AR2+%
       MVKD   #IDECI3+K_AD,*AR2+%
       MVKD   #IDECI4+K_AD,*AR2+%
       MVKD   AR2,IDAR2

       RPTZ     A,#K_DECLG-1          ;同相支路抽取低通滤波
       MAC      *AR2+0%,*AR3+,A
          LD  #7FFFH,15,B
          MINA
          LD  #-7FFFH,15,B
```

```
            MAX     A
    STH A,IDBASE

    STM     #K_DECLG,BK
    STM     #1,AR0
    STM     #DECIMTAB,AR3

    MVDK    QDAR2,AR2       ;正交支路数据输入
    MVKD    #QDECIM+K_AD,*AR2+%
    MVKD    #QDECI1+K_AD,*AR2+%
    MVKD    #QDECI2+K_AD,*AR2+%
    MVKD    #QDECI3+K_AD,*AR2+%
    MVKD    #QDECI4+K_AD,*AR2+%
    MVKD    AR2,QDAR2

    RPTZ    A,#K_DECLG-1        ;正交去路低通滤波
    MAC     *AR2+0%,*AR3+,A
            LD  #7FFFH,15,B
            MIN A
            LD  #-7FFFH,15,B
            MAX     A
    STH A,QDBASE
    B   LOOP

DECIMTAB:
.word       -8,   -11,   -16,   -19,   -18,    -9,     6,    28
.word       51,    70,    77,    67,    35,   -15,   -78,  -138
.word     -180,  -186,  -145,   -54,    74,   218,   343,   413
.word      395,   270,    45,  -252,  -563,  -816,  -930,  -837
.word     -490,   116,   945,  1916,  2917,  3820,  4504,  4872
.word     4872,  4504,  3820,  2917,  1916,   945,   116,  -490
.word     -837,  -930,  -816,  -563,  -252,    45,   270,   395
.word      413,   343,   218,    74,   -54,  -145,  -186,  -180
.word     -138,   -78,   -15,    35,    67,    77,    70,    51
.word       28,     6,    -9,   -18,   -19,   -16,   -11,    -8
.END
```

二、内插

内插是指新的抽样率比原来的抽样率高的变换。现考虑提高整数 L 倍的过程，L 为内插因子。

设待处理的信号为 $x(n_1T_1)$， 经过 I 倍插值处理后的序列为 $y(n_2T_2)$。插值处理的实现方法如下，就是先在已知序列 $x(n_1T_1)$的相邻两抽样点之间等距离地插入（I-1）个 0 值点（我们称其为 0 值内插），然后进行低通滤波，即可求得 I 倍内插的结果，此方案的框图如图 9-14 所示：

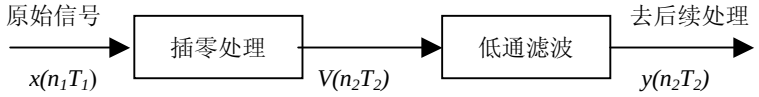


图 9-14 插值方案图

设经过 0 值内插之后得到的序列为 $v(n_2T_2)$，下面分析一下三者的频谱关系。设 $x(n_1T_1)$和 $y(n_2T_2)$为分别已抽样时间间隔 T_1 和 T_2 对 $x(t)$ 进行抽样所得的序列。则它们的频谱 $X(e^{jw})$和 $Y(e^{jw2})$均为周期函数。如果用真实的角频率Ω来表示，则 $X(e^{jw1}) = X(e^{j\Omega T1})$，其周期为 $\Omega_{sa1}=2\pi/T_1$；同理，$Y(e^{jw}) = Y(e^{j\Omega T2})$，其周期为 $\Omega_{sa2}=2\pi/T_2=2I\pi/T_1=I\Omega_{sa1}$。而对于 $v(n_2T_2)$的频谱，$V(e^{jw}) = V(e^{j\Omega T2})$，由于

$$v(n_2T_2)=\begin{cases} x(n_2T_2/I), & n_2=0,\pm I,\pm 2I,\cdots \\ 0, & \text{其他} \end{cases} \tag{9-14}$$

于是

$$\begin{aligned} V(e^{jw}) &= \sum_{n_2=-\infty}^{+\infty} v(n_2T_2)e^{-jw\,n_2} \\ &= \sum_{n_2=-\infty}^{+\infty} x(\frac{n_2T_1}{I})e^{-j\Omega T_1 n_2/I} \end{aligned}$$

由于 $n_2/I=n_1$，所以

$$\begin{aligned} V(e^{jw}) &= \sum_{n_1=-\infty}^{+\infty} x(n_1T_1)e^{-j\Omega T_1 n_1} \\ &= X(e^{jw}) \end{aligned}$$

从而可知 $V(e^{jw})$和 $X(e^{jw})$的频谱是相同的，只是 $X(e^{jw})$是以 $\Omega_{sa1}=2\pi/T_1$ 为周期，而 $V(e^{jw})$则是以 $\Omega_{sa2}=2\pi/T_2$ 为周期。

将 $V(e^{jw})$与 $Y(e^{jw})$相比较，可知多出了从 Ω_c 到 $\Omega_{sa2}-\Omega_c$ 之间的部分。所以，现在我们可以看出要想从 $V(e^{jw})$得到 $Y(e^{jw})$并不困难，只需将 $V(e^{jw})$通过以 Ω_c 为通带边缘频率的低通滤波器即可。

内插原理图如图 9-15 所示，设 I =3。

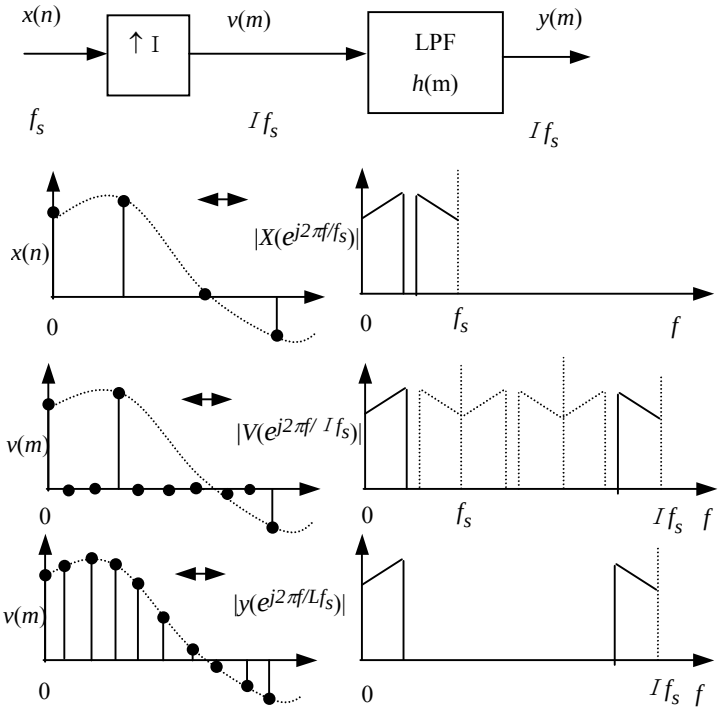


图 9-15 内插整数 I 倍的框图和典型波形

通过上图可知，在已知的相邻抽样点之间插入 L-1 个零值的点。其频谱相当于采样率为 Lf_s，频谱保持不变。经过低通滤波后，就得到了采样率为 Lf_s 的信号频谱。

插值滤波的 DSP 实现如下：

[程序 9.4]该程序是针对基于 DSP 的低中频下变频的数字化实现，其采样速率为 80kHz，中频为 25kHz。插值率为 5。

Interpolat.asm

```
.INCLUDE          "init_54x.ASM"
IINTER   .SET     0H
IINTE1   .SET     1H
IINTE2   .SET     2H
IINTE3   .SET     3H
IINTE4   .SET     4H
QINTER   .SET     5H
QINTE1   .SET     6H
QINTE2   .SET     7H
QINTE3   .SET     8H
QINTE4   .SET     9H
IBASE    .SET     0AH
```

```
QBASE   .SET      0BH
IAR2    .SET      1CH
QAR2    .SET      1DH
INPHASE     .SET      100H
QUAD    .SET      180H
K_RATIO     .SET      5
K_INTLG     .SET      16

MAIN    IDLE      #1                          ;等待下一次中断数据

    ST   #INPHASE,IAR2          ;输入同相分量数据首址
    ST   #QUAD,QAR2             ;输入正交分量数据首址
    STM      #INPHASE,AR2

        STM      #K_RATIO-1,BRC       ;由多抽样变换率决定一个新进数据的
                                          ；滤波次数
    STM      #K_INTLG,BK            ;每次滤波的滤波阶数
    STM      #1,AR0

    MVDK   IAR2,AR2             ;同相滤波
    STM      #IINTER+K_AD,AR1
             STM      #INTFILTTAB,AR3
    LD   IBASE,A
             STL A,*AR2

    RPTB     I_LOOP-1
    RPTZ     A,#K_INTLG-1
    MAC      *AR2+0%,*AR3+,A
             LD   #7FFFH,15,B
             MINA
             LD   #-7FFFH,15,B
             MAX      A
    STHA,1,*AR1+
I_LOOP
    MVKD    AR2,IAR2

    MVDK    QAR2,AR2
    STM      #K_RATIO-1,BRC
    STM      #INTFILTTAB,AR3            ;正交滤波
```

```
        STM     #QINTER+K_AD,AR1
            LD  QBASE,A
                STL A,*AR2

        RPTB    Q_LOOP-1
                RPTZ    A,#K_INTLG-1
                MAC     *AR2+0%,*AR3+,A
                LD  #7FFFH,15,B
                MINA
                LD  #-7FFFH,15,B
                MAX     A
        STH A,1,*AR1+
Q_LOOP
        MVKD    AR2,QAR2
        B   MAIN

INTFILTTAB:
.word           -2,     16,    -70,    226,   -587,   1327, -2866,   7570
.word        30259, -4290,   1683,   -693,    256,    -78,     17,     -2
.word           -3,     26,   -116,    378,   -994,   2295, -5179, 16221
.word        24326, -5915,   2483, -1051,    394,   -120,     27,     -3
.word           -3,     27,   -120,    394, -1051,   2483, -5915, 24326
.word        16221, -5179,   2295,   -994,    378,   -116,     26,     -3
.word           -2,     17,    -78,    256,   -693,   1683, -4290, 30259
.word         7570, -2866,   1327,   -587,    226,    -70,     16,     -2
.word            0,      0,      0,      0,      0,      0,      0, 32767
.word            0,      0,      0,      0,      0,      0,      0,      0
.END
```

三、一个有理因数 M/L 的抽样率变换

有理因数 M/L 的抽样率变换是先进行 I 倍的内插，再进行 D 倍的抽取。其原理框图如图 9-16 所示。

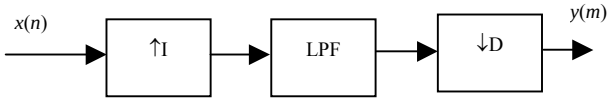


图 9-16 一个有理因数 I/D 的抽样率变换

9.4.3 软件无线电中高效的数字滤波

从前面的讨论已经知道，实现抽样率变换的关键问题如何实现抽取前与内插后的数字滤波。数字变频中往往采用(CIC)梳状滤波器、半带滤波器与 FIR 滤波器实现多级抽取插值结构。目前这部分工作主要是基于 FPGA 与专用数字上下变频器的，故还不涉及 DSP 实现。

一、积分梳状(CIC)滤波器

一般对低通滤波器的一些要求（即带通区、过渡区和阻带区）方面规定了多级系统中各级的滤波器频带。然而，这些滤波器特性比多级抽取器前面几级所需要的严格。多级抽取器前面几级中，0 到 F_S 频带常常只是那级输出抽样率的较小百分数（即 $F_S \ll F_i$），这时，阻带区可分为一些真正的阻带区和 Φ （即不用管）区 。

为了说明这一点，我们研究一下图 9-17（a）所示 I 级内插器的第 i 级滤波器的要求（内插器与抽取器的情况是一一对应的）。图 9-17（b）表示输入信号 $x_i(n)$ 的频谱，图 9-17（c）表示抽样率扩展了一个因子 L_i 后信号 $w_i(m)$的频谱。$h_i(m)$的作用是去除位于 F_i，$2F_i$，$3F_i$ ... 处的 $w_i(m)$中基带的谐波镜像，因此它应具有图 9-17（d）所示的特性。也就是说在各镜像之间，滤波器响应可不受限制，这些区域被称为不用管区域或中间带。

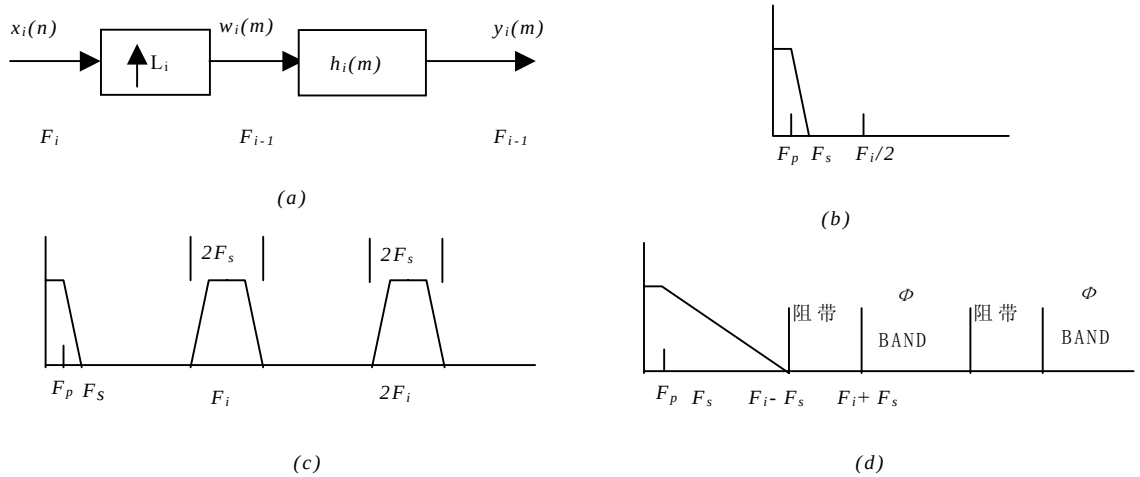


图 9-17 第 I 级内插器的滤波器频响图

当 F_S 比 F_i 小得多时，Φ带的宽度就便得较宽而要求的阻带区便得较窄。在极限情况下，当 F_S 趋向零时，要求的滤波器响应趋近“梳形”滤波器的响应。

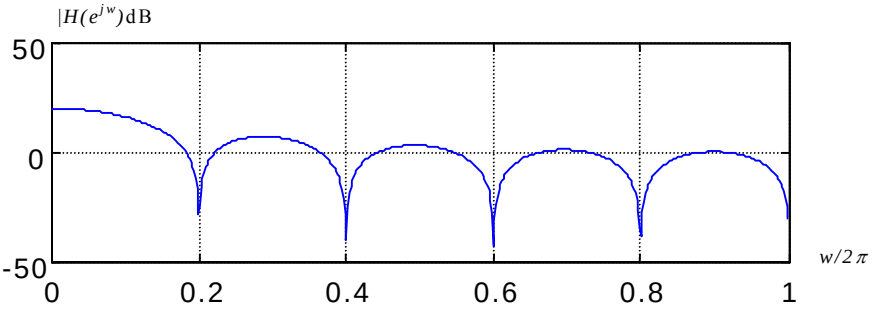
若 z=1 处极点、零点相抵消产生的低通滤波器，又可称为梳形滤波器。其冲激响应为

$$h(n)=\begin{cases}1 & 0 \le n \le N-1\\ 0 & \text{其它}\end{cases} \tag{9-15}$$

梳形滤波器的频响应特性为

$$H(e^{jw}) = \frac{\sin(wN/2)}{\sin(w/2)} \tag{9-16}$$

我们再做 N=10 时梳形滤波器的幅频特性如图 9-18



9-18 梳状滤波器频响图

从上图可以看出，梳形滤波器在 频率 $w = \frac{2\pi k}{N}, k = 1,2,..., N-1$ 处频响为零。一般一级 CIC 滤波器是难以满足实用要求的。为了降低旁瓣电平，可以采用多级 CIC 滤波器级联的办法来解决，例如用 Q 级 CIC 实现时的频率响应为：

$$H_Q(e^{jw}) = \left[\frac{\sin(wN/2)}{\sin(w/2)}\right]^Q \tag{9-17}$$

在多级 CIC 滤波器的设计中，要引入带宽比例因子 b，即设 $w_1 = b(2\pi/D)$ 则

$$A_1 = 20\lg\left|\frac{D\sin[\frac{\pi}{D}(1-b)]}{\sin(b\pi)}\right| \tag{9-18}$$

当 b<<1，D>>1 时，上式可简化为：

$$A_1 \approx -(20\lg b)$$

例如取 b＝0.01(f_s =100MHz，D=20)时，相当于信号带宽

$$f_1 = \frac{w_1}{2\pi} f_s = \frac{bf_s}{D} = 50\text{kHz}$$

则 A_1 ＝-20lg0.01=40

也就是说当 b=0.01 时，单级 CIC 滤波器的无混叠信号带宽内的阻带衰减也能达到 40dB，如果单级衰减不够，则仍可采用多级级联，这时的阻带衰减为：

$$A_1^Q = -Q(20\lg b)$$

$$= QA_1$$

即为单级时的 Q 倍，例如当 Q＝3，b=0.1 时，就能达到 60dB 的衰减，而这时的无混叠信号带宽支能达到 500kHz(f_s =100MHz，D=20)。值得指出的是上面引入的带宽比例因子 b，实际上是信号带宽(B)与抽取后的输出采样率(f_s /D)的比值。即

$$b = \frac{B}{f_s / D}$$

式中， f_s 为输入采样率，B 为抽取后的输出采样率，D 为抽取因子。为使信号带宽 B 一定的情况下，获得较大的带宽比例因子，应尽可能地采用小的抽取因子 D 或增大输入采样率 f_s ，后者就意味着 CIC 抽取滤波器一般要用在抽取系统的第一级（输入采样率最高）或者内插系统的最后一级（输入采样率也最高）。

带宽因子的选取需考虑的第二个问题是在 $w = w_b$ 时的幅度不能下降太多，也就是说在信号通带内幅值容差不能太大，若该容差为 δ_s ，则可求得：

$$\delta_s = 20\log\left|\frac{H(e^{j0})}{H(e^{jw_1})}\right| \qquad (9\text{-}19)$$

$$= 20\log\left|\frac{D\sin(\frac{w_1}{2})}{\sin(\frac{w_1 D}{2})}\right|$$

仍设 $w_1 = b\frac{2\pi}{D}$ 代入可得：

$$\delta_s = 20\lg\left|\frac{D\sin(\frac{b\pi}{D})}{\sin(b\pi)}\right|$$

当 D/b>>1 时， $\sin(\frac{b\pi}{D}) \approx \frac{b\pi}{D}$ ，

$$\delta_s = 20\lg\left|\frac{b\pi}{\sin(b\pi)}\right|$$

例如：当 b=0.1 时，$\delta_s \approx 0.143$dB

当 b=0.05 时，$\delta_s \approx 0.036$dB

也就是说从带内平坦度考虑，带宽因子 b 也不能太大，否则会引引起高频失真。同理我们可以得到 Q 级 CIC 滤波的带内容差为:

$$\delta_s^Q = Q \cdot \delta_s \tag{9-20}$$

也就是说 Q 级 CIC 滤波器的带内容差也是单级时的 Q 倍。由此可以看出多级级联虽然能增大阻带衰减，减小混叠影响，但会增大带内容差。所以 CIC 滤波器的级联数是有限的，不宜太多，一般以 5 阶为限。

二、半带滤波器

当抽取系统的抽取因子较大时，直接把抽样率转换工作一次完成，从计算工作量或存储量来说，往往不如经过两次或两次以上转换来得经济。我们把一次抽取完成所需要的抽样率

转换称为抽样率转换的单级实现，把两次或两次以上的抽取称为多级抽取。用多级实现构成高倍数抽取系统，基本上有两条途径。第一条途径是寻求最优化的方法，即以每秒钟的乘法次数为准则（或以存储量为准则）找出最佳的各级抽取因子，然后设计各级的滤波器。第二条途径则立足于使用抽取因子为 2 的抽取器。抽取因子为二的抽取器可利用一种称为 FIR 半带滤波器的滤波器，这种滤波器的冲激响应中有近一半的值为零，所以完成滤波所需要的乘法次数很少。在这种思路下实现总抽取因子为 2 的 K 次幂（即 $D=2^K$，K 为总的级数）最为方便。

半带滤波器的冲激响应 $h(n)$为实数且为偶对称，即 $h(-n)=h(n)$， $|n| \leqslant L$， $h(n)$的长度为 N，由于$|n| \leqslant L$，所以 N=2L+1 为一奇数。其频率响应有以下特征：

$$H[e^{j(\pi-w)}] = 1 - H(e^{jw}) \tag{9-21}$$

在 $w_s = \pi - w_p$ 的条件下（其中 w_s 及 w_p 分别为滤波器的通带上限和阻带下限频率），二倍抽取后的频率响应在通带内是没有混迭的，而在过渡带中是有混迭的。符合以上两个特征的滤波器称为半带滤波器。 半带滤波器具有以下特性：

（1） 半带滤波器偶数序号（不包括序号 0）的冲激响应为 0，即

$$h(n) = 0 \quad \text{当} \quad n = \pm 2, \pm 4, \ldots, n \neq 0$$

（2） 半带滤波器的频率响应在信号的抽样率降低一半后，虽其过渡带中是有混迭的，但保护了通带不受混迭。

（3） 半带滤波器要求通带误差容限 δ_p 与阻带误差容限 δ_s 相等。

半带滤波器的性质是近乎半数滤波器的系数精确等于零，因此实现这种滤波器时，乘法次数比对称 FIR 设计时少一半，而比任意其它 FIR 设计所需要的乘法次数少四分之三。不过，半带滤波器只适用于抽样率 2 比 1 变化场合，因此，要设计一个 *I* 级 *M* 比 1 的的半带抽取系统，就要在前面 I-1 级的每一级中将抽样率降低一个因子 2。即

$$M_1 = M_2 = \cdots = M_{I-1} = 2$$

最后一级的整数倍抽样率下降因子为 $M_I = \frac{M}{2^{I-1}}$

9.5 软件无线电中的调制解调算法及 DSP 实现

软件无线电具有灵活性、可扩展性等主要特点，这主要是因为软件无线电的所有功能都是用软件来实现（定义）的，通过软件的增加、修改或升级就可以实现新的功能。可以说，功能的软件化是软件无线电的最大优势之一。在所有软件中，数字信号处理软件占据着重要的位置，例如编码、调制、解调、译码、同步提取、频谱分析、信号识别等都可以采用信号处理算法来实现。本章将对调制解调几种数字信号处理算法作重点讨论。

9.5.1 调频（FM）及 DSP 实现

一、传统的 FM 实现

调频信号的数学表达式为

$$s_{FM}(t) = A_c \cos[\omega_c t + k_f \int_{-\infty}^{t} x(\tau) d\tau] \tag{9-22}$$

其中 k_f 为调频指数。

调频信号的产生方法有直接法和间接法两种，间接法的实现框图如图 9-19 所示：

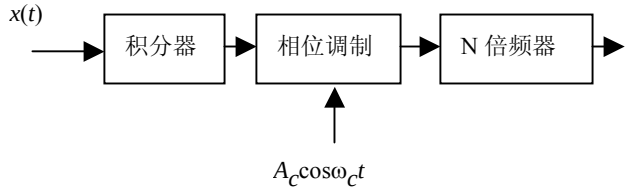


图 9-19　FM 间接法调制框图

调频信号的解调方法有鉴频器解调和锁相环解调两种方式，实现框图分别如图 9-20 所示

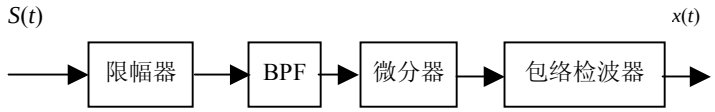


图 9-20 FM 鉴频器解调框图

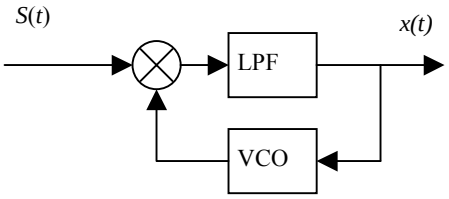


图 9-21　FM 锁相环解调框图

二、 FM 数字化实现

要把式(9-22)转化为离散数学表达式，除了要完成离散化以外，所以还要把式中包含的积分转化为合适数值积分，以便于数字处理。

数值积分有机械求积和插值求积两种。机械求积中又有梯形公式、矩形公式、龙贝格算法等，插值求积法中包括插值型求积公式、牛顿-柯特斯公式复化求积等算法。文献[28]指出复化求积法具有精度高、运算量小、容易得到各样点的积分值等优点，故本文中采用复化求积法实现 FM 连续数学表达式离散化。

复化积分法是将求积空间[a, b]划分为 n 等分，步长 $h=\dfrac{b-a}{n}$，分点为 $x_k=a+kh$，k=0,1,…,n。先求各自区间上积分值 I_k，然后再求和，用 $\sum_{k=0}^{n-1} I_k$ 作为求积分的近似值。

复化梯形公式为：

$$T_n=\sum_{k=0}^{n-1}\frac{h}{2}[f(x_k)+f(x_{k+1})] \tag{9-23}$$

采用复化求积公式后，FM 的离散数学表达式如下：

$$s(nT_s)=\cos[w_1 nT_s+k_f T_s\sum_{i=1}^{n}\frac{x(iT_s)+x[(i-1)T_s]}{2}] \tag{9-24}$$

(9-24)式中 T_s 是步长（采样间隔时间），相当于式(9-23)中的 h。式(9-24)三角展开后可以改写为下面的形式:

$$s(nT_s)=\cos[k_f T_s\sum_{i=1}^{n}\frac{x(iT_s)+x[(i-1)T_s]}{2}]\cos w_1 nT$$

$$-\sin[k_f T_s\sum_{i=1}^{n}\frac{x(iT_s)+x[(i-1)T_s]}{2}]\sin w_1 nT_s \tag{9-25}$$

FM 信号的调制就是根据式(9-25)计算得到的，其框图如图(9-22)所示。

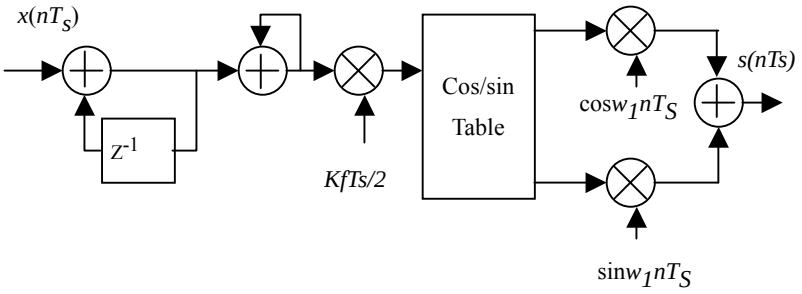


图 9-22 FM 信号的调制框图

FM 信号的解调采用鉴频器解调的方法。由式(9-25)可知 FM 信号是由两个正交分量组成的，将两正交分量通过 I、Q 支路分别下变频，计算出各采样点的相位，再经过微分电路即可得到基带信号。这里的微分完成的是模拟器件中鉴频器的作用。其解调原理框图如图(9-23)所示。

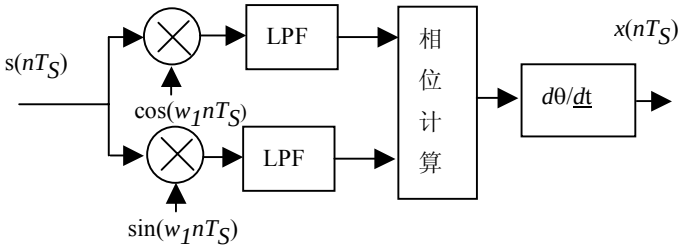


图 9-23　FM 信号解调框图

由于数字化，故连续微分需要转化成数值微分。数值微分有机械求导、插值求导和样条求导三种方法（参见参考文献［28］）。其中样条求导的方法计算量较大，而精度相对于插值求导提高不多，机械求导则精度不够。插值求导则是精度与运算量折衷后的选择。

插值求导方法如下：对于函数 $y=f(x)$，可以运用插值原理建立插值多项式 $y=P_n(x)$ 作为它的近似。多项式求导比较容易，取 $P_n'(x)$ 的值作为 $f'(x)$ 的近似值。

$$f'(x)=P_n'(x) \tag{9-26}$$

式(9-26)称为插值求导公式。

根据插值余项定理，求导公式的余项为

$$f'(x)-P_n'(x)=\frac{f^{(n+1)}(\xi)}{(n+1)!}w_{n+1}'(x)+\frac{w_{n+1}'(x)}{(n+1)!}\frac{d}{dx}f^{(n+1)}(\xi) \qquad (9\text{-}27)$$

式中 $w_{n+1}(x)=\prod_{i=0}^{n}(x-x_i)$

在这个余项公式中，由于ξ是 x 的未知函数，因此对任意给出的 x，误差是无法预估的。但如果限定求节点上的导数值，第二项为 0，则余项公式为

$$f'(x)-P_n'(x)=\frac{f^{(n+1)}(\xi)}{(n+1)!}w_{n+1}'(x) \tag{9-28}$$

设节点是等距的，求节点处的导数值有两点公式、三点公式和五点公式。其中五点公式较为实用。在这里给出五点公式。

设已给出五个节点 $x_i=x_0+ih$，i=0,1,2,3,4 上的函数值，五点公式为

$$\left\{\begin{array}{c} m_0=\frac{1}{12h}[-25f(x_0)+48f(x_1)-36f(x_2)+16f(x_3)-3f(x_4)] \\ m_1=\frac{1}{12h}[-3f(x_0)-10f(x_1)+18f(x_2)-6f(x_3)+f(x_4)] \\ m_2=\frac{1}{12h}[f(x_0)-8f(x_1)+8f(x_3)-f(x_4)] \\ m_3=\frac{1}{12h}[-f(x_0)+6f(x_1)-18f(x_2)+10f(x_3)+3f(x_4)] \\ m_4=\frac{1}{12h}[3f(x_0)-16f(x_1)+36f(x_2)-48f(x_3)+25f(x_4)] \end{array}\right\}$$

式中 m_i 代表一阶导数的近似值。

根据式(9-28)可以计算出各求导公式的余项分别为 $\frac{h^4}{5}f^{(5)}(\xi)$、$-\frac{h^4}{12}f^{(5)}(\xi)$、$\frac{h^4}{30}f^{(5)}(\xi)$、$-\frac{h^4}{12}f^{(5)}(\xi)$、$\frac{h^4}{5}f^{(5)}(\xi)$。由此可见，用 m_2 求导余项最小，所以在求导时采用 m_2 求导。

调频的 DSP 实现如下:

[程序 9.5] 程序就是用 TMS320c54x 实现调频的程序。

```
FM.asm
.DATA
MODE    .EQU    60H
RATE    .EQU    61H
Y0      .EQU    62H
Y1      .EQU    63H
Y2      .EQU    64H
```

```
A       .EQU    65H
B       .EQU    66H
RNG     .EQU    67H
SIG     .EQU    68H

 .SECT  “VECTOR”
RESET   B   START
TABLE   .TEXT
MDE     .WORD   00FAH
RTE     .WORD   014CH
Y00     .WORD   0000H
Y01     .WORD   2998H
Y02     .WORD   278EH
AA      .WORD   0C000H
RG      .WORD   0979H

START   STM     #0,AR0
        RPTZ    #7
        MVPD    TABLE,*+
            PORTW  MODE,0
        PORTW   RATE,1
        SSBX    SXM
WAIT    BC  MAIN,BIO
        B   WAIT

MAIN    LD  #0,A
        PORTR   SIG,2
        ST  T,RNG
        MPY     SIG,A
        STH A,1
        ADD     #5F1FH,A
        STH A,2
        ST  #0,Y2
        ST  #1,B
        LD  #0,A
        LD  *AR0-,1
        MPY     *AR0-,0,A
        LTD*AR0,1
        MPY     *AR0,0,A
```

```
STH A,1,Y0
LD  Y0,A
ADD     Y0,A
STL A,Y0
DMOV    Y0
PORTW  Y0,2
B     WAIT
.END
```

9.5.2 调幅波(AM)及 DSP 实现

一、AM 波调制的传统实现方法

调幅信号的数学表达式为：

$$s(t) = A_c[A_0 + x(t)]\cos\omega_c t$$

$$= A_c A_0[1 + mx(t)]\cos\omega_c t$$

其中 $m = \dfrac{1}{A_0}$ 为调幅度，通常 m<1。

AM 调制的实现框图如图 9-27 所示。

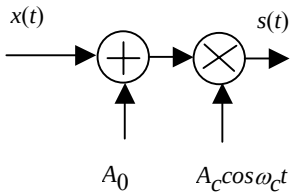


图 9-27　AM 调制实现框图

AM 的解调有包络解调和相干解调两种方法，其中包络解调具有实现简单和不需要同频载波的优点，得到了广泛的应用。包络解调的框图如图 9-28 所示。

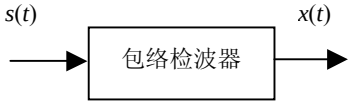


图 9-28 AM 的包络解调框图

二、AM 的数字化实现

AM 的离散数学表达式如下：

$$s(nT_s) = [1 + mx(nT_s)]\cos w_1 nT_s \qquad (9\text{-}29)$$

AM 波信号的调制采用直接计算的方法，根据表达式计算出输出值，调制框图如图 9-29 所示。

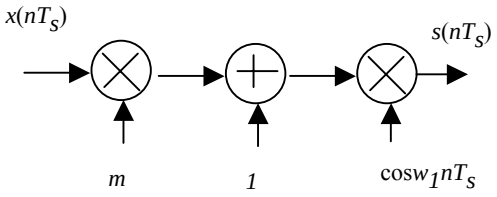


图 9-29　AM 调制框图

AM 信号的解调采用包络解调的方法。包络解调方法有两种：直接计算包络与通过正交载波相乘再求包络。一般采用后一种。这种解调方式的框图如图 9-30 所示。

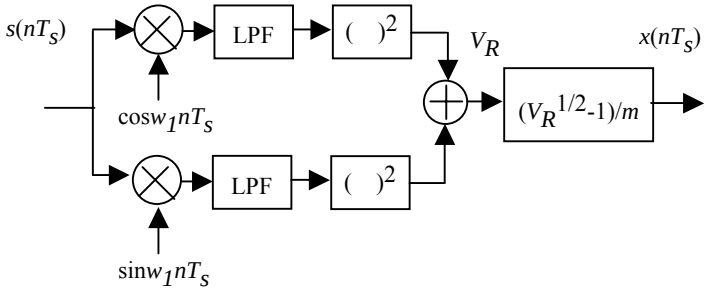


图 9-30　AM 波包络解调

由图 9-30 可知，这种解调方式需要有本地正交载波。虽然在前面的研究条件中，假设有本地同频同相载波。但这种解调方式对本地载波的要求不高。这是因为若本地载波存在频差 Δw，相差 θ，经低通滤波后 I、Q 支路的信号分别为

I：$\dfrac{1}{2}[1+mx(nT_s)]\cos(\Delta wnT_s+\theta)$

Q：$\dfrac{1}{2}[1+mx(nT_s)]\sin(\Delta wnT_s+\theta)$

由图 9-30 可知输出的解调信号为

$$x(nT_s)=\frac{\sqrt{\dfrac{1}{4}[1+mx(nT_s)]^2\cos^2(\Delta wnT_s+\theta)+\dfrac{1}{4}[1+mx(nT_s)]^2\sin^2(\Delta wnT_s+\theta)}-1}{m}$$

$$=\frac{[1+mx(nT_s)]-1}{2m}$$

$$=\frac{x(nT_s)}{2} \tag{9-30}$$

由式(9-30)可知，即使本地载波不同频不同相对解调不产生影响。

AM 的 DSP 实现

[程序 9.6]基于 TMS320c54 的 DSP 实现

```
AM.asm
                INCLUDE             "init_54x.ASM"
*SIN25K
SINSTP              .SET        0H
SIN25P              .SET        1H
SINNB0              .SET        2H
SINNB1              .SET        3H
SININT              .SET        4H
SINTLP              .SET        5H
SINOUT          .SET            6H
COS25P          .SET            7H
COSOUT          .SET            8H
*MAIN
TEMPM           .SET            9H
INPHASE         .SET            0AH
QUAD            .SET            0BH
MUOUT           .SET            19H
MIXDIN          .SET            1BH

K_IF80          .SET            25/80*128           ;步进相位步长

MAIN            IDLE        #1                      ;等待下一个中频采样数据输入,输入单元是
MIXDIN
          LD  MIXDIN,A    ;叠加一直流分量
          ADD       #1000H,A
          ST    A,MIXDIN
          CALL      SIN25K          ;产生 25KHz 的中频正弦余弦信号子程序
          STM               #MIXDIN+K_AD,AR3  ;数字混频
          STM               #SINOUT+K_AD,AR4
          MPY               *AR4,*AR3,A
          STH       A,INPHASE
          B         MAIN
SIN25K:               ST          #K_IF80,SINSTP          ;设置中频步长
SIN2GO          LD          SIN25P,A            ;SIN25P 存着数字压控振荡器上一时刻的相位
                    ADD         SINSTP,A
                    AND               #07FH,A
                    STL         A,SIN25P                  ;叠代出这一次相位的地址

                    ADD         #SINTAB,A             ;读出这次正弦值
```

READA SINOUT

SINRET RET

9.5.3 单边带(SSB)及数字化实现

一、SSB 调制的传统实现方法

SSB 调制传统的实现方法就是在模拟域内实现单边带信号调制的方法。常见的方法有：滤波法、移相法及混合法。

1. 滤波法

滤波法的原理如图 9-32 所示。

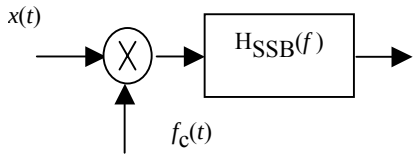


图 9-31 滤波法

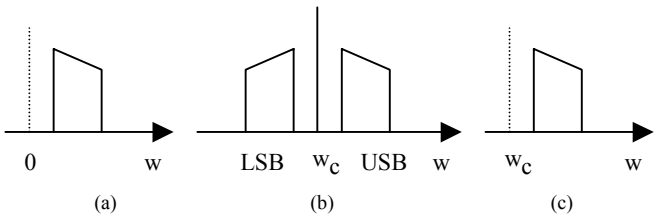


图 9-32 用滤波法产生单边带信号

滤波法实现单边带信号调制原理如下。假定话音信号 $x(t)$，其频谱如图 9-32(a)所示。对载波进行调幅后，所得的调幅波的频谱如图 9-32(b)所示。它由载频、上边带(USB)和下边带(LSB)三部分组成。与载波相乘后，经边带滤波器滤波后得到单边带信号如图 9-32(c)所示。用一个带通滤波器把所需的边带滤出来，而抑制载波与另一个边带。因而载波抑制、边带抑制与带内波动性能都由边带滤波器来决定。

2．移相法

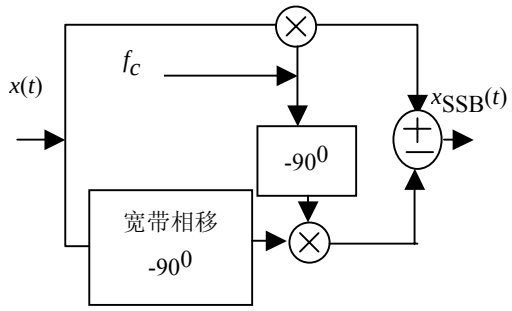


图 9-33 移相法

单边带信号的数学表达式为：

$$x_{SSB}(t) = x(t)\cos\omega_c t \mp \hat{x}(t)\sin\omega_c t \tag{9-31}$$

$\hat{x}(t)$ 是 $x(t)$ 的希尔伯特变换，减式表示上边带信号，加式表示下边带信号。$\hat{x}(t)$ 可以看成是 $x(t)$通过希尔伯特变换器的输出，该变换器的单位冲激响应 $h(t)=\dfrac{1}{\pi t}$，其频率响应是：

$$H(f)=\begin{cases}-j & f>0\\ j & f\le 0\end{cases}$$

也就是说希尔伯特变换器是一个全通滤波器，它对所有 f >0 的所有频率引入-90^0 相移；对所有 f <0 的频率引入 90^0 相移，即希尔伯特变换器相当于一个宽带-90^0 的相移网络。故移相法原理图如图 9-33 所示。

3．混合法

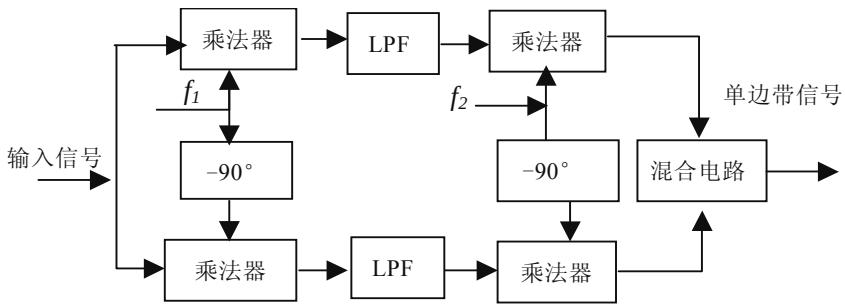


图 9-34 混合法单边带信号产生器

混合法产生单边带信号的方框图如图 9-34 所示。虚线方框中的 f_1 通常取基带信号频谱的中点，即

$$f_1 = F_{\min} + \frac{F_{\max}-F_{\min}}{2} = \frac{1}{2}(F_{\max}+F_{\min}) \tag{9-32}$$

式中，F_{max}、F_{min} 分别为基带信号频谱的最高和最低频率。低通滤波器的通频带为 B

$$B=\frac{1}{2}(F_{\max}-F_{\min}) \tag{9-33}$$

其中 $f_2 = f_c \pm f_1$，当取加号时，则为上边带；而取减号时则为下边带。

移相法中需要对输入的信号的各个频率成分均移相 90^0，而这样一个性能好的宽带移相网络难以实现，用其实现的单边带调制主要性能指标—单边带抑制仅为 30~40dB。混合法采用两个单频相移 90°的网络代替宽带相移网络，但其采用两个载频即 $f_2 = f_c \pm f_1$，多个乘法器等，实现较为复杂。以往用模拟方法来实现单边带调制时，主要采用滤波法。

对于滤波法，虽然工作频率较高，高性能的边带滤波器难以实现，但通过二~三次频谱搬移，降低第一中频的频率从而可以实现性能较好的边带滤波性能，其单边带抑制、载频抑制为 55dB 左右，带内波动<3dB。

二、SSB 调制的数字化实现方法

目前，由于数字信号处理芯片与数字专用芯片的发展，软件无线电的系统结构在短波电台中可以初步实现，其简图如图 9-35 所示。

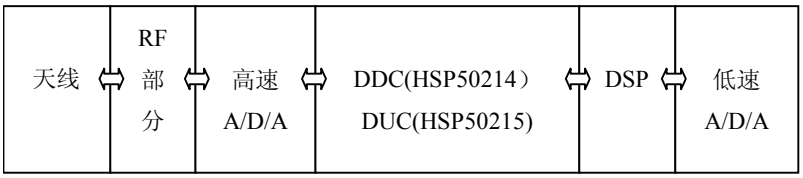


图 9-35 软件无线电系统结构简图

由于滤波法是在搬频之后进行单边带滤波的，因此采用新的系统结构后就需要在高数据速率下实现高性能的数字边带滤波。目前专用芯片与数字信号处理芯片都难以实现，例如假设中频为 500kHz，采样率为 1000kHz，为了达到晶体滤波器的性能，则数字滤波器至少需要 100 阶以上，而且上下各一个边带滤波器。仅实现两个边带滤波则需要 200MIPS 以上的数字信号处理芯片。

由于采用了数字技术，宽带相移网络却变得容易实现了。因而目前单边带信号数字化调制方法主要采用移相法。然而采用简单的移相法是难以实现高性能的单边带调制的。

三、SSB 调制解调复数滤波法

通过上面的分析可知，改进的移相法可以获得较好的单边带调制解调性能。但是改进的移相法为了达到几乎相近的幅频特性，需要较多的阶数，即运算量较大。之所以如此，主要是因为移相网络与对称滤波网络是独立设计的。为了达到较高的边带抑制，设计时必须使两网络的幅频特性尽可能的相近，相差<0.0056dB。为了得到较小的幅频误差，则带内波动要小(<0.02dB)，但实际所需的带内波动是<0.5dB。而滤波器的阶数设计是受带内波动、过渡带宽度与带外抑制等指标影响。这就需要寻求一种新的单边带调制解调方法，可以在满足带内波动、过渡带、带外抑制与边带抑制性能要求的前提下，获得最小的运算量。下一节，从单边带信号的一般表示法理论推导出复数表示法，并由此提出了复数滤波法。复数滤波法可以实现低运算量，高性能。

1. 单边带信号的复数表示

单边带信号的一般表示方法：

$$x_{SSB}(t) = x(t)\cos\omega_c t \mp \hat{x}(t)\sin\omega_c t \tag{9-34}$$

$\hat{x}(t)$ 是 $x(t)$ 的希尔伯特变换。

单边带信号复数的一般表示法的推导如下：

由式(9-34)可得：

$$x_{SSB}(t) = \mathrm{Re}[(x(t) \mp (-j)\hat{x}(t))e^{jw_c t}]$$

$$= \mathrm{Re}[(x(t) * \delta(t) \mp (-j)x(t) * h(t))e^{jw_c t}]$$

$$= \mathrm{Re}[x(t) * (\delta(t) \mp (-j)h(t))e^{jw_c t}]$$

其中 $\delta(t)$ 为冲激函数，$h(t)$为希尔伯特变换冲激响应函数。现令：

$$g(t) = \delta(t) \mp (-j)h(t) \tag{9-35}$$

由于 $h(t)$对应的频率响应为

$$H(f) = \begin{cases} -j & f > 0 \\ j & f \le 0 \end{cases}$$

故当取负号时，$g(t)$对应的频率响应为

$$G(f) = \begin{cases} 1 & f > 0 \\ 0 & f \le 0 \end{cases}$$

当取正号时，其对应的频率响应为

$$G(f) = \begin{cases} 0 & f > 0 \\ 1 & f \le 0 \end{cases}$$

由上可得单边带复数表达式为

$$x_{SSB}(t) = \mathrm{Re}[(x(t) * g(t))e^{jw_c t}] \tag{9-36}$$

$g(t)$为复数滤波器响应函数。

2. 单边带调制复数滤波方法

由式(9-36)可知，单边带调制复数法原理是将基带信号进行复数边带滤波后，进行复调制取实部，得到单边带信号，其原理方框图如图 9-36 所示。

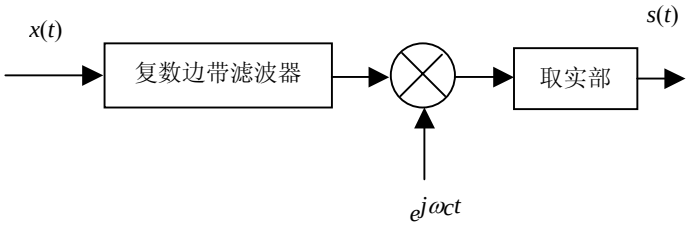


图 9-36 复数滤波法方框图

用复数法产生单边带信号的频谱图如图 9-37 所示。

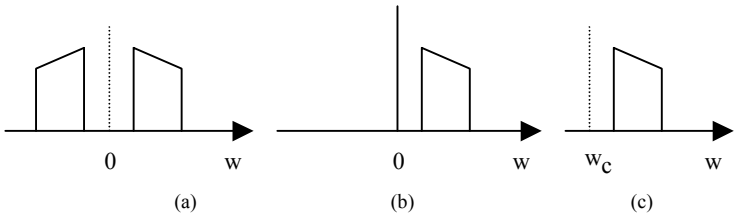


图 9-37 用复数滤波法产生单边带信号频谱图

图 9-37(a)是基带信号频谱图，(b)是经过复数边带滤波后，复数信号的频谱图，(c)是经过复调制取实部后的频谱图。将原理方框图 9-36 具体化则如图 9-38 所示。

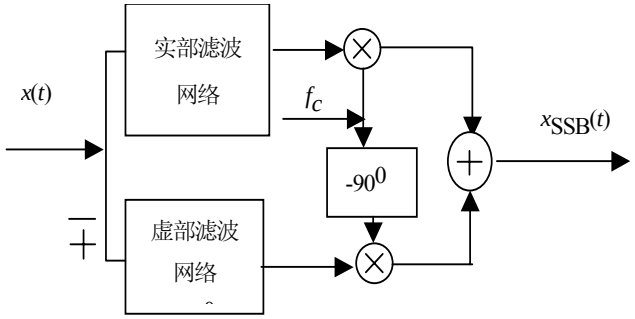


图 9-38 复数滤波法

利用复数滤波法实现独立边带也是很容易的，当图 3.17 中取负进入虚部滤波网络，则相当于式（9-35）中取负号，形成上边带；相反则是下边带

单边带的 DSP 实现如下：

[程序 9.7]基于复数滤波的单边带信号产生 DSP 程序

```
SSB.asm
.INCLUDE            "init_54x.ASM"

INPHASE.SET     1H
QUAD    .SET    2H
OUTRL   .SET    3H
OUTIG   .SET    4H
```

```
AUDOUT1     .SET      5H
REAL        .SET      100H
IMAG        .SET      180H

MAIN        IDLE      #1                              ;
            STM       #REAL,AR3
                      NOP
            NOP
                      LD        INPHASE,A             ;实部滤波
                STL     A,*AR3
                STM     #REAL+126,AR3
                      RPTZ      A,#126
                      MACD      *AR3-,#SSB_REAL,A
                STH     A,OUTRL

                      STM       #IMAG,AR3             ;虚部滤波
            NOP
            NOP
                      LD        QUAD,A
                STL A,*AR3
            STM       #IMAG+126,AR3
            RPTZ      A,#126
            MACD      *AR3-,#SSB_IMAG,A
            STH A,OUTIG
             NOP
            LD  OUTRL,15,A
            SUBOUTIG,15,A
            STH A,AUDOUT1
            NOP
            NOP
            B     MAIN
SSB_REAL.word   2,        4,       -3,      -11,       -5,       -2,      -10,       -5
            .word             -2,      -13,       -3,       -1,      -14,         3,         3,      -13
                        .word             16,       10,       -7,       35,       18,         4,       60,       25
            .word             20,       87,       25,       38,      111,       14,       56,      121
            .word           -12,       71,      111,      -59,       79,       69,    -126,       77
            .word           -12,    -209,       64,    -138,    -301,       41,    -309,    -386
            .word             8,    -526,    -443,      -30,    -790,    -438,      -68, -1121
            .word          -307,    -100, -1622,      160,    -121, -3106,    3582, 10794
```

```
        .word   3582, -3106,  -121,   160, -1622,  -100,  -307, -1121
        .word    -68,  -438,  -790,   -30,  -443,  -526,     8,  -386
        .word   -309,    41,  -301,  -138,    64,  -209,   -12,    77
        .word   -126,    69,    79,   -59,   111,    71,   -12,   121
        .word     56,    14,   111,    38,    25,    87,    20,    25
        .word     60,     4,    18,    35,    -7,    10,    16,   -13
        .word      3,     3,   -14,    -1,    -3,   -13,    -2,    -5
        .word    -10,    -2,    -5,   -11,    -3,     4,     2,     0

SSB_IMAG.word   0,     4,     9,     2,    -5,     2,     0,    -8
        .word     0,    -5,   -15,    -2,   -12,   -23,    -4,   -21
        .word   -30,    -3,   -30,   -32,     4,   -38,   -26,    17
        .word   -41,    -6,    38,   -39,    31,    64,   -28,    84
        .word    91,   -10,   152,   112,    15,   227,   115,    44
        .word   297,    87,    73,   348,    16,    97,   357,  -115
        .word   112,   303,  -322,   114,   149,  -636,   102,  -169
        .word  -1137,   77,  -856, -2160,    41, -3346, -8214,     0
        .word   8214,  3346,   -41,  2160,   856,   -77,  1137,   169
        .word   -102,   636,  -149,  -114,   322,  -303,  -112,   115
        .word   -357,   -97,   -16,  -348,   -73,   -87,  -297,   -44
        .word   -115,  -227,   -15,  -112,  -152,    10,   -91,   -84
            .word     28,   -64,   -31,    39,   -38,     6,    41,   -17
        .word     26,    38,    -4,    32,    30,     3,    30,    21
        .word      4,    23,    12,     2,    15,     5,     0,     8
        .word      0,    -2,     5,    -2,    -9,    -4,     0,     0
            .END
```

主 要 参 考 文 献

1. TMS320C2X User's Guide. Texas Instruments, 1990
2. TMS320C3X User's Guide. Texas Instruments, 1992
3. TMS320C4X User's Guide. Texas Instruments, 1991
4. TMS320C5X User's Guide. Texas Instruments, 1995
5. TMS320C62/C67 User's Guide. Texas Instruments, 1996
6. TMS320C80 User's Guide. Texas Instruments, 1995
7. ADSP2106X User's Guide, Analog Devices Inc. 1st Ed. 1995
8. 苏涛等. 高性能数字信号处理器与高速实时信号处理. 西安：西安电子科技大学出版社，1999 年
9. 李刚等. 数字信号微处理器的原理及其开发应用. 天津：天津大学出版社，2000 年
10. 彭启琮，李玉柏. DSP 技术. 成都：电子科技大学出版社，1997 年
11. A.V.奥本海姆，R.W.谢弗著. 数字信号处理. 北京：科学出版社，1980 年
12. 楼顺天，李博菡编著. 基于 MATLAB 的系统分析与设计—信号处理. 西安：西安电子科技大学出版社，1998 年
13. 沈振元，聂志泉，赵雪荷. 通信系统原理. 西安：西安电子科技大学出版社，1995 年
14. Theodore S. Rappaport. Wireless Communications Principles & Practice（影印本）北京：电子工业出版社，1998 年
15. John G.Proakis Digital. Communications Third Edition. 北京：电子工业出版社，1998 年
16. 王立宁，乐光新，詹菲. MATLAB 与通信仿真. 北京：人民邮电出版社，1999 年
17. 郭梯云，杨家玮，李建东. 数字移动通信. 北京：人民邮电出版社，1996 年
18. 吴启晖. 军用软件无线电中关键技术的研究. 南京：解放军理工大学博士学位论文，2000 年
19. 王新梅，肖国镇. 纠错码—原理与方法. 西安：西安电子科技大学出版社，2001 年
20. R.E.Blahut. 差错控制码的理论与实践. 广州：华南理工大学出版社，1988 年
21. 李建新，刘乃安，刘继平. 现代通信系统分析与仿真—MATLAB 通信工具箱. 西安：西安电子科技大学出版社，2000 年
22. 张雄伟，曹铁勇. DSP 芯片的原理与开发应用. 北京：电子工业出版社，2000 年
23. 宋祖顺等编著. 现代通信系统原理. 电子工业出版社，2001 年
24. 申敏，邓矣兵等编著. DSP 原理及其在移动通信中的应用. 北京：人民邮电出版社，2001 年
25. 赵荣黎. 数字蜂房移动通信系统. 北京：电子工业出版社，1997 年
26. 胡健栋，郑朝晖，尤必起等. 码分多址与个人通信. 北京：人民邮电出版社，1996 年
27. D. Upmal and R. Lackey, SPEAKeasy, The military software radio, IEEE Commun, Mag., May 1995
28. 李庆扬，王能超，易大义编. 数值分析. 武汉：华中理工大学出版社，1995 年
29. Peter G. Cook and Wayne Bonser, Architectural Overview of the SPEAKeasy System, IEEE Journal on Selected Areas in Communications, Vol. 17, No. 4, April 1999
30. Andrew J. Noga, Adiscrete-Time Method of Demodulating Large Deviation FM Signals,

IEEE Trans. on Communications Vol. 47, No. 8, Aug. 1999
31. D. Akos and J. B. Y. Tsui, Design and implementation of a direct digitization GPS receiver front end, IEEE Trans. Microwave Theory Tech., vol. 44, Dec. 1996
32. TMS320C54x Assembly Language Tools，Texas Instruments，1999
33. TMS320C54x Optimizing C Compiler，Texas Instruments，1999
34. TMS320C54x DSP Algebraic Instruction Set，Texas Instruments，1998
35. TMS320C54x DSP Enhanced Peripherals，Texas Instruments，1999
36. http://www.motorola.com/
37. http://www.analog.com/
38. http://www.ti.com/
39. http://www.nec.com/
40. http://www.zilog.com/