第8章 μ'nSP™单片机应用及开发技术

本章将介绍 μ'nSP™系列单片机的应用领域，具体讲述 SPCE061A 单片机在通讯、语音领域里的应用，并详细给出了有关系统的电路原理图、程序流程图以及程序代码，供读者参考。

8.1 μ'nSP™的应用领域

μ'nSP™家族产品具有电源电压范围和工作速率范围较宽、集成度高、性能价格比高以及功耗低等特点，故其有非常广泛的应用领域。μ'nSP™家族系列产品，涵盖了非常广泛的应用。包括：发音与语音识别的微控制器（SPCE 系列）、通信来电辩识应用的微控制器（SPT660x 系列）、以及通用型微控制器等等，主要体现在以下几个方面：

- 用于数字信号处理
- 用于开发研制便携式移动终端
- 用于开发嵌入式计算机应用系统

8.1.1 用于数字信号处理

1. 数字滤波器 (Digital Filter)

数字滤波器是一种计算处理或算法。借助于此，可以将输入的一种数字信号或序列变换为另一种序列输出。数字滤波器已被广泛地应用于数字语音、数字图像处理以及模式识别和频谱分析。

数字信号处理器（DSP，Digital Signal Processor）的作用是通过一系列数字来表示信号及其信息，并借助数字计算方法变换和处理这些信号。为了构成 DSP，必须有一种部件能够快速地完成两个数值的乘法运算并将乘积累加于寄存器。“快速”意味着乘和累加（MAC，Multiply & ACcumulate）较高的运算速度。若以 16 位数值进行乘和累加，其结果应为 32 位。

显然，μ'nSP™的硬件结构与其指令系统的结合足以构成 DSP 应用的硬件 MAC 单元，因而很适用于一些 DSP 方面的应用。

2. 数字信号的压缩编码与解码

通常，用于存储语音、图像等多种媒体信息的数字信号量非常巨大。这无论对于存储还是对于传输都是很不利的。为了节省存储空间或增强传输效率，自然使人想到：

将那些暂不运行的数字信号文件用某种算法进行压缩编码，待要运行时再释放还原，即解码。

3. 数字语音(音乐)信号处理

数字语音(音乐)信号处理是建立在 DSP 硬件基础上。通常 DSP 按运算的复杂程度分为定点和浮点两类，其根本区别在于数值的格式。定点 DSP 采用整数运算，对于大容量、低功耗的应用场合较合适；浮点 DSP 用于实数运算，最适宜于高性能且复杂场合的运算。μ'nSP™可用于定点 DSP 运算，且其成本较低，在语音处理这样的应用场合最能发挥出其特长。譬如像数字语音(音乐)信号处理中的频移处理算法就可用 μ'nSP™实现。

8.1.2 用于开发研制便携移动式终端

随着无线电通信及芯片技术的迅速发展以及电子商务的需求，使得便携移动式终端越来越朝着强实用性、个性化且时尚化的方向发展。个人数字助理（PDA，Personal Digital Assistant）就是符合这种时尚潮流的便携移动式终端。

若以 μ'nSP™家族产品为主，据 CPU 工作速率、存储容量和负荷能力，添加必要的外围电路并配合相应的一些外设，便可方便地设计出实用性强、可靠性高的 PDA 产品。图 8.1是基于 μ'nSP™的 PDA 基本硬件构成框图，该框图是 PDA 比较全方位的一个设计方案。针对不同的要求，可对其外围电路作适当删减修改，而设计出不同功能和不同价位的各款 PDA 产品。

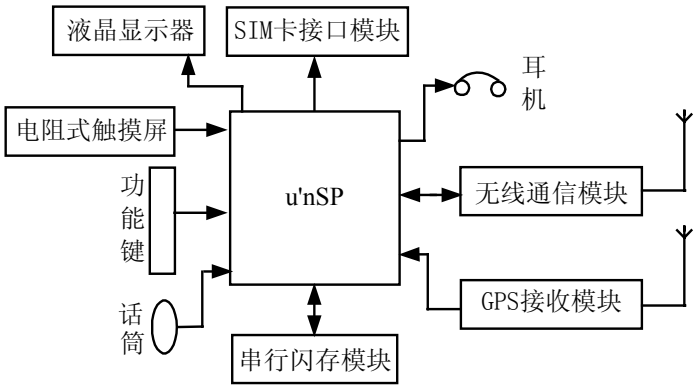

图8.1 基于μ'nSP™家族产品的 PDA 硬件构成

图 8.1中的 PDA 可以设计规划如下一些功能：

- 内置微型实时操作系统（RTOS, Real-Time Operating System）
- 语音识别输入或触摸屏输入
- 无线移动式语音通信
- 无线寻呼机、对讲机
- 双机或多机通信

- 下载升级软件
- 移动定位导航及位置信息服务
- 家电遥控器
- 个人资料数据库（名片册、医疗档案及家庭理财等）
- 皮包工具（字典、计算器、游戏机、学习机、万年历及钟表等）

8.1.3 用于开发嵌入式计算机应用系统

嵌入式计算机系统（ECS，Embedded Computer System）是指专门用于某一应用系统或设备并隐藏于其中的起关键支配作用的计算机应用系统。ECS 与通用计算机系统相比有以下一些特征：专用性、可封装性、外来性、实时性及可靠性。所谓外来性一般是指 ECS 自成一个子系统，与目标系统的其它子系统保持一定的独立性。

在不同的应用领域中对 ECS 有各自特殊的要求。例如：

(1) 小型应用系统，一般不需大量的数据处理，只需较强的实时控制功能，且要求体积小、功耗低等。这类系统如计算机智能化仪表、家电产品的自动控制等。

(2) 简单的工业控制系统，要求有相当强的实时数据处理能力和控制能力。如步进电机的驱动控制、数据采集、智能测量、汽车工业等。

(3) 比较复杂的系统中若采用分布式多机系统，在某些节点要配置智能 I/O 处理机对现场信息进行实时测量和控制。由于现场情况复杂，环境恶劣，故要求高可靠性和抗干扰能力等。如航空航天、尖端武器以及机器人系统等。

对于上述应用领域，第一类通常用 8 位机，如凌阳公司的 SPL 系列微处理器即可满足要求。而对于第二类则用具有定点 DSP 运算功能的 μ'nSP™系列的 16 位微处理器实现较为合适。至于第三类则可根据需要选用若干个 8 位机或若干个 8 位机与 μ'nSP™系列的 16 位机组合形成分布式多机系统。

μ'nSP™的特点决定其能很好地胜任于从简单到复杂的嵌入计算机系统。具体地可在如下一些应用领域里进行开发：

- 工业控制
  - 工厂自动化系统（锅炉、化工、电力等）
  - 智能化仪器仪表
  - 汽车控制（防撞系统、减震系统、静噪系统、燃油喷射系统、通信与音响等）
  - 机器人控制
- 消费、娱乐
  - 数字机顶盒
  - 游戏机、智能玩具、学习机
  - 家用电器
- 通信
  - 数字留言机
  - 数字语音信箱
  - 数字免提电话
- 军事

- 雷达与声纳信号处理
- 导航、制导
- 保密通信
- 全球定位、搜索跟踪

8.2 SPCE061A 单片机的应用举例

单片机的应用非常广泛，为了帮助读者尽快将 SPCE061A 用的得心应手，下面举几个 SPCE061A 单片机在不同领域中的应用实例,通过这些例子，读者可以对 μ'nSP™系列产品的应用窥见一斑。

8.2.1 单片机报时及作息时间控制

原理说明

本例所设计的是一个具有报时功能及作息时间控制钟。它利用 SPCE061A 单片机的 2Hz 时基计时，进行年历计算，并用 SPCE061A 的语音功能将它报出来；在进行时间计算，分每加 1 时，都与规定的作息时间比较，如果相等则进行相应的控制或动作。本例中假定某高校的作息时间如下所示：

08：00-------08：50	第一节课	09：00-------09：50	第二节课
09：52-------10：05	课间操	10：10-------11：00	第三节课
01：10-------12：00	第四节课	12：00-------13：30	午间休息
13：30-------14：20	第五节课	14：30-------15：20	第六节课
15：21-------15：50	播放歌曲		

硬件电路

硬件电路由键盘、声音输出模块和指示灯三部分组成，如图 8.2所示。

系统扩展三个按键用于报时及校正时间。SPCE061A 的 DAC 为电流型输出，经负载电阻 R11、三极管 8050 驱动扬声器 SPEAKER 放音，SPEAKER 可选用 4Ω或 8Ω扬声器。IOA15 接一个 LED，到规定的作息时间用 LED 闪烁来表示，使用者可根据具体需要来控制电铃、播放提示语等。凌阳芯片的工作电压为 3.3v，在图 8.2 中，我们给出了获得工作电压两种方法。

（1）通过两个二极管连续降压使 5v 的电压降至 3.6v，接近 3.3v 供芯片使用，这种方法比较简单，但电压值不是很精确。

（2）通过 LM7833 可获得准确的 3.3V 电压。

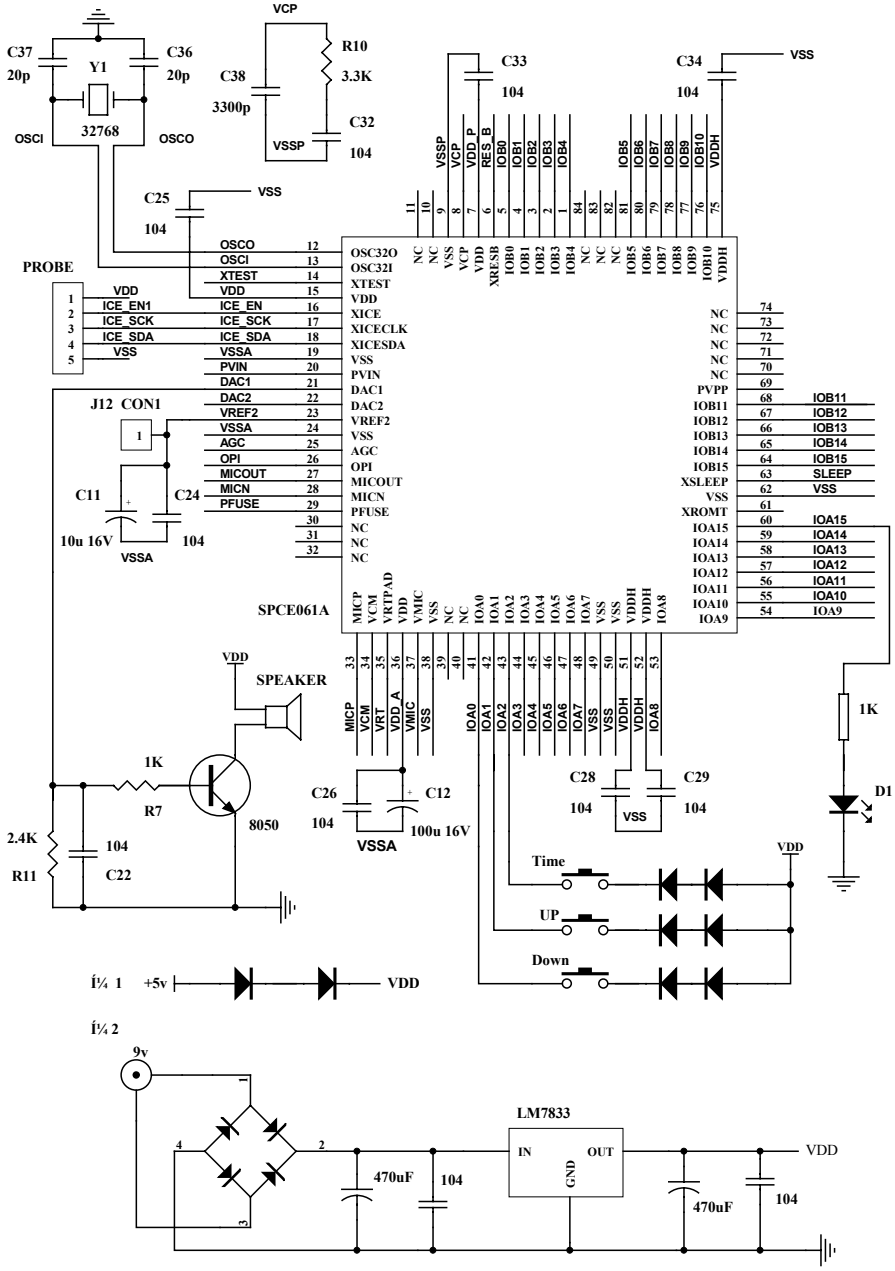

图8.2 硬件连接图

程序说明

整个程序分为主程序、键盘扫描子程序、万年历计算子程序、校时子程序、播放语音子程序几部分。

a) 主程序

程序按照结构化程序设计，所有功能都可通过调用子程序完成，主程序较简单，流程见图 8.3。SPCE061A 具有低功耗的睡眠模式，在睡眠模式下功耗电流可降到 2μA，这对于用电池供电的系统非常重要，睡眠模式可以通过按键中断唤醒。

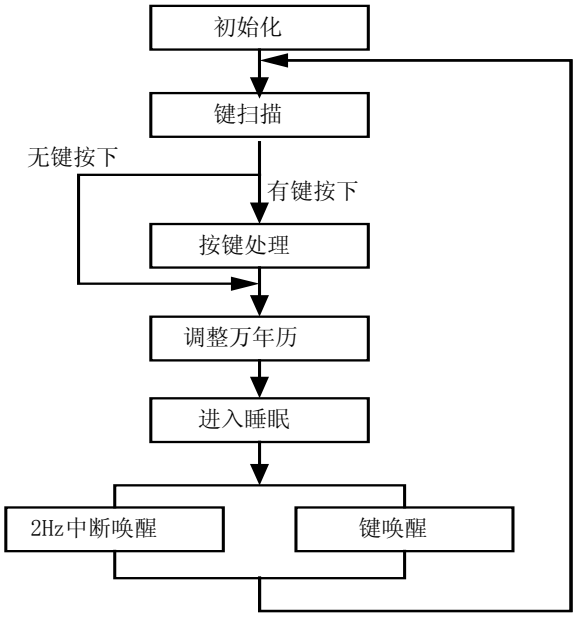

图8.3 主程序流程图

b) 键盘扫描程序

由于机械触点的弹性作用，在键被按下或弹起时会出现电压抖动，从最初按下到接触稳定要经过数毫秒的弹跳时间，如图 8.4所示。为保证键识别的准确，必须进行去抖动处理，去抖动有硬件和软件两种方法。硬件方法就是加去抖动电路，从根本上避免抖动；软件方法有很多种，本例中主要是利用主程序的循环扫描，主程序循环一次，扫描一次按键，当连续 N 次扫描到的键值都一样时，则说明是稳定的按键值。

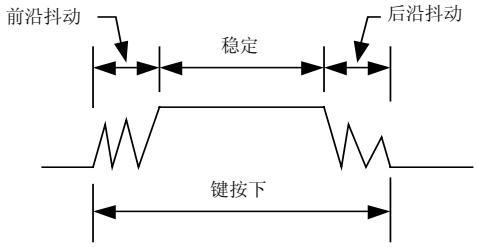

图8.4 键按下的过程

c) 万年历计算子程序

利用 2Hz 中断做时钟源进行计时，每两次中断秒加 1，并进行年历计算，年历范围从 2001 年到 3099 年。在进行年历计算时，有平闰年计算问题。闰年的条件是：能被 400 整除，或者能被 4 整除，但不能被 100 整除。万年历计算子程序流程图见图 8.5。

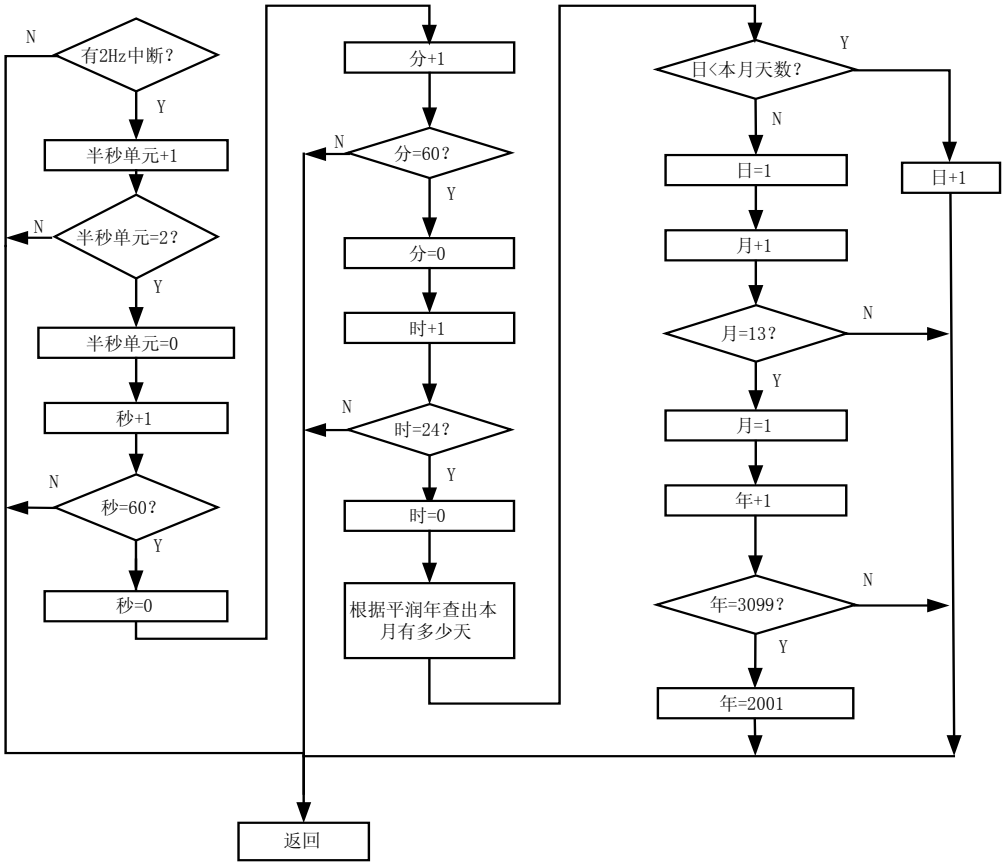

图8.5 万年历计算子程序

d) 校时子程序

系统扩展了三个按键，TIME 键用于报时，由于时间包括年月日和时分，按一次 TIME 键，则报年月日，再按一次则报时分。当一直按住 TIME 键 3 秒则进入时间校正状态，语音报当前是 XXXX 年，按 UP 键年份增加，按 DOWN 键减少；按 TIME 键来切换月、日、时、分，调整完分后，按 TIME 键确认，语音报出年月日时分。时间增加的流程如图 8.6所示，时间减少的流程与增加的相同，所以不再给出时间减少的流程图。在校正时间的状态下，如果连续 2 分钟键没有被按下，则自动退出。

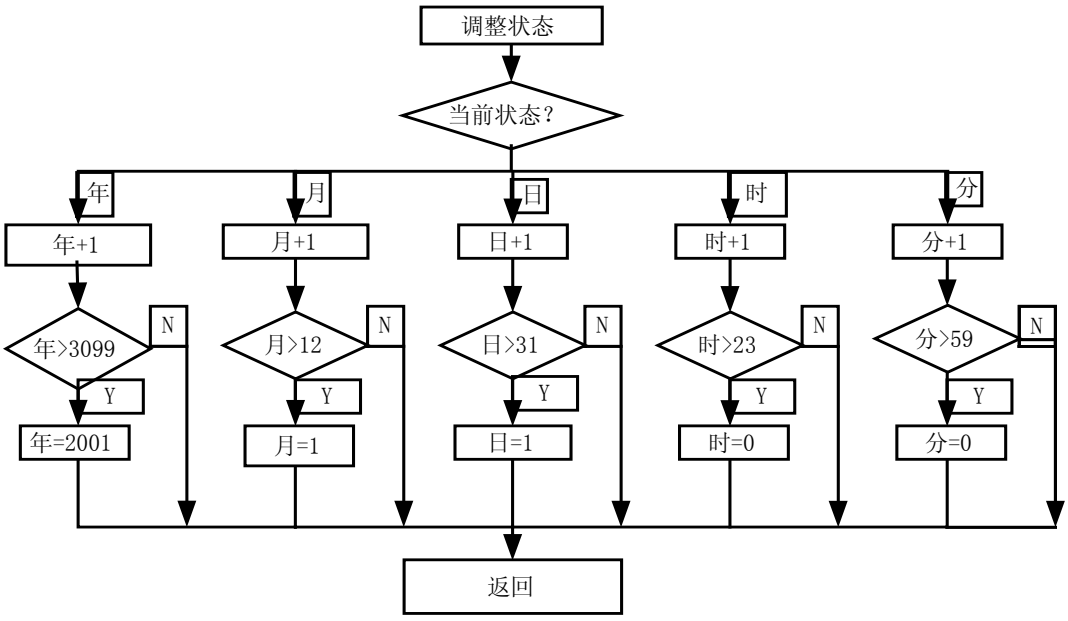

图8.6 校时子程序

e) 播放语音子程序

报时用 SACM-A2000 播放，关于 SPCE061A 放音的子程序，前面章节已有详细介绍，这里不再介绍。

8.2.2 热敏电阻温度计

常见的玻璃管温度计，是靠管内水银升降来判断温度值的高低。当光线较暗时，就看不清水银位置，给观察带来不便。这里介绍一种采用热敏电阻测温并用语音报告温度值的热敏电阻温度计，它具有使用方便的优点。

电阻测温原理：

热敏电阻是一种新型半导体感温元件，具有灵敏度高、体积小、寿命长的优点。热敏电阻可分为正温度系数和负温度系数两种类型。

负温度系数热敏电阻具有负的电阻温度特性，当温度升高时，电阻值减小；当温度降低时，电阻值增大，其阻值——温度特性曲线如图 8.7所示。热敏电阻的阻值——温度特性曲线是一条指数曲线，非线性较大，在实际使用中要进行线性化处理，但比较复杂，一般只使用线性度较好的一段（如图 8.7所示 ab 段）。如果测出热敏电阻的阻值，就可以算出对应的温度值。

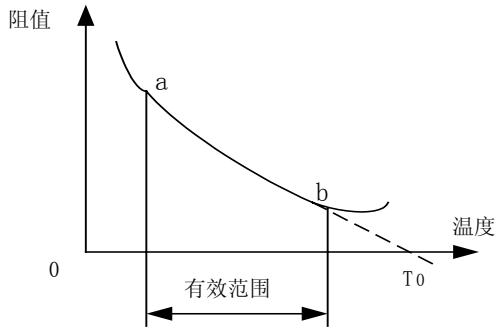

图8.7　热敏电阻温度特性曲线

硬件电路：

用热敏电阻测温的硬件连接见图 8.8。将热敏电阻 RT 与固定电阻 R 串联，接 3.3V 电源，当温度改变时，RT 阻值改变，其两端的电压随之改变，测量两端的电压，通过以下公式求得温度值：

$T=T0-KV_T$

其中：

T———-被测温度

T0———与热敏电阻特性有关的温度参数

K———-与热敏电阻特性有关的系数

V_T———热敏电阻两端的电压

此例中选用负温度系数热敏电阻 MFD-502-34，其线性化较好的一段是在-20℃～80℃。

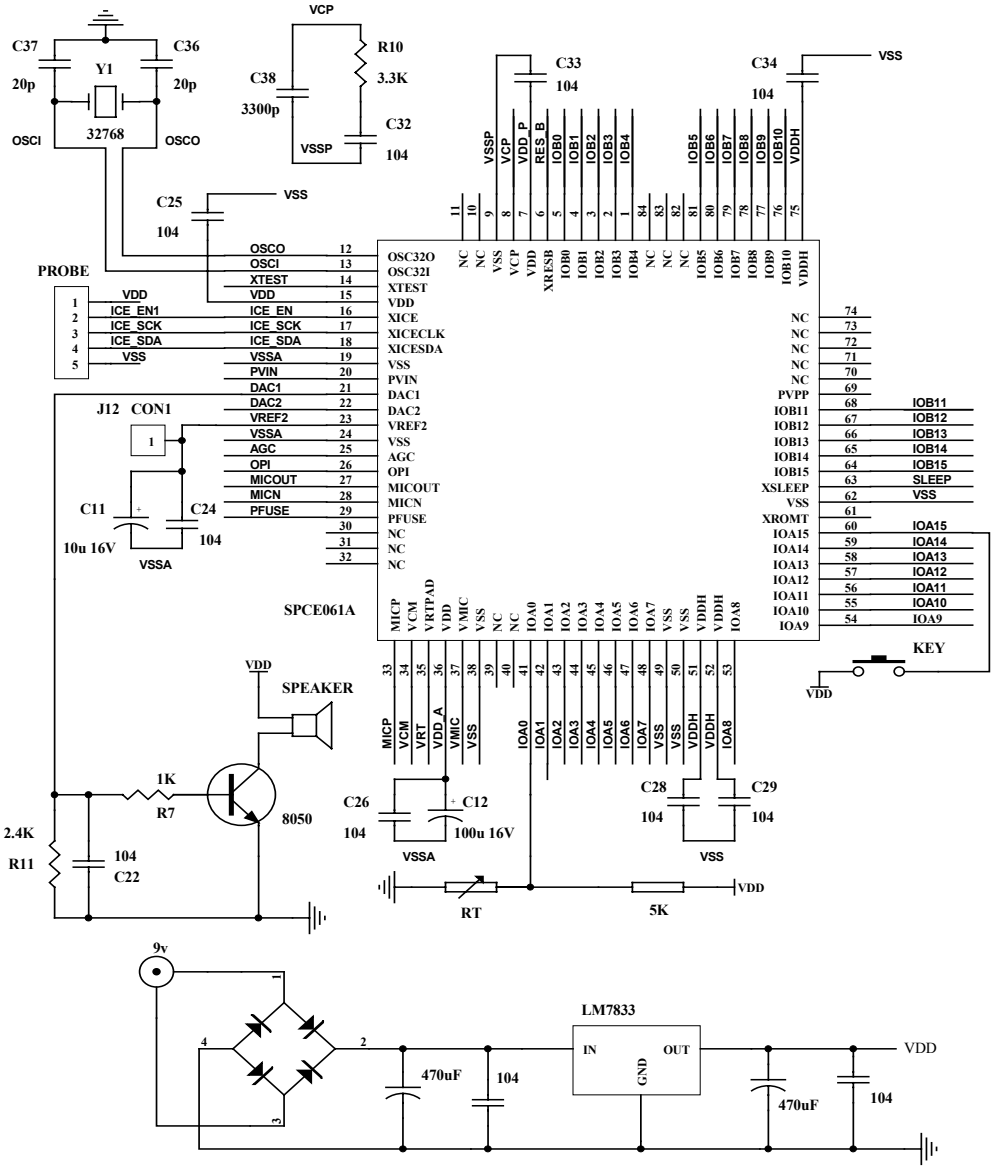

图8.8 硬件连接图

固定电阻 R 阻值的选取：

MFD-502-34 型热敏电阻线性化较好的一段是在-20℃——80℃，为了在最高温度和最低温度时使被测信号基本接近满量程值，采取线性区域内中间某一点温度的阻值作为固定电阻的值。它们分压后，AD 的输入电压是 AD 的输入电压范围一半。在 25℃时热敏电阻的阻值为 5KΩ，所以选取固定电阻 R 的值为 5KΩ。

在-20℃时热敏电阻的阻值为 37.399KΩ，热敏电阻两端电压 VR_T=2.9V，接近 A/D 输入电压的上限 3.3V；在 80℃时热敏电阻的阻值为 0.796KΩ，热敏电阻两端电压 VR_T=0.45V，接近 A/D 输入电压的下限 0V。

在温度线性化较好的区域内 SPCE061A 的 A/D 值都没有达到极限值。按照图 8.8接法时，T0=76，K=0.1022，根据以上公式和参数，测出热敏电阻两端的电压就可以求出被测温度。

温度计算：

系统扩展了一个按键，接于 IOA15，当按键按下时，就进行 A/D 转换初始化，并进行 4 次 A/D 转换，SPCE061A 的 A/D 转换结果在高 10 位，每次将其移入低 10 位再计算 4 次平均值作为 AD 有效结果返回；为了提高准确度，变量 TempAD、Temper 都采用浮点数，计算完成用语音将温度值报出来。由于在放音时播放函数会改变一些参数，为了稳定起见，在每次 A/D 转换前都做一次初始化。

由于每个热敏电阻的特性并非一样、与热敏电阻串联的固定电阻的不准确等原因，每支温度计在整个测量范围内至少找 5 点进行校正，并适当的修改参数以达到最佳状态。

A/D 转换程序：

```
.PUBLIC _ReadAD
_ReadAD: .proc
R2=4                                    //共进行 4 次转换
R3=0
TestLoop:
    R1=[P_ADC_MUX_Data]                 //进行一次 AD 转换
    R1=0x8000
TempConverLoop:
    TEST R1,[P_ADC_MUX_Ctrl]
    JZ TempConverLoop                   //转换完成?
    R1=[P_ADC_MUX_Data]                 //读取 AD 转换值
    R1=R1 LSR 4                         //将 AD 值移到低 10 位
    R1=R1 LSR 2
    R3+=R1                              //四次 A/D 值累加
    R2-=1
    JNZ  TestLoop
    R3=R3 LSR 2                         //结果除以 4，求 4 次 A/D 值的平均值
    R1=R3                               //R1-------A/D 返回值
    RETF
  .ENDP
```

温度计算及语音播报部分程序：

```
if(Key==0x8000)
    {
        InitAD();
         TempAD=ReadAD();               //进行一次 AD 转换
        Temper=76-0.1022*TempAD;        //温度计算
         if(Temper<-20 | Temper>80)
          Temper=0xFFFF;                //温度超出范围
          PlayVoice(Temper);            //语音报温
    }
```

8.2.3 三角波、正弦波、方波波形发生器

此例介绍几种基本波形的形成原理和实现方法。为满足不同的需要，在这个例子中所有波形的频率都为可调的。下面具体介绍一下波形的形成过程。

1. 原理分析

三角波形成原理：如图 8.9所示三角波的上升沿和下降沿都是由 N 个 DAC 输出的小阶梯构成，由于阶梯很小，从宏观上看它可以近似成三角波波形，如图 8.9中实线所画的波形。阶梯之间的时间间隔 t1、t2 由定时器 TimerA 的定时值决定。通过改变 t1、t2 的值来改变三角波上升沿和下降沿的时间 T1、T2，从而改变三角波的频率。

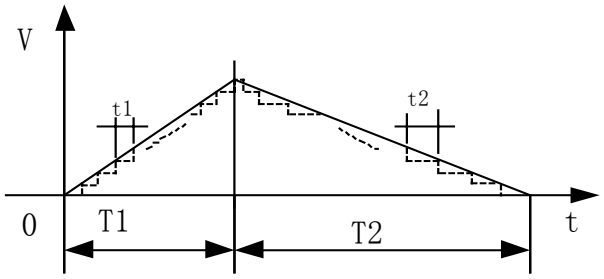

图8.9 三角波原理图

正弦波形成原理：SPCE061A 的开发环境提供了数学运算函数库 math.lib。利用库中的 sinf（float<[x]>）函数，sinf（float<[x]>）函数的 x 从 0---2π变化时，就是一个完整的正弦波。以时间间隔 t 求的波形对应值 32736*sinf（t）(注：32736 为 SPCE061A 的 D/A 输出最大值 0XffC0 的一半)，经 SPCE061A 的 D/A 转换在管脚 DAC1（DAC2）输出，并经电容滤波后就可以从 DAC1（DAC2）管脚得到需要的正弦波。通过改变时间间隔 t 可以改变正弦波的频率。

正弦信号包括正负半周（波形如图 8.10所示），由于 SPCE061A 无法输出负电平，所以将正弦信号电平正向偏移 1.65V（最大输出 3.3V 的一半）， 具体内容可参见后面的程序说明部分。

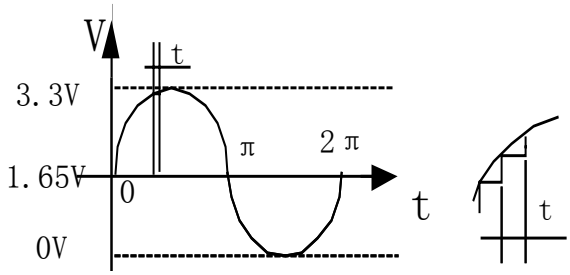

图8.10 正弦波

方波形成原理：如图 8.11所示，在 t1 期间 DAC 输出电压的峰值为 3.3V，t2 期间 DAC 输出 0V 电压。通过改变 t1、t2 的值来改变方波的频率（占空比）。

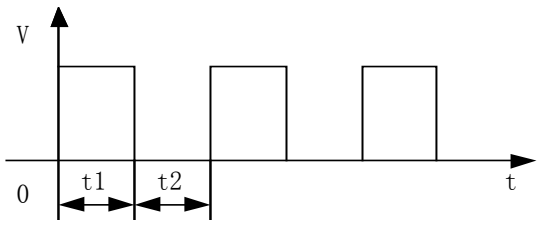

图8.11 方波

2. 硬件电路

SPCE061A 内部具有 D/A 转换模块，所以其硬件电路很简单,只要在 DAC 的输出端接一个负载电阻，将 DAC 输出的电流信号转换为电压信号即可。因需要输出不同的波形，在此例中选用按键方式进行选择，具体电路如图 8.12所示。

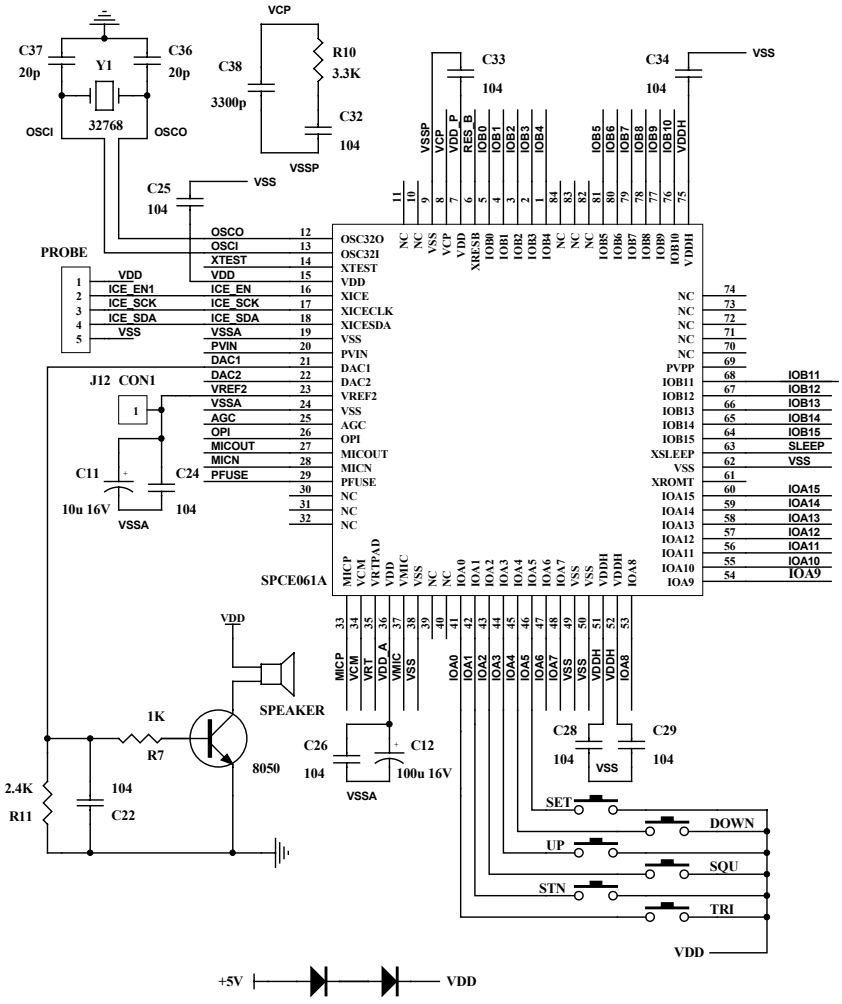

图8.12 硬件连接图

按 TRI 键输出三角波，按 SIN 键输出正弦波，按 SQU 键输出方波。

在输出三角波时，UP 用于增加上升沿 T1 或下降沿 T2 时间，DOWN 键用于减少上升沿 T1 或下降沿 T2 时间，SET 用于选择上升沿或下降沿。

在输出方波时，UP 用于增加高电平 T1 或低电平 T2 时间，DOWN 键用于减少高电平 T1 或低电平 T2 时间，SET 用于选择高电平或低电平。

在输出正弦波时，UP 用于增加正弦波的周期，DOWN 键用于减少正弦波的周期，在输出正弦波时请接入滤波电容，输出三角波和方波时不用接入滤波电容。

3. 程序说明

1）三角波

SPCE061A 具有两通道 10 位电流输出型的 DAC（AUD1、AUD2），每通道电流输出为 3mA，在输出端接电阻 R(R 值取 600Ω)，则输出的电流便转换为电压。

三角波的上升沿和下降沿都由 100 个小阶梯组成。主程序判断是上升沿还是下降沿，如果是上升沿，则将 TimerA 的定时值改为 t1；如果是下降沿，则将 TimerA 的定时值改为 t2，则三角波上升沿时间为 100*t1、下降沿时间为 100*t2。定时器 TimerA 的时钟源采用 12.288M，定时器初值计算：

M1=65535-(12288*T_High)/1000;

//t1 初值。定时器初始值＝FFFF-(定时时钟×上升沿时间)/1000

M2=65535-(12288*T_Low)/1000;

//t2 初值

DAC 数据的输出由中断程序完成。

2）方波：

在方波的高电平，DAC 输出最大值 0xFFC0，在低电平时 DAC 输出最小值 0，方波高、低电平的宽度由定时器 TimerB 定时值确定。方波的占空比为 D=t1/（t1+t2）。

3）正弦波：

正弦波是靠调用库函数 sinf（x）产生，当弧度在 0-2π之间变化时，sinf（x）的值在-1~1 的范围内，将 sinf（x）的值乘以 32736（0XffC0 的一半），即将波形放大并将 0 点偏移到 1.65V，波形对应 AD 值计算部分程序：

```
mx= mz * PI/2;                          //计算弧度值
my = sinf(mx);                          //计算 sinf(mx)
vv= (int) 32736 * my;                   //值
vv ^= 0x8000;                           //最高位取反
outdac(vv);                             //DA 输出
delay10ms(DelTime);                     //延时 t
mz=mz+0.02;                             //增量
if(((unsigned int)mz)== 4) mz=0;        //循环
```

当弧度在 0-π之间变化时 sin（x）* 32736 的值为正（最高位为 0），经 D/A 转换之后其波形如图 8.13中虚线 a 所示，当弧度在π-2π之间变化时 sin（x）* 32736 值为负(二进制补码表示，最高位为 1)，经 D/A 转换之后其波形如图 8.13中虚线 b 所示，将最高位取反后的 D/A 输出波形如图 8.13实线 A、B 所示。

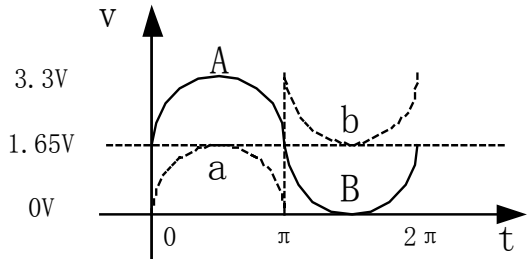

图8.13 正弦波

8.2.4 红外遥控

红外线遥控是目前使用最广的一种遥控手段。红外线遥控装置具有体积小、功耗低、功能强、成本低等特点，因而继彩电、录像机之后，在录音机、音响设备、空调机，以及玩具等其它小型电器装置上也纷纷采用红外线遥控。

1. 原理分析

1) 遥控指令编码规律

遥控器所发送的功能指令码一般采用多位二进制串行码,本例程序是海尔 29T6B-T 型彩色电视的红外遥控码，其编码规律为：头脉冲、系统码、资料码、资料反码和结束位。头脉冲用作一帧命令的起始位；系统码用于区别不同类的电器；资料码用于完成命令功能。海尔 29T6B-T 型彩色电视的系统码为 0x08，资料码见表 8.1，资料反码是将资料码按位取反的码。每次进行发送都是先发送脉宽 4510us、周期 2*4510us 的头脉冲，然后连续发送两次系统码、接着发送资料码及资料反码、最后发送结束位，波形见图 8.14。

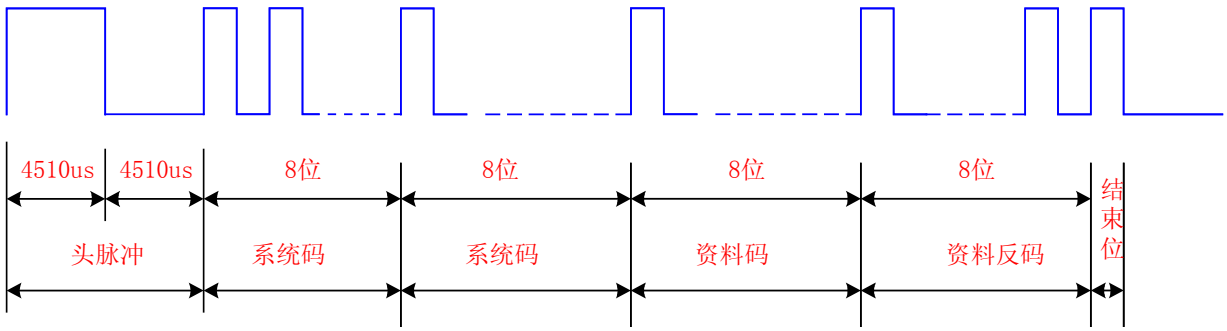

图8.14 遥控指令编码图

表8.1 遥控器资料码表

遥控功能	资料码	遥控功能	资料码
1	00	AV	0F
2	01	CH-	10
3	02	CH+	11
4	03	VOL-	12
5	04	VOL+	13
6	05	MUTE	14
7	06	SLEEP	15
8	07	DISPLAY	16
9	08	SMPX	17
0	09	MENU	1C
-/--/---	0A	SCAN	1E
POWER	0B	VOL M	2A
SYSTEM	0C		

2) 数据的脉冲编码

红外通讯数据采用脉冲编码，所谓脉冲编码，就是将每位数据信号用一个脉冲来表示。例子程序的红外编码以脉宽 561μs、周期 4*561μs 代表“1”；以脉宽 561μs、周期 2*561μs 代表“0”。脉冲信号都调制在占空比为 1/3，频率为 38kHz 的载波上再发送出去，调制后的信号“1”和“0”如图 8.15所示。这样做有两点好处：第一，减少了有效的发射时间，有利于降低平均功耗，这对于采用干电池供电的发射器十分重要；第二，外部干扰信号多为缓变信号，有利于抗干扰。

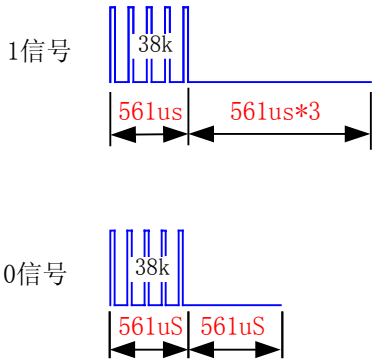

图8.15 信号 0 和 1

2. 硬件电路

系统由键盘电路和红外发射电路组成，利用 SPCE061A 的 IOA 口扩展了 4*8 键盘矩阵；发射电路中三极管 Q1（选用 8050）用于对信号放大，R4 选用 200Ω电阻，R5 选用 10Ω电阻，C5 选用 220μF，D1 为红外发射管。见图 8.16。

各个按键功能如表 8.2所示：

表8.2 按键功能表

按键	遥控功能	按键	遥控功能
S1	MUTE	S14	8
S2	AV	S15	9
S3	SLEEP	S16	SYSTEM
S4	POWER	S17	-/--/---
S5	1	S18	0
S6	2	S19	SMPX
S7	3	S20	DISPLAY
S8	SCAN	S21	CH+
S9	4	S22	MENU
S10	5	S23	CH-
S11	6	S24	VOL-
S12	VOL M	S25	VOL+
S13	7		

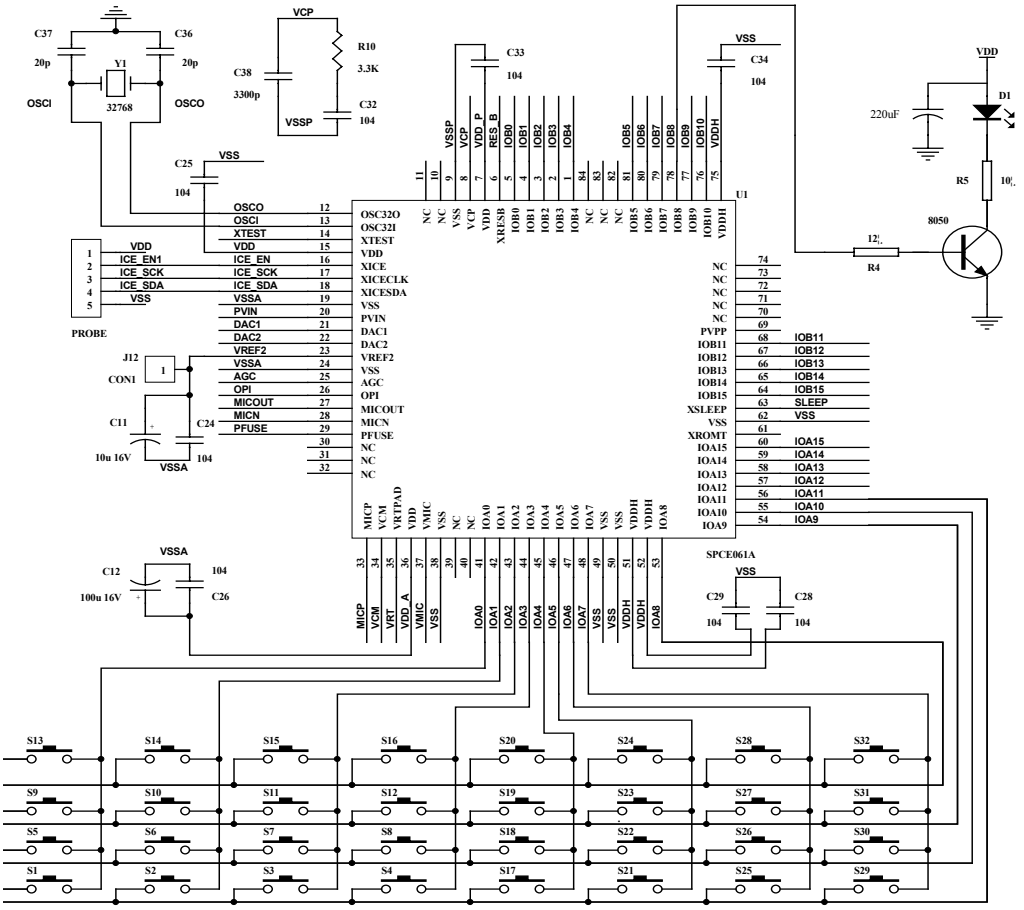

图8.16 红外线遥控装置硬件连接

3．程序设计

SPCE061A 的 IOA0-IOA7 设置为输入时具有按键唤醒功能，例子程序中将其设置为带下拉电阻的输入口，IOA8-IOA11 设置为带数据缓存器的输出口。键盘程序比较简单，此处不做详细介绍。串行码的发送主要用到 SPCE061A 的 TimerA 和 TimerB 两个定时器，IOB8 编程为第二功能时，可以由定时器 TimerA 控制输出占空比可调的脉宽调制信号 APWMO，38K 载波信号就是利用 TimerA 的 APWMO 输出产生，将 APWMO 信号频率设置为 38K，串行码为 1 时打开 APWMO 输出，为 0 时关闭 APWMO 输出（输出低电平）；用 TimerB 控制脉冲宽度，发射流程见图 8.17(a)，头脉冲、0 信号、1 信号的发射流程见图 8.17(b)。

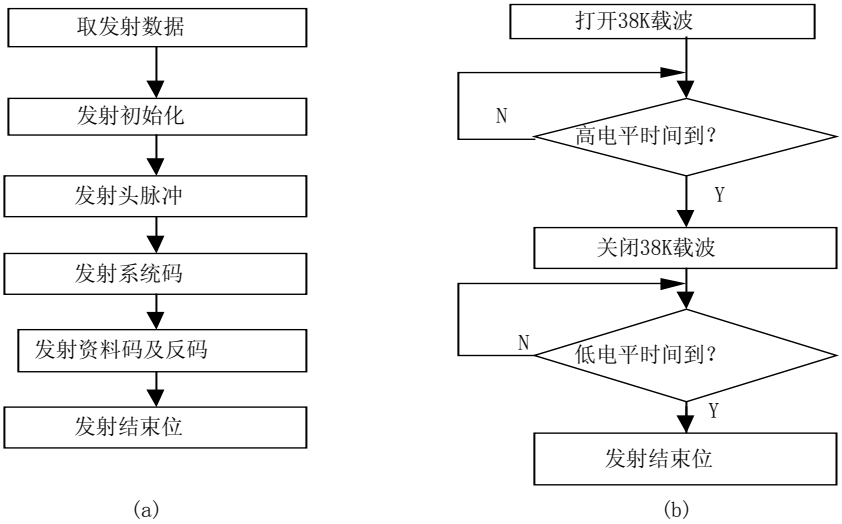

图8.17 程序流程图

8.2.5 SPCE061A 做语音录放

录音时，通过 A/D 转换器将语音信号转换成数字信号，编码后存入存储器中；放音时，将数据从存储器中取出并解码，然后经 D/A 转换变成语音信号输出。

1. 原理分析

1）录音

SPCE061A 的 A/D 转换器有 8 个通道，其中有 1 个通道是 MIC-IN 输入，它专门用于对语音信号进行采样。语音信号经 MIC 转换成电信号，由隔直电容隔掉直流成分，然后输入至 SPCE061A 内部前置放大器。SPCE061A 内部自动增益控制电路 AGC 能随时跟踪、监视前置放大器输出的音频信号电平，当输入信号增大时，AGC 电路自动减小放大器的增益；当输入信号减小时，AGC 电路自动增大放大器的增益，可使进入 A/D 的信号保持在最佳电平，又可使削波减至最小。

A/D 转换器对输入的音频信号进行 8kHz 采样，并按照凌阳音频编码格式进行编码，每秒将占用 16kBits 的存储器空间。系统扩展了一块容量为 1Mbits 的 SRAM 存储器 HM628128A 来存储语音数据。

2）放音

将 HM628128A 中存储的语音数据顺序取出，解码后，以 8kHz 的速率进行 D/A 转换输出，经电容滤波后，恢复原始语音波形，通过三极管驱动扬声器放音。

2. 硬件电路

HM628128A 与 SPCE061A 的连接电路见图 8.18，由 SPCE061A 的 IO 口完成地址线、数据线和控制线扩展功能。系统扩展了三个按键，REC 用于开始录音，PLAY 用于放音，STOP 用于停止录音或放音。

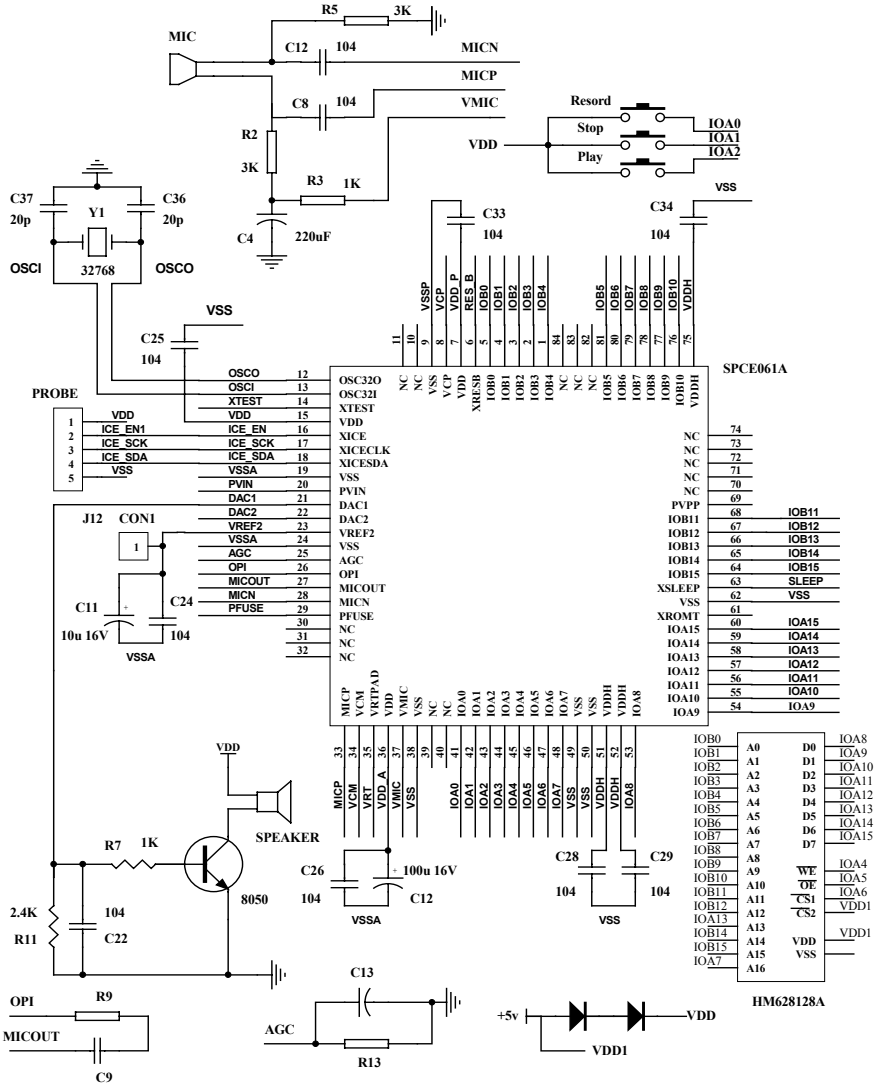

图8.18 HM628128A 与 SPCE061A 的硬件连接图

录放音的编解码是靠调用库函数完成的，我们只要完成数据的存入和读出即可。

主程序流程图见图 8.19。

程序部分主要是完成 HM628128A 的读写，对语音采样数据的编码处理是通过调用库函数完成的。写（读）程序都包括初始化和写（读）两部分，写入（读出）程序较简单，只要按照 HM628128A 的写（读）时序就可以，下面讲解初始化和写入时的参数传递过程，读出时参照写入部分。

```
_SP_InitWriteSRAM:          .PROC
F_SP_InitWriteSRAM:
        PUSH R1 TO [SP]

        R1 = 0xFFF0
        [P_IOA_Dir] = R1          // IOA8---IOA15 接数据线，写入时设置为同相高电平输出
        [P_IOA_Attrib] = R1       // IOA4--IOA7 接 SRAM 控制线（低电平有效），
                                  //设置为同相高电平输出
        [P_IOA_Data] = R1         // IOA0--IOA3 接键盘，设置为带下拉电阻的输入

        R1 = 0xFFFF
        [P_IOB_Dir] = R1          // IOB15--IOB0 接 SRAM 地址线，设置为输出
        [P_IOB_Attrib] = R1       //
        [P_IOB_Data] = R1         //

        POP R1 FROM [SP]
        RETF
        .ENDP
```

初始化包括地址线、数据线和控制线的设置。IOB15---IOB0、IOA7 接地址线，设置为输出，IOA15---IOA8 接数据线，写入时设置为输出，IOA6—IOA4 接选通信号，设置为输出。

```
_SP_InitReadSRAM:           .PROC
F_SP_InitReadSRAM:
        PUSH R1 TO [SP]
        R1 = 0x00F0
        [P_IOA_Dir] = R1       //IOA8---IOA15 接数据线，读出时设置为带下拉电阻的输入
        [P_IOA_Attrib] = R1    //IOA4--IOA7 接 SRAM 控制线（低电平有效）
                               //设置为同相高电平输出
        R1 = 0x00F0            //IOA0--IOA3 接键盘，设置为带下拉电阻的输入
        [P_IOA_Data] = R1

        R1 = 0xFFFF
        [P_IOB_Dir] = R1       //IOB15--IOB0 接 SRAM 地址线，设置为同相高电平输出
        [P_IOB_Attrib] = R1
        [P_IOB_Data] = R1


        POP R1 FROM [SP]
```

```
RETF
.ENDP
```

由于HM628128A有17根地址线,在主程序中记录地址的变量设置为长整型数据(32位），在调用写入子程序时，先将高 16 位压入堆栈，再压入低 16 位，所以在写入子程序中取两次地址（高、低），并判断高位是否为 0，不为 0，则将 A16 置 1，表明读、写的是高 64K RAM。读者只要对上面电路、程序稍加改动，就可以实现语音复读机、跟读机、留言机等功能。

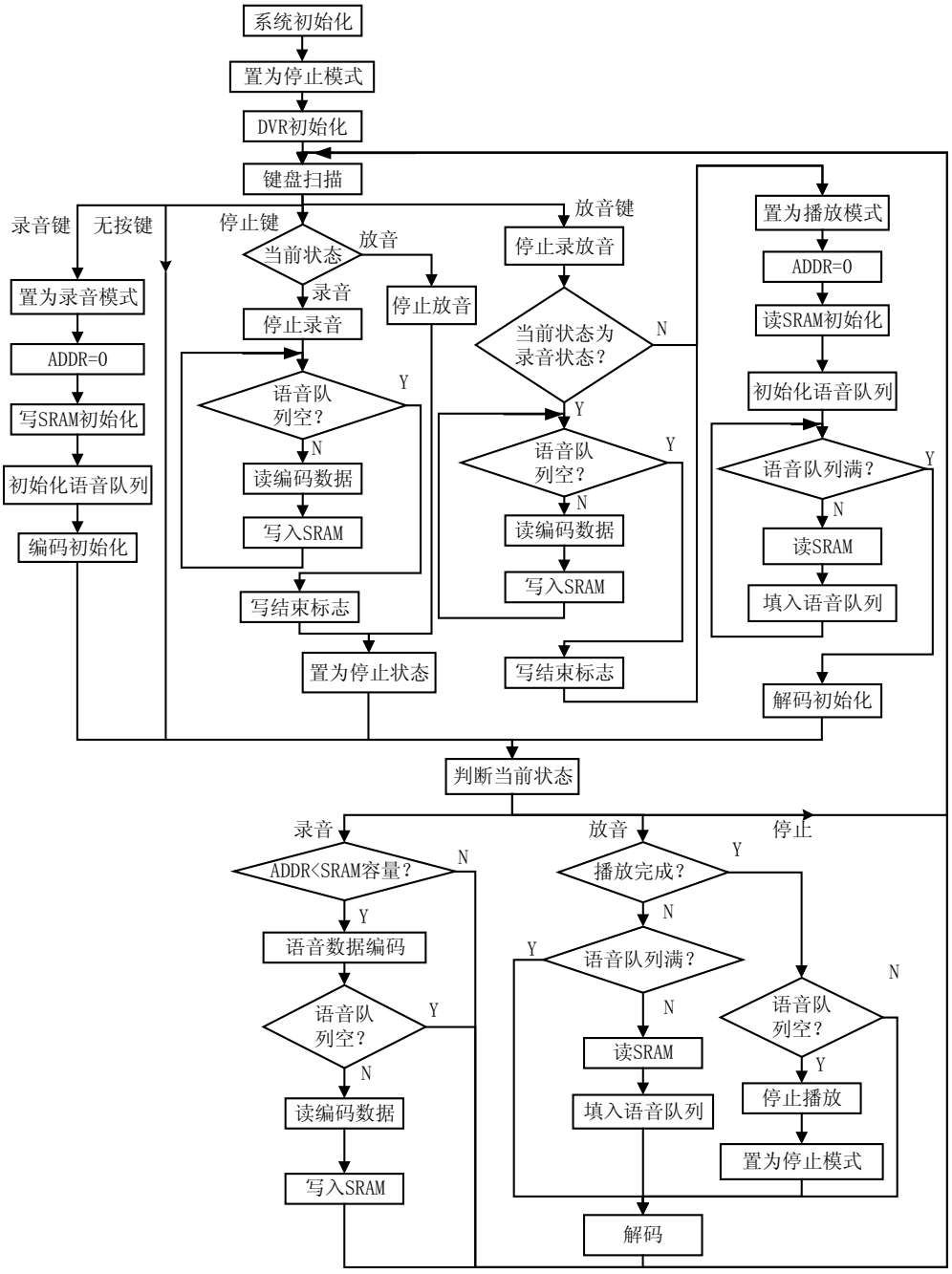

图8.19 程序流程图

8.2.6 语音识别

1．语音识别原理

语音识别电路基本结构如图 8.20所示：

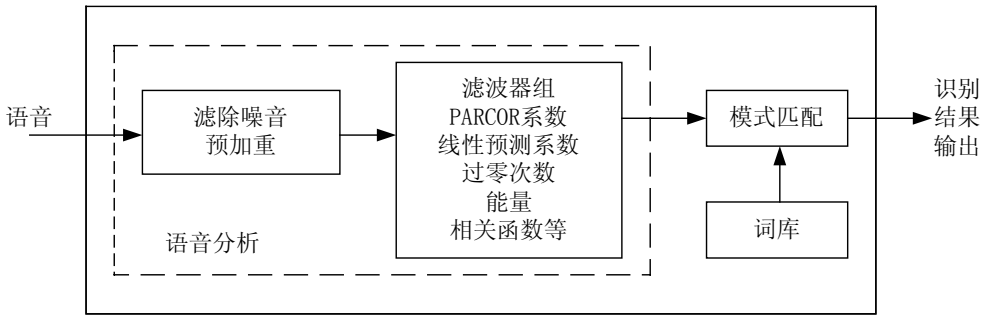

图8.20 语音识别电路基本结构

语音识别分为特定发音人识别（Speaker Dependent）和非特定发音人识别（Speaker Independent）两种方式。

特定发音人识别是指语音样板由单个人训练，对训练人的语音命令识别准确率较高，而其他人的语音命令识别准确率较低或不识别。非特定发音人识别：是指语音样板由不同年龄、不同性别、不同口音的人进行训练，可以识别一群人的命令。语音样板的提取非常重要。例子程序就是采用特定发音人识别方式。

我们将标准模式的存储空间称之为“词库”，而把标准模式称之为“词条”或“样板”。所谓建立词库，就是将待识别的命令进行频谱分析，提取特征参数作为识别的标准模式。

识别过程首先要滤除输入语音信号的噪音和进行预加重处理，提升高频分量，然后用线性预测系数等方法进行频谱分析，找出语音的特征参数作为未知模式，接着与预先存储的标准模式进行比较，当输入的未知模式与标准模式的特征相一致时，便被机器识别，产生识别结果输出。如果输入的语音与标准模式的特征完全一致固然好，但是语音含有不确定因素，完全一致的条件往往不存在，事实上没有人能以绝对相同的语调把一个词说两遍，因此，预先制定好计算输入语音的特征模式与各特征模式的类似程度，或距离度的算法规则固化在 ROM 中，把该距离最小，即最类似的模式作为识别相应语音的手段。当然，影响识别率的因素还有一些，如连续发音（如英语）与

断续发音（如汉语）的不同（二者区别在于单词间隔有 200ms 以上的空隙时间）。

例子程序采用特定人识别方式，将训练的标准样板存于内部 RAM 中（掉电丢失），每次上电复位后都要进行训练，用户可以扩展一块 FLASH，将训练的标准样板存于 FLASH，这样就不需要在每次上电复位后再次训练。

2．硬件电路

硬件电路比较简单，MIC 选用驻极体电容话筒，这种话筒具有灵敏度高、无方向性、重量轻、体积小、频率响应宽、保真度好等优点，驻极体话筒的偏压由 SPCE061A 的 VMIC 管脚提供。如图 8.21所示。

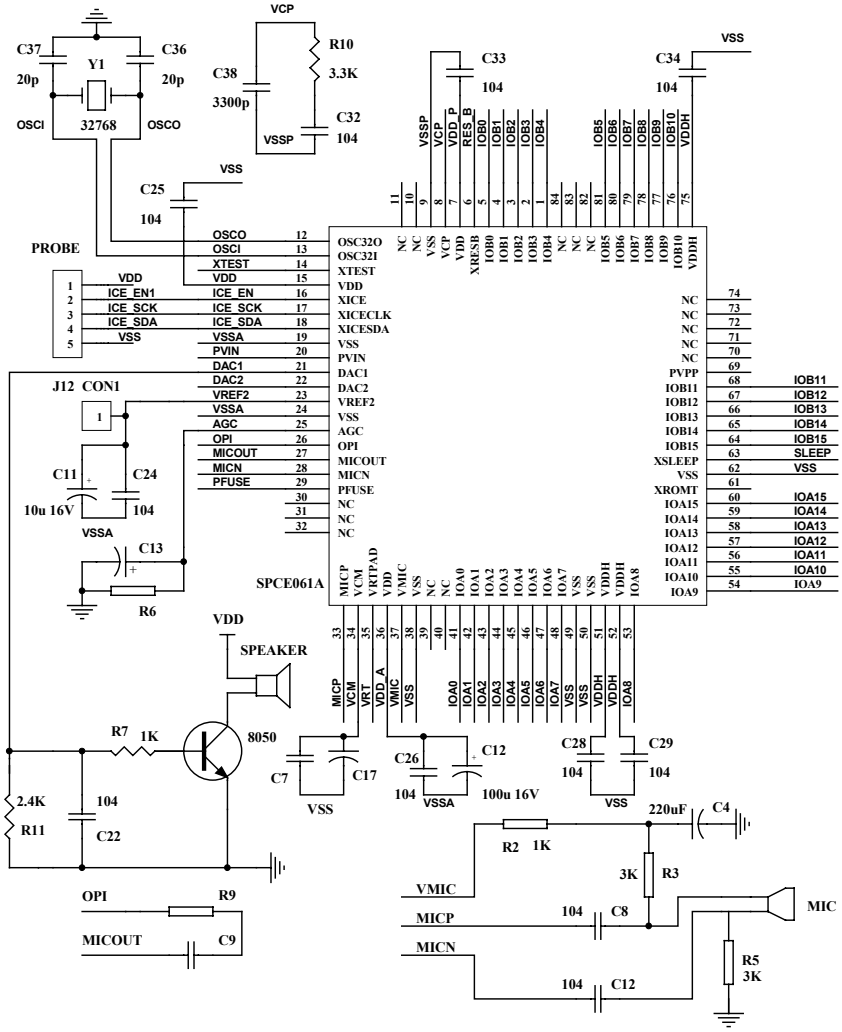

图8.21 特定人语音识别硬件连接图

3．程序设计

程序包括三部分：训练样本、识别和语音提示。

由于语音样本是存在内部 RAM 中，掉电将丢失，所以在每次上电复位时都必须重新训练，训练过程主要是靠调用库函数 BSR_Train 来完成，为了防止误命令，每条语音命令训练 2 遍，只有 2 次命令相同时才成功，BSR_Train 函数有 8 种可能的返回值：

0------训练成功；

-1------没有检测到命令；

-2------需要再训练一次，每条命令训练 2 次，第一次训练成功则返回-2；

-3------环境太吵；

-4------存储器满；

-5------两次命令不一样；

-6------命令序号超出范围；

-7------命令已存在；

训练成功则训练下一条，否则继续训练。

语音识别程序包括识别程序和中断服务程序。识别程序完成选取词库、初始化 A/D 和定时器 TimerA、识别运算及识别结果处理，流程如图 8.22。中断服务程序定时读取 A/D 转换结果，并存入缓冲区，A/D 的输入为 MIC 通道的语音信号。语音识别和放音分时复用 TimerA FIQ 中断，由标志位判断是语音识别处理还是放音处理。中断服务程序的流程图如图 8.23所示。

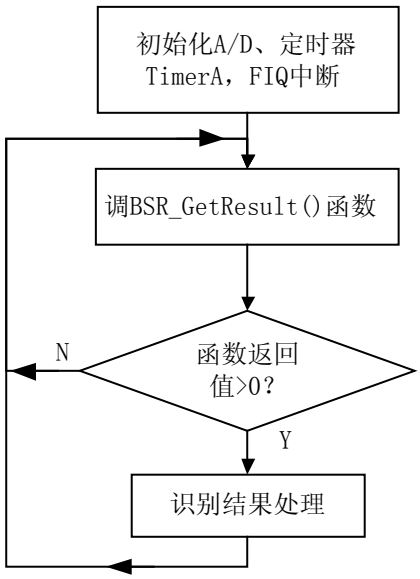

图8.22 特定人语音识别主流程图

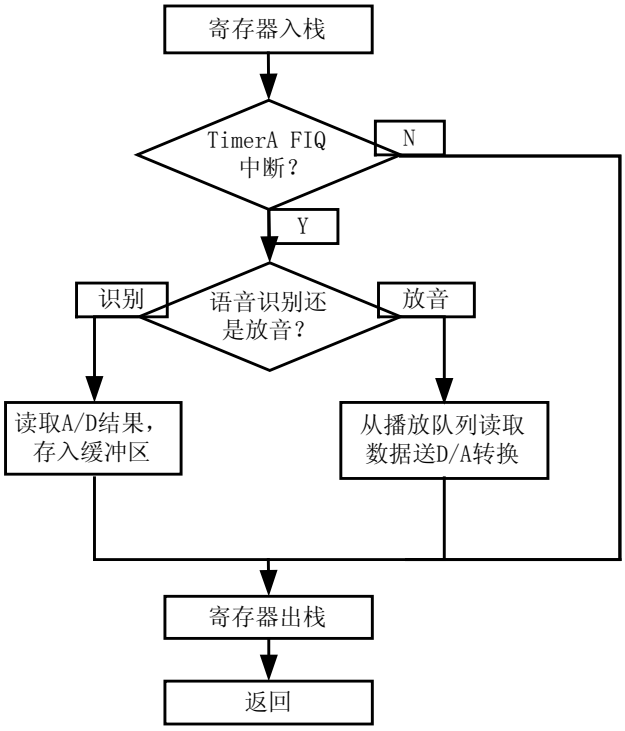

图8.23 特定人语音识别中断流程图

识别程序如下：

```
BSR_InitRecognizer(BSR_MIC);
//初始化识别器，包括 AGC、ADC、TimerAFIQ 中断
while(1)
{
res = BSR_GetResult();
//主识别函数，识别成功则结果为语音命令的顺序号
if(res > 0)
//结果>0，表明识别成功，相应处理
}
```

中断服务程序如下：

```
_FIQ:
PUSH R1,R4 TO [SP]                    //寄存器入栈
R1 = [P_INT_Ctrl]                     //读中断标志位
R1 &= 0x2000
JZ  FIQ_ret                           //不是 TimerA  FIQ 中断
R1 = [__gIsStopRecog]
JNZ TimerA_is_S480
```

```
TimerA_is_Recognize:                    //TimerA  FIQ 为语音识别服务
call _BSR_FIQ_Routine                   //语音识别服务函数
JMP FIQ_ret
TimerA_is_S480:                         //TimerA  FIQ 为放音服务
CALL F_FIQ_Service_SACM_S480;           //放音服务函数
FIQ_ret:
R1 = 0xa800;
[P_INT_Clear] = R1;                     //清中断标志
POP R1,R4 FROM [SP];                    //寄存器出栈
reti;
```

8.3 数字滤波程序

8.3.1 μ'nSP 实现 FIR 滤波：乘-累加（MAC）功能

FIR 滤波器响应的复杂性需每个采样周期进行数字 MAC 操作。μ'nSP™的乘法累加求和指令（Muls）之格式如下：

MR = [Rd] * [Rs] {, ss}{, n};

MR = [Rd] * [Rs] , us {, n};

式中：

MR 为用于累加的寄存器对，由 R3、R4 充当；

Rd 为目标寄存器，在此用于采样数据指针；

Rs 为源寄存器，在此用于系数寄存器指针；

n 为参加滤波运算的采样样本数目；

ss 为有符号数相乘的设定，缺省设定即为此；us 为有符号数与无符号数相乘的设定。

假设参加滤波的采样样本数为 4，且由采样数据指针 R1 和系数寄存器指针 R2 分别指向的存储器内容在执行 Mul 指令之前如图 8.24(a)所示。

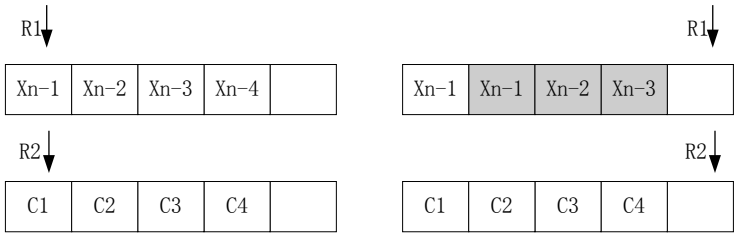

图8.24 Muls 指令执行前后指针位置以及存储器内容
(a) 指令执行前 (b) 指令执行后

那么，当执行了指令 MR = [R1] * [R2]，4 以后，会产生如下一些动作：

1）累加器 MR 清零后进行 MAC 计算：

$MR = C_1 * X_{n-1} + C_2 * X_{n-2} + C_3 * X_{n-3} + C_4 * X_{n-4}$;

且指针向右移动了 n（n=4）个字的位置，如图 8.24 (b)所示。

2）值得注意的是，指针 R1 指向的存储器的内容（采样数据样本）向前移动了一个字的位置。其意义在于当采样新的数据样本（Xn）时，可将 Xn 依序存放在 X_{n-1} 之后，而最旧的数据样本（X_{n-4}）会被次旧的数据样本（X_{n-3}）取代，如图 8.24所示。

Muls 指令只占一个字（即 16 位）的存储空间且指令执行的时间仅为（10n+6）个时钟周期，具体到本例是 10*4+6 = 46 个时钟周期。

8.3.2 用μ'nSP™实现低通 FIR 滤波器

FIR 滤波器维护一个固定数目 n 的最新样本表。每次 FIR 迭代前，新样本加入表中，如图 8.24所示。FIR 迭代后，最旧的样本被替代，见图 8.24(b)。此表表示样本向量，它与系数向量相乘得到当前输出。

设我们要实现一个 16 阶 FIR 低通滤波器。采样频率为 $f_{sample} = 32k$，截止频率为 $f_{cut} = 2k$。

算法：采用有限项傅氏级数计算滤波系数。此算法属于非窗口化算法，

阶数：$N = 16$

归一化频率为：$\lambda = \pi \dfrac{f_{cut}}{f_{sample}/2}$

系数计算公式为：$h[k] = \dfrac{\sin(m\lambda)}{m\pi}$，其中 $m = k - \dfrac{N-1}{2}$

这样可以计算 16 个系数为：

h[0]=h[15]=0.00827989
h[1]= h[14]=0.027206692
h[2]= h[13]=0.048120909
h[3]= h[12]=0.069376367
h[4]= h[11]=0.089198186
h[5]= h[10]=0.105865999
h[6]= h[9]=0.117895665
h[7]= h[8]=0.124198356

由于μ' nSP™ 适于做定点 DSP 运算，故系数必须以整数的形式存储在 ROM 中。所

以我们将每个系数放大 100 倍后，进行四舍五入取整得到如下系数：

h[0]=h[15]=1
h[1]= h[14]=3
h[2]= h[13]=5
h[3]= h[12]=7
h[4]= h[11]=9
h[5]= h[10]=11
h[6]= h[9]=12
h[7]= h[8]=12

于是我们可以定义如下系数表：

```
.DATA
h0:     .dw  1;
h1:     .dw  3;
h2:     .dw  5;
h3:     .dw  7;
h4:     .dw  9;
h5:     .dw  11;
h6:     .dw  12;
h7:     .dw  12;
h8:     .dw  12;
h9:     .dw  12;
h10:    .dw  11;
h11:    .dw  9;
h12:    .dw  7;
h13:    .dw  5;
h14:    .dw  3;
h15:    .dw  1;
```

下面建立一个变量序列：

```
.ISRAM
x0:     .dw  1;
x1:     .dw  1;
x2:     .dw  1;
x3:     .dw  1;
x4:     .dw  1;
x5:     .dw  1;
x6:     .dw  1;
x7:     .dw  1;
x8:     .dw  1;
x9:     .dw  1;
x10:    .dw  1;
x11:    .dw  1;
x12:    .dw  1;
```

```
x13:    .dw 1;
x14:    .dw 1;
x15:    .dw 1;
```

FIR1 滤波程序:

```
//*********************************************************************************************/
.DEFINE P_INT_Ctrl_New          0x7010
.DEFINE P_ADC_MUX_Ctrl          0x702b
.DEFINE P_ADC_MUX_Data          0x702c
.INCLUDE hardware.inc
.PUBLIC _main
.CODE

//=============================================================================
// 函数： main()
// 描述：主函数
//=============================================================================
_main:
    INT OFF;                    //关中断，以便进行初始化
    R1 = 0x00f0;                //系统时钟设定,Fosc=49.152MHz,CPUCLOCK=Fosc
    [P_SystemClock] = R1;
    R1 = 0x0030;                //TimerA 设定,clckA 选择 Fosc/2;屏蔽 ClockB
    [P_TimerA_Ctrl] = R1;
    R1 = 0xfd00;
    [P_TimerA_Data] = R1;       //32kHz 中断频率
    R1 = 0;
    [P_DAC_Ctrl] = R1;          //DA 设定,直接将 DAR 锁存到 DAC；□
    R1 = 0x1;
    [P_ADC_Ctrl] = R1;          //通过读 P_ADC 触发 ADC 转换,AD 设定,允许数模转换
    R1 = 0x1;
    [P_ADC_MUX_Ctrl] = R1       //选择 Line_IN1 输入
    R1 = 0x2000;
    [P_INT_Ctrl_New] = R1;      //中断设置,允许 timeA 的 FIQ 中断
    R1 = x0;                    //数据指针复位
    R2 = h0;                    //系数指针复位
    FIQ ON;                     //开中断
L_Loop:
   NOP;
   GOTO L_Loop;                 //等中断

//=============================================================================
//函数: FIQ()
//语法：void FIQ(void )
//描述：FIQ()中断服务子程序
```

```
//参数：无
//返回：无
//============================================================================
.TEXT
.PUBLIC _FIQ
_FIQ:
    R3 = [P_ADC_MUX_Data];          //取 AD 采样数据
    [R1] = R3;                      //更新
    MR = [R1]*[R2],US,16            //FIR 滤波运算
    R4 = R4 LSR 4;
    R3 = R3 ROR 4;
    R4 = R4 LSR 4;
    R3 = R3 ROR 3;                  //取计算结果最高有效的 16 位
    [P_DAC1] = R3;                  //刷新 DA 转换
    R1 = 0x2000;
    [P_INT_Clear] =R1;              //清中断
    R1 = x0;                        //数据指针复位
    R2 = h0;                        //系数指针复位
    RETI;
//*************************************************************************************************/
// main.asm 结束
//*************************************************************************************************/
```

现在我们从 1kHz 的方波信号中去掉 3 次以上的谐波（包括 3 次谐波），得到 1kHz 的正弦波。对于方波信号展开傅立叶级数表达式如下。方波的频谱关系图如图 8.25所示。

$$v(t) = \frac{V_s}{2} + \frac{2V_s}{\pi}(\sin\ \omega_0 t + \frac{1}{3}\sin\ 3\omega_0 t + \frac{1}{5}\sin\ 5\omega_0 t + \cdots)$$

式中，$\omega_0 = \frac{2\pi}{T}$，$\frac{V_s}{2}$ 是方波信号的直流分量，$\frac{2V_s}{\pi}\sin\ \omega_0 t$ 称为该方波信号的基波，它的周期 $\frac{2\pi}{\omega_0}$ 与方波本身的周期相同。式中其余各项都是高次谐波分量，它们的角频率是基波角频率的整数倍。

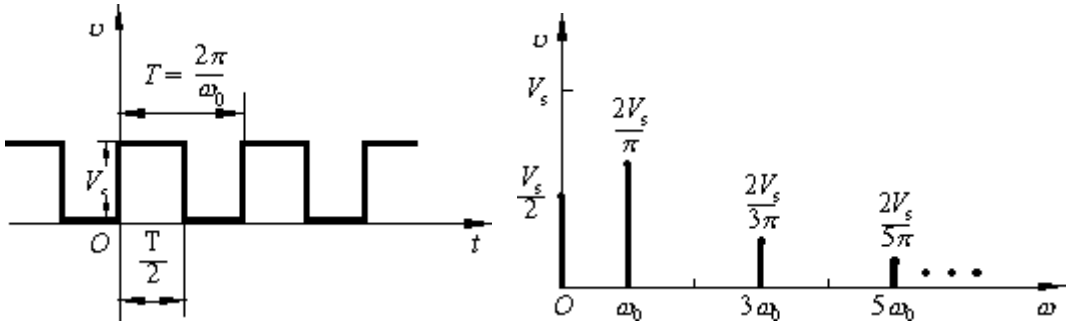

图8.25 方波频谱关系

由于 SPCE061A 的 DA 输出为电流型输出，故我们需要加入一个 1k 的电阻，把电流信号转换为电压信号，电路图如图 8.26所示：

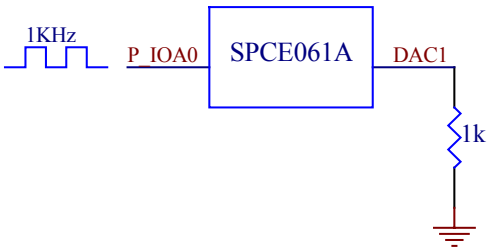

图8.26 FIR 滤波试验电路原理图

此时，在 DAC1 输出的波形将是一个 1kHz 的正弦波。

8.3.3 μ'nSP 实现 FIR 滤波需要注意的问题

1) 系数 h[k]的整数化
由于通过公式计算出的系数往往是小于 1 的小数，所以需要对其进行适当的放大并进行取整处理。

2) 滤波运算的符号问题
因为滤波运算有两种方式：mr=[r1]*[r2],us,n（有符号数与无符号数相乘）和 mr=[r1]*[r2],ss,N（有符号数与有符号数相乘）。那么如果系数 h[k]均为正数，运算较为简单，使用参数 us 即可得到正确结果。如果系数 h[k]出现了负数，那么就需要考虑把 AD 采样的数据处理为有符号数的整数（如果 AD 结果最高位可能出现 1，可以通过右移一位，确保最高位为 0；如果信号幅值小，AD 结果最高位一定为 0，则无需进行转换），再进行 ss 滤波运算，才能得到正确结果。此时还涉及到 DA 输出的数据是无符号数，在实际应用中应根据情况进行灵活处理。

3) 滤波运算结果
由于滤波运算结果是 32 位的有符号数，如果结果是大于 0 的非溢出的数据，我们则根据其出现的最高有效位进行移位处理，选择最高有效的 16 位作为真正的输出。但是如果结果出现了负数，那么我们还要进行加一个偏移量，将结果全部转换为非溢出正数，再提取高 16 位有效数据。如果结果溢出，那么就要调整系数的放大倍数，使得滤波运算的结果落在合适的范围。

8.3.4 滤波系数出现负数时的滤波运算

当采样频率为 32kHz，截止频率为 4kHz，计算出的系数将出现负值：

h[0]=h[15]= -0.016241589
h[1]= h[14]= -0.045243075
h[2]= h[13]= -0.053469089
h[3]= h[12]= -0.027069316
h[4]= h[11]= 0.034803406
h[5]= h[10]= 0.1176319969
h[6]= h[9]= 0.196053326
h[7]= h[8]= 0.24362384

对系数进行放大 50 倍后取整得到：

h[0]=h[15]=-1
h[1]= h[14]=-2
h[2]= h[13]=-3
h[3]= h[12]=-1
h[4]= h[11]=2
h[5]= h[10]=6
h[6]= h[9]=10
h[7]= h[8]=12

系数出现负值时，中断程序中滤波运算代码：

```
_FIQ:
    R3 = [P_ADC_MUX_Data];   //取 AD 采样数据
    R3 = R3 LSR 1;
    [R1] = R3;               //更新
    MR = [R1]*[R2],SS,16     //FIR 滤波运算
    R3 += 0xffff;
    R4 += 0x3,Carry          //数据平移为正数
    R4 = R4 LSR 4;
    R3 = R3 ROR 4;
    R4 = R4 LSR 4;
    R3 = R3 ROR 1;           //取计算结果最高有效的 16 位
    [P_DAC1] = R3;           //刷新 DA 转换
    R1 = 0x2000;
    [P_INT_Clear] =R1;       //清中断

    R1 = x0;                 //数据指针复位
    R2 = h0;                 //系数指针复位
    RETI;                    //系数指针复位
```

8.4 卷积编码以及数字比特译码

8.4.1 卷积码编码和维特比译码

在大多数无线通讯系统中，卷积码是克服通信信道中传输错误的优选的前向纠错编码方法之一，维特比译码是具有最佳译码性能的卷积码译码方法。

卷积码是将发送的信息序列通过一个线性的、有限状态的移位寄存器而产生的码。通常，该移位寄存器由 K 级（每级 k 比特）和 n 个线性的代数函数生成器组成，如图 8.27所示。二进制数据移位输入到编码器，沿着移位寄存器每次移动 k 比特位。每个 k 比特长的输入序列对应一个 n 比特长的输出序列。卷积码通常记作（n, k, K），它的编码效率定义为 $Rc=k/n$，参数 K 称为卷积码的约束长度。

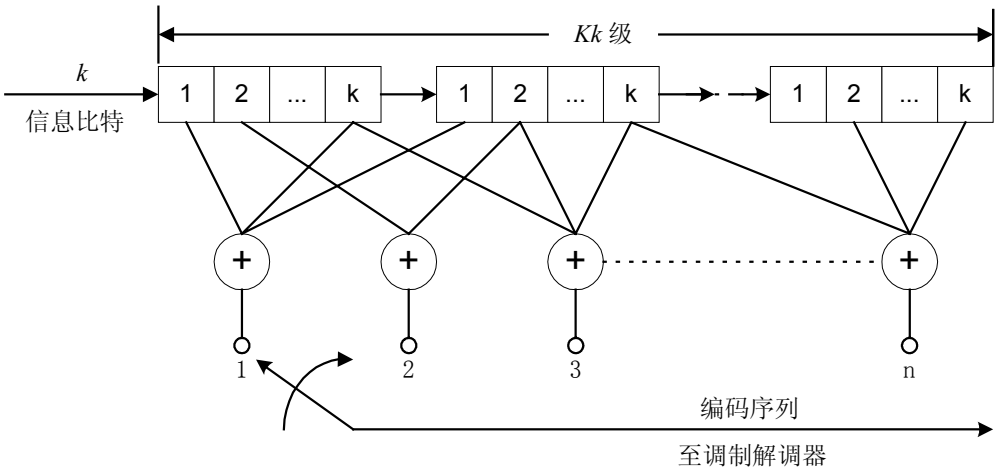

图8.27 卷积编码器

编码器中移位寄存与模二和的连接关系可以用 n 个函数生成矢量来表示，每个矢量对应 n 个模二加法器中的一个，每个矢量有 Kk 维，包含编码器和模二加法器之间连接关系的信息。某矢量第 i 个元素如果是“1”，表示相应的移位寄存器第 i 级和模 2 加法器相连；反之，如果在该位置上为 0，表明这一级移位寄存器和模二加法器不相连。以（2，1，3）卷积码为例，见图 8.28，它的函数生成矢量为

$\mathbf{g}_1$=[1 1 1]

$\mathbf{g}_2$=[1 0 1]

可用八进制方式更简洁地表示为（7，5）。

对于给定输入序列，利用函数生成矢量可以计算出编码后的输出序列。

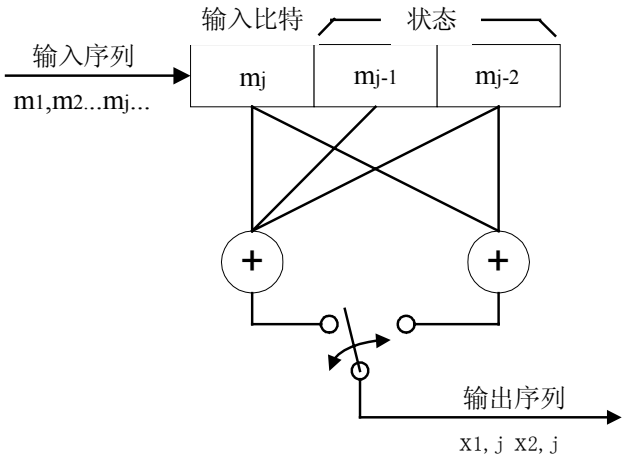

图8.28 K=3, k=1, n=2 卷积编码器

卷积码的编码过程用伪码表示如下：

```
Allocate the output buffer                          //分配输出序列缓存区
Initialize the shift register                       //初始化移位寄存器
Add m flushing bits to the end of input frame, here m=Kk-1
//将编码器刷新的 m 位加到输入帧的结束比特位（LSB），此处 m= Kk-1
for each input frame                                //每输入一帧
{
    for each input k bits                           //每输入 k 比特位
    {
        Update the shift register (the oldest k bits are shifted out, and the new k bits
        are shifted in)
        //更新移位寄存器（旧的 k 比特位被移出，新的 k 比特位被移入）
        Caculate n output bits (mod-two adder output)according the generative vectors an
        d shift register
        //据函数生成矢量和移位寄存器计算编码器 n 位输出（即模二加法器输出）
        Put the output n bits into the output buffer
        //将计算出来的 n 位输出置入输出缓存区
    }
}
```

卷积码是根据可能的状态转移进行译码的。所有可能的码序列限制在由状态转移得到网格图中。图 8.29表示了（2，1，3）卷积码的网格图，图中四行小圆圈表示移位寄存器的四种状态（一般情况下可能会有 $2^{k(K-1)}$ 种状态）。实线箭头表示输入为 0 时的状态转移，虚线箭头表示输入为 1 时的状态转移，支路上标注的码元为输出比特。

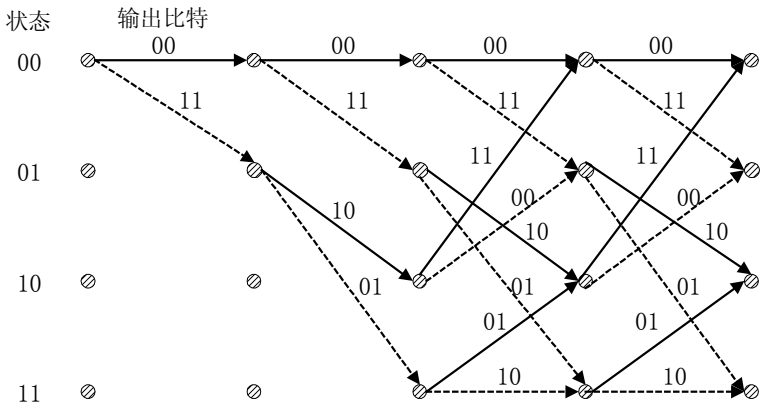

图8.29 （2，1，3）卷积码的网格图表示

譬如假设移位寄存器处于初始状态 00，即图 8.28中的 $m_{j-2}m_{j-1}$=00；当第一个输入比特 m_1=0 时，输出比特 $x_{1,1}x_{1,2}$=00；若 m_1=1，则 $x_{1,1}x_{1,2}$=11。因此，从初始状态出发有两条支路供选择，m_1=0 时取实线支路，m_1=1 则取虚线支路。输入第二个比特时，移位寄存器右移一位后，实线支路情况下移位寄存器的状态仍为 00，虚线支路的状态则为 01。新的一位输入比特到来时，随着移位寄存器状态和输入比特的不同，又可继续分叉成四条支路，二条对应于输入比特为 0 的实线支路和二条对应于输入比特为 1 的虚线支路。如此继续下去，即可得到图 8.29所示的网格图形。

对接收到的卷积编码序列译码时，利用网格图寻找最大可能的输入码序列作为译码输出。维特比译码算法通过限制搜索的序列数目简化了译码过程，对每个新输入的符号只保留转移到各个状态的最可能路经。

如前所述，卷积码网格图中在一般情况下可能会有 $2^{k(K-1)}$ 种状态，每个状态节点有 2^k 条支路引入亦有 2^k 条支路引出。为简便起见，只讨论 k=1 的情况，从全 0 状态起始点来讨论。由网格图的前 $K-1$ 条连续支路所构路径互不相交，即最初的 $2^{(K-1)}$ 条路径各不相同，当进行到第 K 条支路时，每条路径都有两条支路延伸到第 K 级上，而第 K 级上的每两条支路又都汇集在一个节点上。维特比译码算法是把聚集在每个节点上的两条路径的对数似然函数累加值进行比较，而后选择具有较大对数似然函数累加值（或具较小汉明距离）的路径，丢弃另条路径，结果到第 K 级只留下 $2^{(K-1)}$ 条“幸存”路径。显而易见，上述译码过程中的基本操作是加-比-选（ACS，Add-Compare-Select），即每级求出对数似然函数累加值，然后将两两比较后作出选择。当出现两条路径的对数似然函数累加值相等的情形时，可任意选择其中一条作为“幸存”路径。

作为最大似然译码器，维特比算法的实质是根据接收到的符号序列计算出最大概率匹配发送序列的码序列作为译码输出。维特比算法由两步构成：①路径度量的计算与更新(metric updata)，即 ACS；②回溯(traceback)。

路径度量的计算在硬判决（对接收信号进行 1 比特量化）时采用汉明距离量度，

而在软判决（对接收信号进行多比特量化）时采用欧氏距离或绝对值距离量度。

对于软判决输入，计算支路度量时的欧氏距离（对于码率为 1/C 时）定义为

$$d(j)=\sum_{i=0}^{C-1}[SD_i-G_i(j)]^2$$

绝对值距离定义为：

$$d(j)=\sum_{i=0}^{C-1}\left|SD_i-G_i(j)\right|$$

这里，SD_i为软判决输入，$G_i(j)$为每条支路的期望输入，j是支路标号。回溯时取译码约束长度为编码约束长度（K）的 3~5 倍。

维特比译码过程用伪码表示如下：

```
for each frame:
{
    Initialize metrics                          //量度初始化
    for each symbol                             //每接收一个符号序列
    {
        //Metric Update or Add-Compare-Select (ACS)   ；加-比-选
        For each delay state                    //每一延迟状态
        {
            Calculate local distance of input to each possible path
            //计算输入符号的每条可能路径的局部汉明距或欧氏距
            Accumulate total distance for each path
            //累加局部距离得出每条路径的总距离
            Select and save minimum distance
            //选择并存储最小距离
            Save indication of path taken
            //存储选用路径的轨迹
        }
    }
    //Traceback                                 ；回溯
    for each bit in a frame (or for minimum # bits)
    //对一帧里的每一位
    {
        Calculate position in transition data of the current state
        //计算当前状态下转换数据的位置
        Read selected bit corresponding to state
        //依据状态读出被选择的比特位
        Update state value with new bit
        //用新比特位来更新状态值
    }
    //because the LSB is sent first, reverse the order of output bits
    //由于编码数据帧的最低有效位(LSB)是先被传输的,故译码输出的数据位顺序反回
```

```
    reverse output bit ordering
    //反回译码输出的数据位顺序
}
```

8.4.2 用μ'nSP™实现卷积编译码

根据卷积码的编译码算法，用 μ'nSP™的汇编语言编写出码率为 1/2 的卷积编码及维特比译码程序，采用 3 比特软判决，距离量度采用绝对值距离量度。

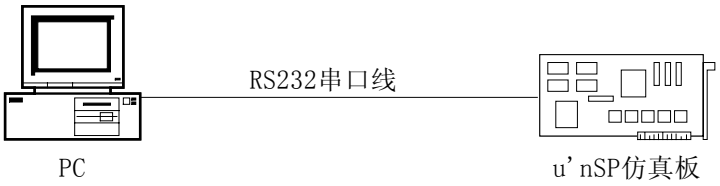

图8.30 卷积编译码通信实验框图

为了测试卷积编译码算法，可设计一个串口通信实验，实验框图如图 8.30：

本实验通过 PC 机把卷积编码后的文件发送到 μ'nSP™仿真板，在仿真板上译码后回送给 PC。通过这个实验可显示出 μ'nSP™在通信领域中的应用。

程序流程图如图 8.31和图 8.32 所示：

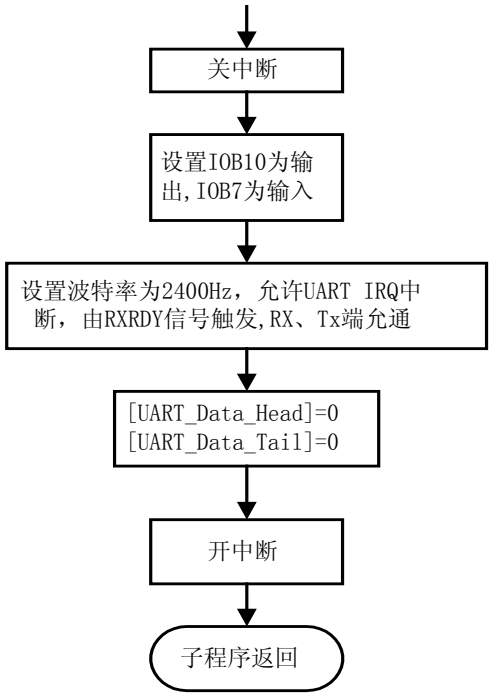

图8.31 串口设置流程图

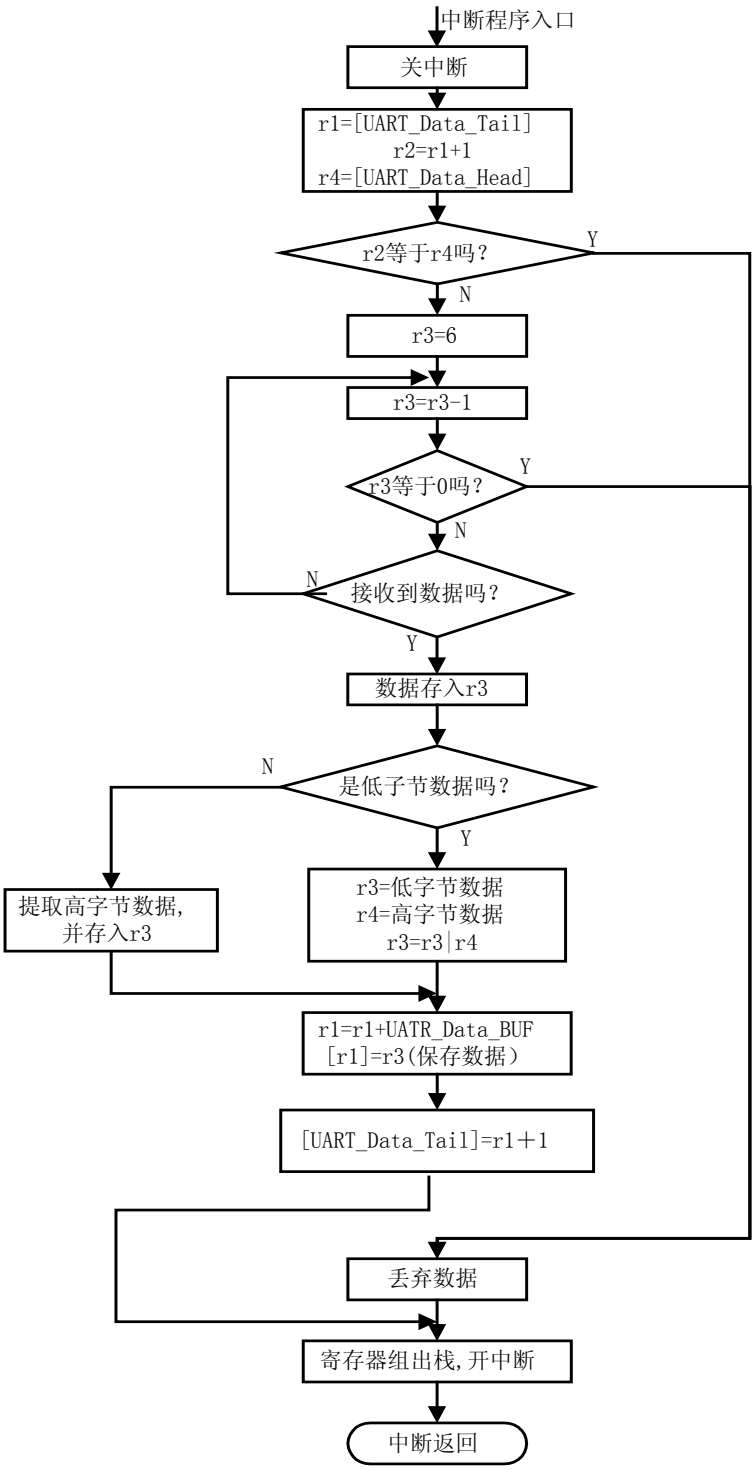

图8.32 中断服务程序流程图

串口通信中断处理程序如下：

```
.DEFINE  P_UART_Command1          0x7021
.DEFINE  P_UART_Command2          0x7022
.DEFINE  P_UART_Data              0x7023
.DEFINE  P_UART_BaudScalarLow     0x7024
.DEFINE  P_UART_BaudScalarHigh    0x7025

.DEFINE  P_IOB_Data               0x7005
.DEFINE  P_IOB_Buffer             0x7006
.DEFINE  P_IOB_Dir                0x7007

.DEFINE  P_IOB_Attrib             0x7008
.DEFINE  UART_Data_Size           0x1ff;
.RAM
UART_Data_BUF: .dw 0x100 dup (0);
.var  UART_Data_Head,UART_Data_Tail;

.CODE
//设置串口
UART_Setup:  .proc
        INT  OFF
                                  //设置 IOB10 为输出，IOB7 为输入
        R1=0x0480
        [P_IOB_Attrib]=R1
        R1=0x0400
        [P_IOB_Dir]=R1
                                  // 设置波特率为 2400bps
        R1=0x00
        [P_UART_BaudScalarLow]=R1
        R1=0x14
        [P_UART_BaudScalarHigh]=R1
                                  //允许 UART  IRQ 中断(由 RxRDY 信号触发)
        R1=0x84
        [P_UART_Command1]=R1
                                  //发送、接收管脚允通
        R1=0xC0
        [P_UART_Command2]=R1
        R1=0
        [UART_Data_Head]=R1
        [UART_Data_Tail]=R1
        INT  IRQ
        RETF
        .ENDP
```

```
//串口通信中断处理
.public _IRQ7
UART_RECV_IRQ: .PROC
_IRQ7:
        INT OFF
        PUSH R1,R4 to [sp]
        R1=[UART_Data_Tail]
        R2=R1+1
        R2=R2 & UART_Data_Size
        R4=[UART_Data_Head]
        CMP R2,R4
        JNE get_data
        R3=[P_UART_Data]                    // 丢弃数据
        JMP leave_ready
get_data:
        R3=6                                // 计数赋值
?not_ready:
        R3-=1;
        JZ leave_ready
        R4=0x0080                           // 判断数据是否已接收完毕
        TEST R4,[P_UART_Command2]
        JZ ?not_ready
        R3=[P_UART_Data]
        TEST R1,0x1
        JNZ ?low_byte
        R3=R3 LSL 4
        R3=R3 LSL 4
        R3=R3 & 0xff00
        R1=R1 ASR 1
        JMP save_byte
?low_byte:
        R3=R3 & 0xff
        R1=R1 ASR 1
        R4=[R1]
        R4=R4 &0xff00
        R3=R3 | R4
save_byte:
        R1=R1+UART_Data_BUF
        [R1]=R3
        [UART_Data_Tail]=R2
        JMP leave_irq7
leave_ready:
        R3=[P_UART_Data];                   //丢弃数据
leave_irq7:
```

```
        POP  R1,R4  from  [sp]
        INT  IRQ
        RETI
        .ENDP
.END
```

8.5 CRC 校验程序

8.5.1 CRC 简介

传统的差错检验法有：奇偶校验法，校验和法，行列冗余校验法等。这些方法都 是在数据后面加一定数量的冗余位同时发送出去，例如在单片机的通讯方式 2 和 3 中，TB8 就可以作为奇偶校验位同数据一起发送出去，在数据的接收端通过对数据信息进行比较、判别或简单的求和运算，然后将所得和接收到的冗余位进行比较，若相等就认为数据接收正确，否则就认为数据传送过程中出现错误。但是冗余位只能反映数据行或列的奇偶情况，所以这 类检验方法对数据行或列的偶数个错误不敏感，漏判的概率很高。因此，此种方法的可靠性 就差。 循环冗余码校验英文名称为 Cyclical Redundancy Check，简称 CRC。它是利用除法及余数 的原理来作错误侦测（Error Detecting）的。

8.5.2 CRC 原理

CRC 检验原理实际上就是在一个 p 位二进制数据序列之后附加一个 r 位二进制检验码(序列),从而构成一个总长为 $n=p+r$ 位的二进制序列，例如，p 位二进制数据序列 $D=[d_{p-1}d_{p-2}\ .....d_1d_0]$，$r$ 位二进制检验码 $R=[r_{r-1}\ r_{r-2}....r_1\ r_0]$，所得到的这个 n 位二进制序列就是 $M=[d_{p-1}d_{p-2}\ .....d_1d_0\ r_{r-1}\ r_{r-2}....r_1\ r_0]$； 附加在数据序列之后的这个检验码与数据序列的内容之间存在着某种特定的关系。如果因干扰等原因使数据序列中的某一位或某些位发生错误，这种特定关系就会被破坏,因此,通过检查这一关系, 就可以实现对数据正确性的检验。

校验码 R 是通过对数据序列 D 进行二进制除法取余式运算得到的，它被一个称为生成多项式的（$r+1$）位二进制序列 $G=[g_r\ g_{r-1}....\ g_1\ g_0]$来除，用多项式形式表示为

$$\frac{x^rD(x)}{G(x)}=Q(x)+\frac{R(x)}{G(x)} \qquad (1)$$

其中，$x^rD(x)$表示将数据序列 D 左移 r 位（即在 D 的末尾再增加 r 个 0 位），$Q(x)$代表这一除法所得的商，$R(x)$就是所需的余式。这一运算关系还可以用式（2）来表达

$$R(x) = \mathrm{Re}[\frac{x^r D(x)}{G(x)}] \qquad (2)$$

其中，Re[ ]表示对括号内的式子进行取余式运算。

检验码的编码计算如上所述，而检验过程则是对 M序列直接进行除法取余式运算，即

$$\frac{M(x)}{G(x)} = Q(x) + \frac{R(x)}{G(x)} \qquad (3)$$

或表示为

$$R(x) = \mathrm{Re}[\frac{M(x)}{G(x)}] \qquad (4)$$

所得到的余式 $R(x)$若为零则表示数据正确，否则认为发生错误。

CRC 校验程序

```
//=============================================================================
//函数: cal_crc()
//语法：unsigned int cal_crc(unsigned char *ptr, unsigned char len);
//描述：CRC 校验
//参数：1.校验数据的地址 2.校验数据的个数
//返回：校验码
//=============================================================================
.IRAM
.VAR C_Shift;
.VAR C_CRC;                 //存储 C_CRC 校验码
.CODE
.PUBLIC _F_CRC;
_F_CRC:
  PUSH BP TO [SP];
  PUSH R1,R2 TO [SP];       //压栈保护
  R1 = 0x0000;
  [C_CRC] = R1;             //变量清零
  POP R1,R2 FROM [SP]
  BP = SP + 1
  R1 = [BP + 3];            //校验数据的地址
  R2 = [BP + 4];            //校验数据的个数
  BP = R1;
L_CRCOutLoop:
  BP = [R1++]               //校验数据的地址加一
  R3 = 0x0080;
  [C_Shift] = R3;           //为移位服务
L_CRCInLoop:
  R3 = [C_CRC];
```

```
  R3 &= 0x8000;
  JZ L_CRC1;
  R4 = [C_CRC];
  R4 = R4 LSL 1;          //左移
  R3 = 0x1021;
  R4 ^=  R3;
  [C_CRC] = R4;
  GOTO L_CRC2;
L_CRC1:
  R4 = [C_CRC];
  R4 = R4 LSL 1;          //左移
  [C_CRC] = R4            //保存 C_CRC 检验码
L_CRC2:
  R4 = [C_Shift];
  R4 &= BP;
  JZ L_CRC3;
  R3 = [C_CRC];
  R4 = 0x1021;
  R3 ^= R4 ;              //异或 0x1021
  [C_CRC] = R3;           //保存 C_CRC 检验码
L_CRC3:
  R4 = [C_Shift];
  R4 = R4 lsr 1;          //右移
  JZ L_CRC4;
  [C_Shift] = r4;
  GOTO L_CRCInLoop;
L_CRC4:
   R2 -= 1;
   JZ L_Exit;
   GOTO L_CRCOutLoop;
L_Exit:
  R1 = [C_CRC]
  POP BP,BP FROM [SP]
  RETF;
```