第7章 凌阳音频压缩算法

7.1 背景介绍

7.1.1 音频的概述（特点、分类）

我们所说的音频是指频率在 20 Hz～20 kHz 的声音信号，分为：波形声音、语音和音乐三种，其中波形声音就是自然界中所有的声音，是声音数字化的基础。语音也可以表示为波形声音，但波形声音表示不出语言、语音学的内涵。语音是对讲话声音的一次抽象。是语言的载体，是人类社会特有的一种信息系统，是社会交际工具的符号。音乐与语音相比更规范一些，是符号化了的声音。但音乐不能对所有的声音进行符号化。乐谱是符号化声音的符号组，表示比单个符号更复杂的声音信息内容。

7.1.2 数字音频的采样和量化

将模拟的（连续的）声音波形数字元化（离散化），以便利数字计算机进行处理的过程，主要包括采样和量化两个方面。

数字音频的质量取决于：采样频率和量化位数这两个重要参数。此外，声道的数目、相应的音频设备也是影响音频质量的原因。

7.1.3 音频格式的介绍

音频文件通常分为两类：声音文件和 MIDI 文件

（1）声音文件：指的是通过声音录入设备录制的原始声音，直接记录了真实声音的二进制采样数据，通常文件较大；

（2）MIDI 文件：它是一种音乐演奏指令序列，相当于乐谱，可以利用声音输出设备或与计算机相连的电子乐器进行演奏，由于不包含声音数据，其文件尺寸较小。

1）声音文件的格式

WAVE 文件——*.WAV

WAVE 文件使用三个参数来表示声音 ，它们是：采样位数、采样频率和声道数。在计算机中采样位数一般有 8 位和 16 位两种，而采样频率一般有 11025Hz（11KHz），22050Hz（22KHz）、44100Hz（44KHz）三种。我们以单声道为例，则一般 WAVE 文件的比特率可达到 88K~704Kbps。具体介绍如下：

（1） WAVE 格式是 Microsoft 公司开发的一种声音文件格式，它符合 RIFF（Resource Interchange File Format）文件规范；

（2） 用于保存 Windows 平台的音频信息资源，被 Windows 平台及其应用程序所广泛支持。
（3） WAVE 格式支持 MSADPCM、CCITT A　Law、CCITT μ Law 和其它压缩算法，支持多种音频位数、采样频率和声道，是 PC 机上最为流行的声音文件格式。
（4） 但其文件尺寸较大，多用于存储简短的声音片段。

AIFF 文件——AIF/AIFF
（1） AIFF 是音频交换文件格式（Audio Interchange File Format）的英文缩写，是苹果计算机公司开发的一种声音文件格式；
（2） 被 Macintosh 平台及其应用程序所支持，Netscape Navigator 浏览器中的 LiveAudio 也支持 AIFF 格式，SGI 及其它专业音频软件包同样支持这种格式。
（3） AIFF 支持 ACE2、ACE8、MAC3 和 MAC6 压缩，支持 16 位 44.1Kz 立体声。

Audio 文件——*.Audio
（1） Audio 文件是 Sun Microsystems 公司推出的一种经过压缩的数字声音格式，是 Internet 中常用的声音文件格式；
（2） Netscape Navigator 浏览器中的 LiveAudio 也支持 Audio 格式的声音文件。

MPEG 文件——*.MP1/*.MP2/*.MP3
（1） MPEG 是运动图像专家组（Moving Picture Experts Group）的英文缩写，代表 MPEG 标准中的音频部分，即 MPEG 音频层（MPEG Audio Layer）；
（2） MPEG 音频文件的压缩是一种有损压缩，根据压缩质量和编码复杂程度的不同可分为三层（MPEG Audio Layer1/2/3）,分别对应 MP1、MP2 和 MP3 这三种声音文件；
（3） MPEG 音频编码具有很高的压缩率，MP1 和 MP2 的压缩率分别为 4：1 和 6：1~8：1，而 MP3 的压缩率则高达 10：1~12：1，也就是说一分钟 CD 音质的音乐，未经压缩需要 10MB 存储空间，而经过 MP3 压缩编码后只有 1MB 左右，同时其音质基本保持不失真，因此，目前使用最多的是 MP3 文件格式。

RealAudio 文件——*.RA/*.RM/*.RAM
（1） RealAudio 文件是 RealNerworks 公司开发的一种新型流式音频（Streaming Audio）文件格式；
（2） 它包含在 RealMedia 中，主要用于在低速的广域网上实时传输音频信息；
（3） 网络连接速率不同，客户端所获得的声音质量也不尽相同：对于 28.8Kbps 的连接，可以达到广播级的声音质量；如果拥有 ISDN 或更快的线路连接，则可获得 CD 音质的声音。

2）MIDI 文件——*.MID/*.RMI
（1） MIDI 是乐器数字接口（Musical Instrument Digital Interface）的英文缩写，是数字音乐/电子合成乐器的统一国际标准；
（2） 它定义了计算机音乐程序、合成器及其它电子设备交换音乐信号的方式，还规

定了不同厂家的电子乐器与计算机连接的电缆和硬件及设备间数据传输的协议，可用于为不同乐器创建数字声音，可以模拟大提琴、小提琴、钢琴等常见乐器；

（3） 在 MIDI 文件中，只包含产生某种声音的指令，这些指令包括使用什么 MIDI 设备的音色、声音的强弱、声音持续多长时间等，计算机将这些指令发送给声卡，声卡按照指令将声音合成出来，MIDI 在重放时可以有不同的效果，这取决于音乐合成器的质量；

（4） 相对于保存真实采样资料的声音文件，MIDI 文件显得更加紧凑，其文件尺寸通常比声音文件小得多。

7.1.4 语音压缩编码基础

语音压缩编码中的数据量是指：数据量=(采样频率×量化位数)/8(字节数) ×声道数目。

压缩编码的目的：通过对资料的压缩，达到高效率存储和转换资料的结果，即在保证一定声音质量的条件下，以最小的资料率来表达和传送声音信息。

压缩编码的必要性：实际应用中，未经压缩编码的音频资料量很大，进行传输或存储是不现实的。 所以要通过对信号趋势的预测和冗余信息处理，进行资料的压缩，这样就可以使我们用较少的资源建立更多的信息。

举个例子，没有压缩过的 CD 品质的资料，一分钟的内容需要 11MB 的内存容量来存储。如果将原始资料进行压缩处理，在确保声音品质不失真的前提下，将数据压缩一半，5.5MB 就可以完全还原效果。而在实际操作中，可以依需要来选择合适的算法。

常见的几种音频压缩编码：

1) 波形编码：将时间域信号直接变换为数字代码，力图使重建语音波形保持原语音信号的波形形状。波形编码的基本原理是在时间轴上对模拟语音按一定的速率抽样，然后将幅度样本分层量化，并用代码表示。译码是其反过程，将收到的数字序列经过译码和滤波恢复成模拟信号。

如：脉冲编码调制(Pulse Code Modulation，PCM)、差分脉冲编码调制（DPCM）、增量调制(DM)以及它们的各种改进型，如自适应差分脉冲编码调制（ADPCM）、自适应增量调制（ADM）、自适应传输编码（Adaptive Transfer Coding，ATC）和子带编码（SBC）等都属于波形编码技术。

波形编码特点：高话音质量、高码率，适于高保真音乐及语音。

2) 参数编码：参数编码又称为声源编码，是将信源信号在频率域或其它正交变换域提取特征参数，并将其变换成数字代码进行传输。译码为其反过程，将收到的数字序列经变换恢复特征参量，再根据特征参量重建语音信号。具体说，参数编码是通过对语音信号特征参数的提取和编码，力图使重建语音信号具有尽可能高的准确性，但重建信号的波形同原语音信号的波形可能会有相当大的差别。

如：线性预测编码（LPC）及其它各种改进型都属于参数编码。该编码比特率可压缩到 2Kbit/s-4.8Kbit/s，甚至更低，但语音质量只能达到中等，特别是自然度较低。

参数编码特点：压缩比大，计算量大，音质不高，廉价！

3) 混合编码：混合编码使用参数编码技术和波形编码技术，计算机的发展为语音编码技术的研究提供了强有力的工具，大规模、超大规模集成电路的出现，则为语音编码的实现提供了基础。80 年代以来，语音编码技术有了实质性的进展，产生了新一代的编码算法，这就是混合编码。它将波形编码和参数编码组合起来，克服了原有波形编码和参数编码的弱点，结合各自的长处，力图保持波形编码的高质量和参数编码的低速率。

如：多脉冲激励线性预测编码（MPLPC），规划脉冲激励线性预测编码（KPELPC），码本激励线性预测编码（CELP）等都是属于混合编码技术。其数据率和音质介于参数和波形编码之间。

总之，音频压缩技术之趋势有两个：

1)降低资料率，提高压缩比，用于廉价、低保真场合（如：电话）。

2)追求高保真度，复杂的压缩技术（如：CD）。**语音合成、辨识技术的介绍：**

按照实现的功能来分，语音合成可分两个档次：

(1) 有限词汇的计算机语音输出

(2) 基于语音合成技术的文字语音转换（TTS：Text-to-Speech）

按照人类语言功能的不同层次，语音合成可分为三个层次：

(1) 从文字到语音的合成（Text-to-Speech）

(2) 从概念到语音的合成（Concept-to-Speech）

(3) 从意向到语音的合成（Intention-to-Speech）

图 7.1 是文本到语音的转换过程:

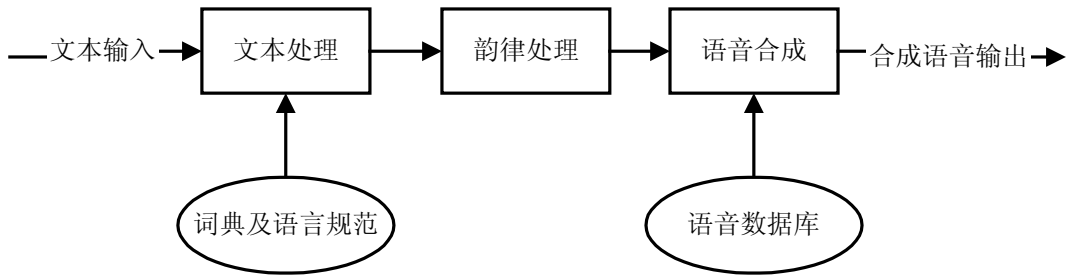

图7.1 从文本到语音转换过程示意

语音辨识：

语音辨识技术有三大研究范围：口音独立、连续语音及可辨认字词数量。

口音独立：

1）早期只能辨认特定的使用者即特定语者(Speaker Dependent，SD)模式，使用者可针对特定语者辨认词汇(可由使用者自行定义，如人名声控拨号)，作简单快速的训

练纪录使用者的声音特性来加以辨认。随着技术的成熟，进入语音适应阶段 SA(speaker adaptation),使用者只要对于语音辨识核心，经过一段时间的口音训练后，即可拥有不错的辨识率。

2）非特定语者模式(Speaker Independent，SI)，使用者无需训练即可使用，并进行辨认。任何人皆可随时使用此技术，不限定语者即男性、女性、小孩、老人皆可。

连续语音：

1）单字音辨认：为了确保每个字音可以正确地切割出来，必须一个字一个字分开来念，非常不自然，与我们平常说话的连续方式，还是有点不同。

2）整个句子辨识：只要按照你正常说话的速度，直接将要表达的说出来，中间并不需要停顿，这种方式是最直接最自然的，难度也最高，现阶段连续语音的辨识率及正确率，虽然效果还不错但仍需再提高。然而，中文字有太多的同音字，因此目前所有的中文语音辨识系统，几乎都是以词为依据，来判断正确的同音字。

可辨认词汇数量：

内建的词汇数据库的多寡，也直接影响其辨识能力。因此就语音辨识 的词汇数量来说亦可分为三种：

1）小词汇量(10-100)

2）中词汇量(100-1000)

3）无限词汇量（即听写机）

图 7.2 是简化的语音识别原理图，其中实线部分成为训练模块，虚线部分为识别模块。

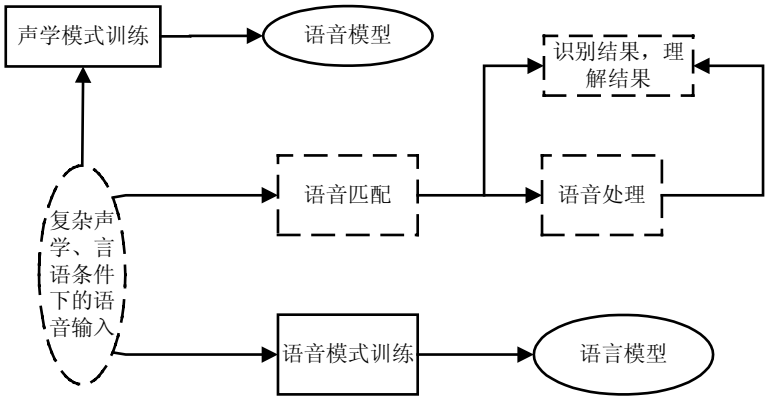

图7.2 语音识别原理简图

7.2 凌阳音频简介

7.2.1 凌阳音频压缩算法的编码标准

表 7.1 是不同音频质量等级的编码技术标准 （频响）：

表7.1

信号类型	频率范围（Hz）	采样率（kHz）	量化精度（位）
电话话音	200~3400	8	8
宽带音频（AM 质量）	50~7000	16	16
调频广播（FM 质量）	20~15k	37.8	16
高质量音频（CD 质量）	20~20k	44.1	16

凌阳音频压缩算法处理的语音信号的范围是 200Hz－3.4KHz 的电话话音。

7.2.2 压缩分类

压缩分无损压缩和有损压缩。

无损压缩一般指：磁盘文件，压缩比低：2:1～4:1。

而有损压缩则是指：音 / 视频文件，压缩比可高达 100:1。

凌阳音频压缩算法根据不同的压缩比分为以下几种 (具体可参见语音压缩工具一节内容)：

SACM-A2000：压缩比为 8: 1 ，8:1.25，8:1.5

SACM-S480： 压缩比为 80:3，80:4.5

SACM-S240： 压缩比为 80:1.5

按音质排序：A2000>S480>S240

7.2.3 凌阳常用的音频形式和压缩算法

1) 波形编码：**sub-band** 即 SACM-A2000

特点：高质量、高码率，适于高保真语音 / 音乐。

2) 参数编码：声码器（vocoder）模型表达，抽取参数与激励信号进行编码。如：SACM-S240。

特点：压缩比大，计算量大，音质不高，廉价！

3) 混合编码：**CELP** 即 SACM-S480

特点：综合参数和波形编码之优点。

除此之外,还具有 FM 音乐合成方式即 SACM-MS01。

7.2.4 分别介绍凌阳语音的播放、录制、合成和辨识

凌阳的 SPCE061A 是 16 位单片机，具有 DSP 功能，有很强的信息处理能力，最高时钟频率可达到 49MHz，具备运算速度高的优势等等，这些都无疑为语音的播放、录放、合成及辨识提供了条件。

凌阳压缩算法中 SACM_A2000、SACM_S480、SACM_S240 主要是用来放音，可用于语音提示，而 DVR 则用来录放音。对于音乐合成 MS01，该算法较繁琐，而且需要具备音乐理论、配器法及和声学知识，所以对于特别爱好者可以到我们的网站去了解相关内容，这里只给出它的 API 函数介绍及程序代码的范例，仅供参考。

对于语音辨识主要有以下两种：

1）特定发音人识别 SD（Speaker　Dependent）：是指语音样板由单个人训练，也只能识别训练人的语音命令，而他人的命令识别率较低或几乎不能识别。

2）非特定发音人识别 SI（Speaker　Independent）：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。

语音识别电路基本结构如图 7.3 所示:

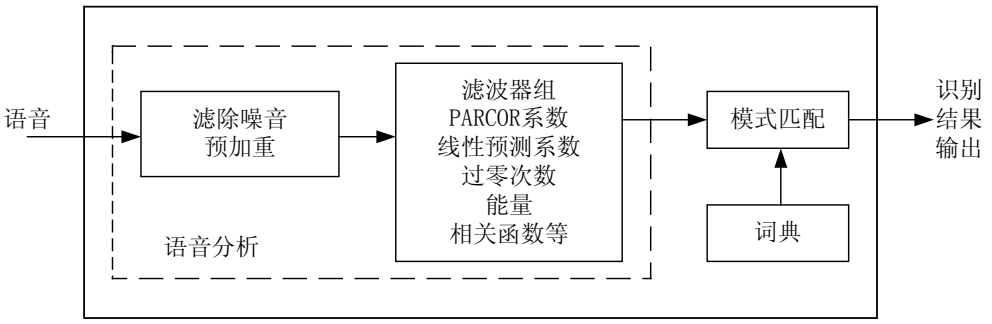

图7.3　语音识别电路结构

具体应用及程序代码可参考 7.3.4

7.3 常用的应用程序接口 API 的功能介绍及应用

7.3.1 概述

表 7.2 所列出的是凌阳音频的几种算法：

表7.2 SACM-lib 库中模块及其算法类型

模块名称（Model-Index）	语音压缩编码率类型	资料采样率
SACM_A2000	16Kbit/s，20 Kbit/s，24 Kbit/s	16KHz
SACM_S480/S720	4.8 Kbit/s，7.2 Kbit/s	16KHz

SACM_S240	2.4 Kbit/s	24KHz
SACM_MS01	音乐合成（16Kbits/s，20 Kbits/s，24 Kbits/s）	16KHz
SACM_DVR（A2000）	16 Kbit/s 的资料率，8 K 的采样率，用于 ADC 通道录音功能	16KHz

语音和音乐与我们的生活有着非常密切的关系，而单片机对语音的控制如录放音、合成及辨识也广泛应用在现实生活中。我们知道对于语音处理大致可以分为 A/D、编码处理、存储、解码处理以及 D/A 等见图 7.4 所示。然而，通过前面介绍我们知道麦克风输入所生成的 WAVE 文件，其占用的存储空间很大，对于单片机来说想要存储大量的信息显然是不可能的，而凌阳的 SPCE061A 提出了解决的方法，即 SACM-LIB，该库将 A/D、编码、解码、存储及 D/A 作成相应的模块，对于每个模块都有其应用程序接口 API，所以您只需了解每个模块所要实现的功能及其参数的内容，然后调用该 API 函数即可实现该功能，例如在程序中插入语音提示，或连续播放一段语音或音乐，也可以根据自己需要的空间或使用范围选择适合自己的算法如表 7.2 所示。

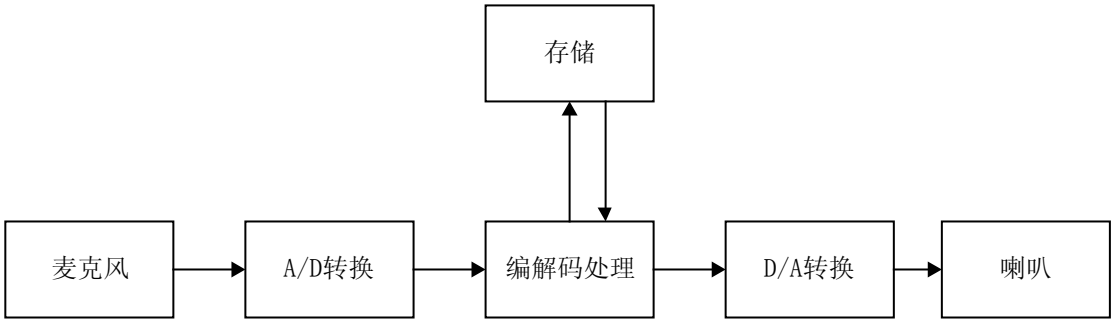

图7.4 单片机对语音处理过程

以下就不同的算法具体介绍各自的 API 函数的格式、功能、参数、返回值、备注及应用范例。

7.3.2 SACM_A2000

该压缩算法压缩比较小(8:1)所以具有高质量、高码率的特点适用于高保真音乐和语音。

其相关 API 函数如下所示：

```
void SACM_A2000_Initial(int Init_Index)                          //初始化
void SACM_A2000_ServiceLoop(void)                                //获取语音资料，填入译码队列
void SACM_A2000_Play(int Speech_Index, int Channel, int Ramp_Set)    //播放
void SACM_A2000_Stop(void)                                       //停止播放
void SACM_A2000_Pause (void)                                     //暂停播放
void SACM_A2000_Resume(void)                                     //暂停后恢复
void SACM_A2000_Volume(Volume_Index)                             //音量控制
unsigned int SACM_A2000_Status(void)                             //获取模块状态
```

```
void SACM_A2000_InitDecode(int Channel)                    //译码初始化
void SACM_A2000_Decode(void)                               //译码
void SACM_A2000_FillQueue(unsigned int encoded-data)//填充队列
unsigned int SACM_A2000_TestQueue(void)                    //测试队列
Call F_FIQ_Service_ SACM_A2000                             //中断服务函数
```

下面对各个函数进行具体介绍：

1)【API 格式】C：void SACM_A2000_Initial(int Init_Index)

ASM：R1=[Init_Index]

Call F_ SACM_A2000_Initial

【功能说明】SACM_A2000 语音播放之前的初始化。

【参　　数】Init_Index=0 表示手动方式；Init_Index=1 则表示自动方式。

【返 回 值】无

【备　　注】该函数用于对定时器、中断和 DAC 等的初始化。

2)【API 格式】C：void SACM_A2000_ServiceLoop(void)

ASM：Call F_ SACM_A2000 _ServiceLoop

【功能说明】从资源中获取 SACM_A2000 语音资料，并将其填入译码队列中。

【参　　数】无。

【返 回 值】无。

3)【API 格式】

C ： void SACM_A2000_Play(int Speech_Index, int Channel, int Ramp_Set)；

ASM：R1=[Speech _Index]

R2=[Channel]

R3=[Ramp_Set]

Call SACM_A2000_Play

【功能说明】播放资源中 SACM_A2000 语音或乐曲。

【参　　数】Speech _Index:表示语音索引号。

Channel： 1.通过 DAC1 通道播放；

2.通过 DAC2 通道播放；

3.通过 DAC1 和 DAC2 双通道播放。

Ramp_Set：0.禁止音量增/减调节；

1.仅允许音量增调节；

2.仅允许音量减调节；

3.允许音量增/减调节。

【返 回 值】无。

【备　　注】

①SACM_A2000 的数据率有 16Kbps\20Kbps\24Kbps 三种，可在同一模块的几种算法中自动选择一种。

② Speech_Index 是定义在 resource.inc 文件中资源表（T_SACM_A2000_SpeechTable）的偏移地址。

③中断服务子程序 F_FIQ_Service_ SACM_A2000 必须安置在 TMA_FIQ 中断向量上（参见第五章**中断系统**内容）。

函数允许 TimerA 以所选的的数据采样率（计数溢出）中断。

程序7-1 以自动方式播放一段 SACM_A2000 语音，并自动结束。(见光盘)

SACM_A2000 自动方式主程序流程图：

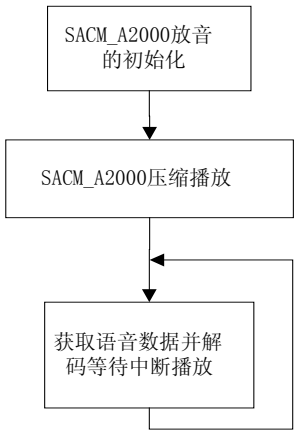

图7.5 A2000 自动方式主程序流程

前台程序：

```
#define     Speech_1        0
#define     DAC1            1
#define     DAC2            2
#define     Ramp_UpDn_Off   0
#define     Ramp_Up_On      1
#define     Ramp_Dn_On      2
#define     Ramp_UpDn_On    3
Main ()
{
   SACM_A2000_Initial(1);
   SACM_A2000_Play(Speech_1, DAC1+DAC2, Ramp_UpDn_On);    //放音
   while(SACM_A2000_Status()&0x01)
   {
      SACM_A2000_ServiceLoop();
   }
}
```

背景程序:

```
.TEXT
```

```
.INCLUDE      hardware.inc
.INCLUDE      A2000.inc
.INCLUDE      Resource.inc
//=========================================================
//函数: FIQ()
//语法：void FIQ(void)
//描述：FIQ 中服务断函数
//参数：无
//返回：无
//=========================================================
.PUBLIC  _FIQ;
_FIQ:
 PUSH     R1,R4 TO [sp];              //入栈保护
 R1=0x2000;
 TEST R1,[P_INT_Ctrl];               //是否为定时器 A 中断
 JNZ L_FIQ_TimerA;
 R1=0x0800;
 TEST R1,[P_INT_Ctrl];               //是否为定时器 B 中断
 JNZ L_FIQ_TimerB;
L_FIQ_PWM:
 R1=C_FIQ_PWM;
 [P_INT_Clear]=R1;                   //清中断
 POP R1,R4 from[sp];                 //恢复现场
 RETI;
L_FIQ_TimerA:                        //定时器 A 中断处理
 [P_INT_Clear]=R1;                   //清中断
 CALL F_FIQ_Service_SACM_A2000;      //调用 A2000 中断服务函数
 POP R1,R4 FROM [sp];                //恢复现场
 RETI;
L_FIQ_TimerB:                        //定时器 B 中断处理
 [P_INT_Clear]=R1;                   //清中断
 POP R1,R4 FROM [sp];                //恢复现场
 RETI;                               //中断返回
 /*************************************************************************/
```

注：播放语音文件中数据，当出现 FF FF FFH 数据时便停止播放。

4）【API 格式】C：void SACM_A2000_Stop(void)；

ASM：Call F_ SACM_A2000_Stop

【功能说明】停止播放 SACM_A2000 语音或乐曲。

【参　　数】无。

【返 回 值】无。

5) 【API 格式】C：void SACM_A2000_Pause (void)；

ASM：Call F_ SACM_A2000_Pause

【功能说明】暂停播放 SACM_A2000 语音或乐曲。

【参　　数】无。
【返 回 值】无。

6) 【API 格式】C：void SACM_A2000_Resume(void)；
ASM：Call F_ SACM_A2000_Resume
【功能说明】恢复暂停播放的 SACM_A2000 语音或乐曲。
【参　　数】无。
【返 回 值】无。

7) 【 API 格式】C：void SACM_A2000_Volume(Volume_Index)；
ASM：R1=[Volume_Index]
Call F_ SACM_A2000_Volume
【功能说明】在播放 SACM_A2000 语音或乐曲时改变主音量。
【参　　数】Volume_Index 为音量数，音量从最小到最大可在 0~15 之间选择。
【返 回 值】无。

8) 【API 格式】C：unsigned int SACM_A2000_Status(void)；
ASM：Call F_ SACM_A2000_ Status
[返回值]=R1
【功能说明】获取 SACM_A2000 语音播放的状态。
【参　　数】无。
【返 回 值】当 R1 的 bit0=0，表示语音播放结束；bit0=1，表示语音在播放中。

9) 【API 格式】ASM：Call F_FIQ_Service_ SACM_A2000
【功能说明】用作 SACM_A2000 语音背景程序的中断服务子程序。通过前台子程序（自动方式的 SACM_A2000_ServiceLoop 及手动方式的 SACM_A2000_Decode）对语音资料进行解码，然后将其送入 DAC 通道播放。
【参　　数】无。
【返 回 值】无。
【备　　注】SACM_A2000 语音背景子程序只有汇编指令形式，且应将此子程序安置在 TMA_FIQ 中断源上。

10) 【API 格式】C：void SACM_A2000_InitDecode(int Channel)；
ASM：Call F_ SACM_A2000_Decode
【功能说明】开始对 SACM_A2000 语音资料以非自动方式（编程控制）进行译码。
【参　　数】Channel=1，2，3；分别表示使用 DAC1、DAC2 通道以及 DAC1 和 DAC2 双通道。
【返 回 值】无。
【备　　注】用户只能通过非自动方式对语音资料解压缩。

11) 【API 格式】C：void SACM_A2000_Decode(void)；
ASM：Call F_ SACM_A2000_Decode

【功能说明】从语音队列里获取的 SACM_A2000 语音资料，并进行译码，然后通过中断服务子程序将其送入 DAC 通道播放。

【参　　数】无。

【返 回 值】无。

【备　　注】用户仅能通过非自动方式对语音资料进行译码。

12) 【 API 格式】C：void SACM_A2000_FillQueue(unsigned int encoded-data)；

ASM：R1=[语音编码资料]

Call F_ SACM_A2000_FillQueue

【功能说明】将从用户存储区里获取 SACM_ A2000 语音编码资料，然后将其填入语音队列中等候译码处理。

【参　　数】encoded-data 为语音编码资料。

【返 回 值】无。

【备　　注】用户仅能通过非自动方式对语音资料进行译码。

13) 【 API 格式】C：unsigned int SACM_A2000_TestQueue(void)；

ASM：Call F_ SACM_A2000_TestQueue

[返回值]=R1

【功能说明】获取语音队列的状态。

【参　　数】无。

【返 回 值】R1=0，1，2；分别表示语音队列不空不满，语音队列满及语音队列空。

【备　　注】用户仅能通过非自动方式测试语音队列状态。

程序7-2 SACM_A2000 非自动方式（编程控制）播放语音。(见光盘)

SACM_A2000 非自动方式主程序流程见图 7.6：

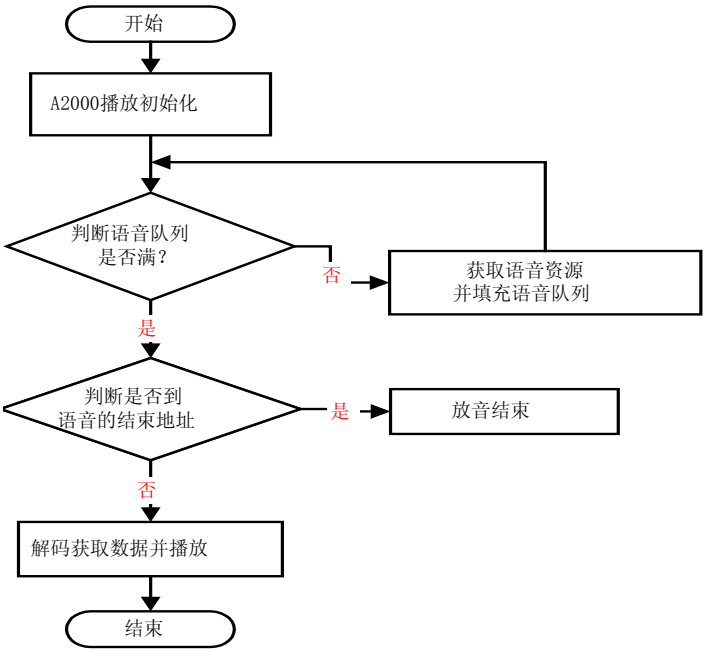

图7.6 SACM_A2000 非自动方式主程序流程

中断服务子程序流程见图 7.7：

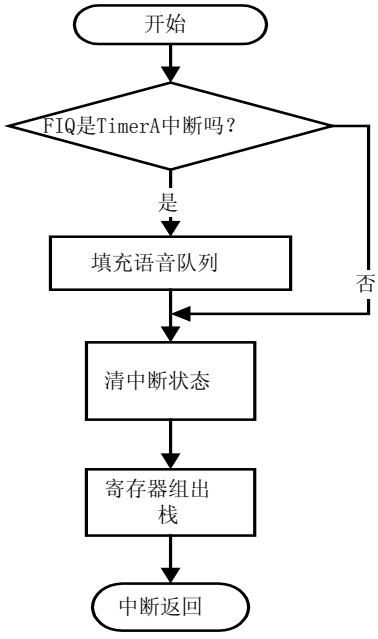

图7.7 SACM_A2000 中断服务子程序流程

前台程序：

```
    #define     Manual    0
    #define     Auto      1
    #define     Full      1
    #define     Empty     2
    #define     DAC1      1
    #define     DAC2      2
    Main()
{
      Addr=RES_A32_SA;                          //长整型资源地址
      SACM_A2000_Initial(Manual);               //选择非自动方式
      SACM_A2000_InitDecode(DAC1+DAC2);         //使用双通道
      While(SACM_A2000_TestQueue()!=Full)       //若队列不满，填入资料
      {
        Ret=GetResource(Addr);                  //从 ROM 中取语音资料
        SACM_A2000_FillQueue(Ret);              //将语音资料填入队列
        Addr++;                                 //指向下一个资料地址
      }
      While(1)
      {
        If(SACM_A2000_TestQueue()!=Full)        //继续填资料到队列中
          {
          Ret=GetResource(Addr);
          SACM_A2000_FillQueue(Ret);
          Addr++;
          }
        if(Addr< RES_A32_EA)
          SACM_A2000_Decode();                  //对语音资料进行译码
        else SACM_A2000_Stop();                 //地址结束，停止播放
      }
}
```

注：

1）文件的结束是由用户位址变量控制的。

2）在非自动方式播放语音，其音量的增/减是通过外部子程序（SP_Ramp_Up,SP_Ramp_Dn）控制的。

7.3.3 SACM_S480

该压缩算法压缩比较大 80:3, 存储容量大,音质介于 A2000 和 S240 之间,适用于语音播放, 如“文曲星”词库。

其相关 API 函数如下所示：

```
int SACM_S480_Initial(int Init_Index)                          //初始化
void SACM_ S480_ServiceLoop(void)                              //获取语音资料，填入译码队列
void SACM_ S480_Play(int Speech_Index, int Channel, int Ramp_Set)
```

```
                                                    //播放
void SACM_ S480_Stop(void)                          //停止播放
void SACM_S480_Pause (void)                         //暂停播放
void SACM_S480_Resume(void)                         //暂停后恢复
void SACM_S480_Volume(Volume_Index)                 //音量的控制
unsigned int SACM_S480_Status(void)                 //获取模块的状态
Call F_FIQ_Service_ SACM_S480                       //中断服务函数
```

各函数具体内容如下：

1）【API 格式】C：int　SACM_S480_Initial(int Init_Index)

ASM：R1=[Init_Index]

Call F_ SACM_ S480_Initial

【功能说明】SACM_S480 语音播放之前的初始化。

【参　　数】Init_Index=0 表示手动方式；Init_Index=1 则表示自动方式。

【返 回 值】0：代表语音模块初始化失败

1：代表初始化成功。

【备　　注】该函数用于对定时器、中断和 DAC 等的初始化。

2) 【API 格式】C：void SACM_S480_ServiceLoop(void)

ASM：Call F_ SACM_S480_ServiceLoop

【功能说明】从资源中获取 SACM_S480 语音资料，并将其填入解码队列中。

【参　　数】无。

【返 回 值】无。

【备　　注】播放语音文件中数据，当出现 FF FF FFH 数据时便停止播放。

3) 【API 格式】

C：int SACM_S480_Play(int Speech_Index, int Channel, int Ramp_Set)；

ASM：R1=[Speech _Index]

R2=[Channel]

R3=[Ramp_Set]

Call SACM_S480_Play

【功能说明】播放资源中 SACM_S480 语音。

【参　　数】Speech _Index 表示语音索引号。

Channel：　1.通过 DAC1 通道播放；

2.通过 DAC2 通道播放；

3.通过 DAC1 和 DAC2 双通道播放。

Ramp_Set：0.禁止音量增/减调节；

1.仅允许音量增调节；

2.仅允许音量减调节；

3.允许音量增/减调节。

【返 回 值】无。

【备　　注】

① SACM_S480 的数据率有 4.8Kbps\7.2Kbps 三种，可在同一模块的几种算法中自动选择一种。

②Speech_Index 是定义在 resource.inc 文件中资源表（T_SACM_S480_SpeechTable）的偏移地址。

③ 中断服务子程序中 F_FIQ_Service_ SACM_S480 必须放在 TMA_FIQ 中断向量上（参见 SPCE 的中断系统）。

④函数允许 TimerA 以所选的的数据采样率（计数溢出）中断。

程序7-3 以自动方式播放一段 SACM_S480 语音，并自动结束。(见光盘)

SACM_S480 自动方式主程序流程见图 7.8：

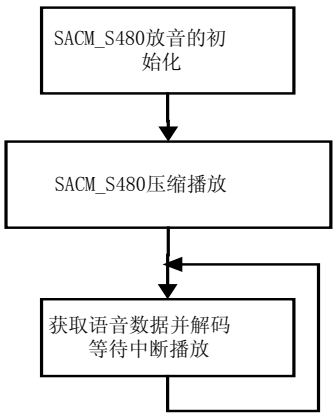

图7.8 SACM_S480 自动方式主程序流程

中断流程同上。

前台程序：

```
//*****************************************************************************************/
// 描述：  s480 只有自动播放方式,在中断 FIQ 的 FIQ_TMA 中断源中通过
//         主程序的 SACM_S480_ServiceLoop()对语音数据进行解码，然后将其
//         送入 DAC 通道播放
//*****************************************************************************************/
#include  "s480.h"
#define       Speech_1          0
#define       DAC1              1
#define       DAC2              2
#define       Ramp_UpDn_Off     0
#define       Ramp_UpDn_On      3
#define       Auto              1
//=========================================================================
// 函数：  main()
```

```
//=====================================================================
main ()
{
SACM_S480_Initial(1);              //自动方式播放初始化
SACM_S480_Play(Speech_1, DAC1+DAC2, Ramp_UpDn_On);
                                   //定义语音索引号、播放通道、允许音量增/减调节
while(SACM_S480_Status()&0x01)//是否播放结束
        SACM_S480_ServiceLoop(); //获取语音数据并将其填入解码队列
    }
```

背景程序:

```
    .TEXT
    .INCLUDE hardware.inc
    .INCLUDE S480.inc
    .PUBLIC   _FIQ;
    _FIQ:
     PUSH     R1,R4 TO [sp];              //入栈保护
     R1=0x2000;
     TEST R1,[P_INT_Ctrl];               //是否为定时器 A 中断
     JNZ L_FIQ_TimerA;
     R1=0x0800;
     TEST R1,[P_INT_Ctrl];               //是否为定时器 B 中断
     JNZ L_FIQ_TimerB;
    L_FIQ_PWM:
     R1=C_FIQ_PWM;
     [P_INT_Clear]=R1;                   //清中断
     POP R1,R4 from[SP];                 //恢复现场
     RETI;
    L_FIQ_TimerA:                        //定时器 A 中断处理
     [P_INT_Clear]=R1;                   //清中断
     CALL F_FIQ_Service_SACM_S480;       //调用 S480 中断服务函数
     POP R1,R4 FROM [SP];                //恢复现场
     RETI;
    L_FIQ_TimerB:                        //定时器 B 中断处理
     [P_INT_Clear]=R1;                   //清中断
     POP R1,R4 FROM [SP];                //恢复现场
     RETI;                               //中断返回
```

注：自动放音时，当语音资源文件中的资料为 FF FF FFH 时便停止播放。

4）【API 格式】C：void SACM_S480_Stop(void)；

ASM：Call F_ SACM_S480_Stop

【功能说明】停止播放 SACM_S480 语音。

【参　　数】无。

【返 回 值】无。

5) 【API 格式】C：void SACM_S480_Pause (void)；
ASM：Call F_ SACM_S480_Pause
【功能说明】暂停播放 SACM_S480 语音。
【参　　数】无。
【返 回 值】无。

6) 【 API 格式】C：void SACM_S480_Resume(void)；
ASM：Call F_ SACM_S480_Resume
【功能说明】恢复暂停播放的 SACM_S480 语音。
【参　　数】无。
【返 回 值】无。

7) 【API 格式】C：void SACM_S480_Volume(Volume_Index)；
ASM：R1=[Volume_Index]
Call F_Model-Index_Volume
【功能说明】在播放 SACM_S480 语音时改变主音量。
【参　　数】Volume_Index 为音量数，音量从最小到最大可在 0~15 之间选择。
【返 回 值】无。

8) 【 API 格式】C：unsigned int SACM_S480_Status(void)；
ASM：Call F_ SACM_S480_ Status
[返回值]=R1
【功能说明】获取 SACM_S480 语音播放的状态。
【参　　数】无。
【返 回 值】当 R1 的值 bit0=0，表示语音播放结束；bit0=1，表示语音在播放中。

9) 【 API 格式】ASM：Call F_FIQ_Service_ SACM_S480
【功能说明】用作 SACM_S480 语音背景程序的中断服务子程序。通过前台子程序（自动方式的 SACM_S480_ServiceLoop 及手动方式的 SACM_S480_Decode）对语音资料进行解码，然后将其送入 DAC 通道播放。
【参　　数】无。
【返 回 值】无。
【备　　注】SACM_S480 语音背景子程序只有汇编指令形式，且应将此子程序安置在 TMA_FIQ 中断源上。

7.3.4 SACM_S240

该压缩算法的压缩比较大 80:1.5,价格低,适用于对保真度要求不高的场合, 如玩具类产品的批量生产，编码率仅为 2.4 Kbps。

其相关 API 函数如下所示：

```
int SACM_S240_Initial(int Init_Index)                    //初始化
void SACM_ S240_ServiceLoop(void)                        //获取语音资料，填入译码队列
void SACM_ S240_Play(int Speech_Index, int Channel, int Ramp_Set)
```

```
                                                    //播放
void SACM_ S240_Stop(void)                          //停止播放
void SACM_S240_Pause (void)                         //暂停播放
void SACM_S240_Resume(void)                         //暂停后恢复
void SACM_S240_Volume(Volume_Index)                 //音量控制
unsigned int SACM_S240_Status(void)                 //获取模块状态
Call F_FIQ_Service_ SACM_S240                       //中断服务函数
```

下面具体介绍一下各个函数：

1) 【API 格式】C：int SACM_ S240_Initial(int Init_Index)

ASM：R1=[Init_Index]

Call F_ SACM_ S240_Initial

【功能说明】SACM_ S240 语音播放之前的初始化。

【参　　数】Init_Index=0 表示手动方式；Init_Index=1 则表示自动方式。

【返 回 值】0：代表语音模块初始化失败

1：代表初始化成功。

【备　　注】函数用于 S240 语音译码的初始化以及相关设备的初始化。

2) 【API 格式】C：void SACM_ S240_ServiceLoop(void)

ASM：Call F_ SACM_S240 _ServiceLoop

【功能说明】从资源中获取 SACM_S240 语音资料，并将其填入解码队列中。

【参　　数】无。

【返 回 值】无。

3) 【 API 格式】

C：int SACM_ S240_Play(int Speech_Index, int Channel, int Ramp_Set)；

ASM：R1=[Speech _Index]

R2=[Channel]

R3=[Ramp_Set]

Call SACM_S240_Play

【功能说明】播放资源中 SACM_ S240 语音。

【参　　数】Speech _Index 表示语音索引号。

Channel: 1.通过 DAC1 通道播放；

2.通过 DAC2 通道播放；

3.通过 DAC1 和 DAC2 双通道播放。

Ramp_Set：0. 禁止音量增/减调节；

1. 仅允许音量增调节；

2 仅允许音量减调节；

3. 允许音量增/减调节。

【返 回 值】无。

【备　　注】

① SACM_S240 的数据率为 2.4Kbps，

②Speech_Index 是定义在 resource.inc 文件中资源表（T_SACM_S240_SpeechTable）的偏移地址

③ 中断服务子程序 F_FIQ_Service_ SACM_S240 必须安置在 TMA_FIQ 中断向量上（参见第五章**中断系统**内容）。

④ 函数允许 TimerA 以所选的的数据采样率（计数溢出）中断。

程序7-4 以自动方式播放一段 SACM_ S240 语音，并自动结束。(见光盘)

SACM_S240 在自动方式下的主程序流程见图 7.9：

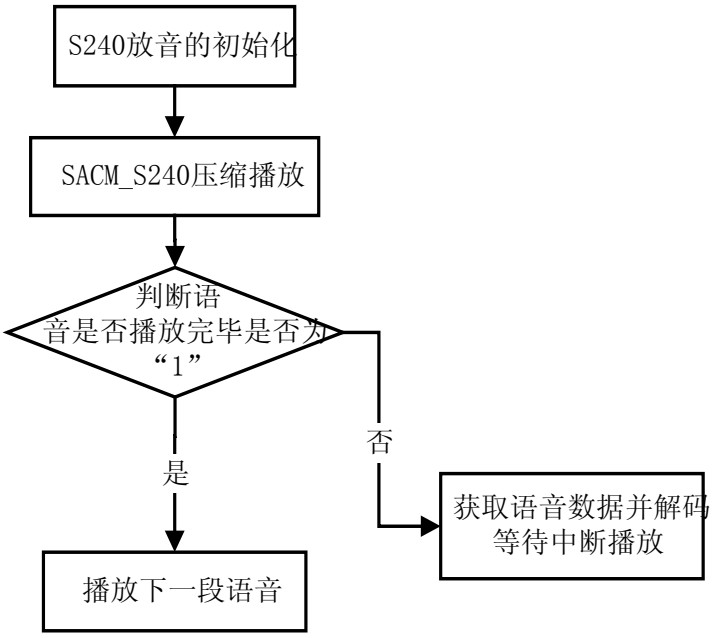

图7.9 SACM_S240 在自动方式下的主程序流程

```
//*****************************************************************************************/
// 名称：  S240
// 描述：  s240 只有自动播放方式,在中断 FIQ 的 FIQ_TMA 中断源中通过
//         主程序的 SACM_S240_ServiceLoop()对语音数据进行解码，然后将其
//         送入 DAC 通道播放
// 日期：  2002/12/6
//*****************************************************************************************/

#include "hardware.h"
#include "s240.h"
#define       DAC1              1
#define       DAC2              2
#define       Ramp_UpDn_Off     0
#define       Ramp_Up_On        1
```

```
#define          Ramp_Dn_On       2
#define          Ramp_UpDn_On     3
#define        MaxSpeechNum 3                                    //播放语音的最大个数
#define          Auto             1
//=============================================================================
// 函数： main()
//=============================================================================
 main()
{
      int SpeechIndex = 0;                                       // 选择第一首语音
      while(1)
      {
       SACM_S240_Initial(Auto);                                    //自动方式播放初始化
       SACM_S240_Play(SpeechIndex,DAC1+DAC2, Ramp_UpDn_On);// 播放第一首
       while(SACM_S240_Status()&0x01)                              //判断第一首是否播完
       SACM_S240_ServiceLoop();                                 //获取语音数据并将其填入解码队列
       SpeechIndex++;
       if(SpeechIndex==3)                                       //修改播放语音序号
       SpeechIndex=0;

    }
}
//=============================================================================
//函数: FIQ()
//=============================================================================
.TEXT
.INCLUDE      hardware.inc
.INCLUDE      S240.inc
.INCLUDE      Resource.inc
.EXTERNAL     F_FIQ_Service_SACM_S240;

.PUBLIC  _FIQ;
_FIQ:
      PUSH     R1,R4 TO [SP];              //入栈保护
      R1=0x2000;
      TEST R1,[P_INT_Ctrl];                //是否为定时器 A 中断
      JNZ L_FIQ_TimerA;
      R1=0x0800;
      TEST R1,[P_INT_Ctrl];                //是否为定时器 B 中断
      JNZ L_FIQ_TimerB;
L_FIQ_PWM:
      R1=C_FIQ_PWM;
      [P_INT_Clear]=R1;                    //清中断
      POP R1,R4 FROM [sp];                 //恢复现场
```

```
    RETI;                                  //中断返回
L_FIQ_TimerA:                              //定时器 A 中断处理
    [P_INT_Clear]=R1;                      //清中断
    CALL F_FIQ_Service_SACM_S240;          //调用 S240 中断服务函数
    POP R1,R4 FROM [sp];                   //恢复现场
    RETI;                                  //中断返回
L_FIQ_TimerB:                              //定时器 B 中断处理
    [P_INT_Clear]=R1;                      //清中断


    POP R1,R4 FROM [sp];                   //恢复现场
    RETI;                                  //中断返回
```

注：自动放音时，当语音资源文件中的资料为 FF FF FFH 时便停止播放。

4）【API 格式】C：void SACM_ S240_Stop(void)；
ASM：Call F_ SACM_ S240_Stop【功能说明】停止播放 SACM_ S240 语音。
【参　　数】无。
【返 回 值】无。

5) 【 API 格式】C：void　SACM_ S240_Pause (void)；
ASM：Call F_ SACM_ S240_Pause
【功能说明】暂停播放 SACM_ S240 语音。
【参　　数】无。
【返 回 值】无。

6) 【API 格式】C：void SACM_ S240_Resume(void)；
ASM：Call F_ SACM_ S240_Resume
【功能说明】恢复暂停播放的 SACM_ S240 语音的播放。
【参　　数】无。
【返 回 值】无。

7) 【API 格式】C：void SACM_ S240_Volume(Volume_Index)；
ASM：R1=[Volume_Index]
Call F SACM_ S240_Volume
【功能说明】在播放 SACM_ S240 语音时改变主音量。
【参　　数】Volume_Index 为音量数，音量从最小到最大可在 0~15 之间选择。
【返 回 值】无。

8) 【 API 格式】C：unsigned int SACM_ S240_Status(void)；
ASM：Call F_ SACM_ S240_ Status
[返回值]=R1
【功能说明】获取 SACM_ S240 语音播放的状态。

【参　　数】无。【返 回 值】当 R1 中 bit0=0，表示语音播放结束；bit0=1，表示语音在播放中。

9) 【 API 格式】ASM：Call F_FIQ_Service_ SACM_ S240

【功能说明】用作 SACM_S240 语音背景程序的中断服务子程序。通过前台子程序（自动方式的 SACM_S240_ServiceLoop 及手动方式的 Model-Index_Decode）对语音资料进行译码，然后将其送入 DAC 通道播放。

【参　　数】无。

【返 回 值】无。

【备　　注】SACM_ S240 语音背景子程序只有汇编指令形式，且应将此子程序安置在 TMA_FIQ 中断源上。

7.3.5 SACM_MS01

该算法较繁琐，但只要具备音乐理论、配器法和声学知识了解 SPCE 编曲格式者均可尝试。遵照 SPCE 编曲格式 用 DTM＆MIDI（音源＋MIDI 键盘＋作曲软件）的方法演奏自动生成*.mid 文件，再用凌阳 MIDI2POP.EXE 转成*.pop 文件。但需要专业设备与软件，具备键盘乐演艺技能，了解 SPCE 编曲格式。对于初学者或非专业用途一般了解放音或录放音即可。

其相关 API 函数如下所示：

```
void SACM_MS01_Initial(int Init_Index)                    //初始化
void SACM_ MS01_ServiceLoop(void)                         //获取语音资料，填入译码队列
void SACM_ MS01_Play(int Speech_Index, int Channel, int Ramp_Set)
//播放
void SACM_ MS01_Stop(void)                                //停止播放
void SACM_ MS01_Pause (void)                              //暂停播放
void SACM_ MS01_Resume(void)                              //暂停后恢复
void SACM_ MS01_Volume(Volume_Index)                      //音量控制
unsigned int SACM_ MS01_Status(void)                      //获取模块状态
void SACM_ MS01_ChannelOn(int Channel)                    //接通通道
void SACM_ MS01_ChannelOff(int Channel)                   //关闭通道
void SACM_ MS01_SetInstrument(Channel,Instrument,Mode)
//设置乐曲配器类型
```

中断服务函数：

```
ASM： F_FIQ_Service_ SACM_MS01
ASM： F_IRQ2_Service_ SACM_MS01
ASM： F_IRQ4_Service_ SACM_MS01
```

下面具体的介绍一下各个函数：

1) 【API 格式】C：void SACM_MS01_Initial(int Init_Index)

ASM：R1=[Init_Index]
Call F_ SACM_MS01_Initial

【功能说明】SACM_MS01 语音播放之前的初始化：设置中断源、定时器和播放方式（手动、自动）

【参　　数】Init_Index=0 表示手动方式；Init_Index=1 则表示自动方式。

Init_Index：
0：代表 PWM 音频输出方式
1：DAC 音频输出方式下 24K 的播放率。
2：DAC 音频输出方式下 20K 的播放率。
3：DAC 音频输出方式下 16K 的播放率。

【返 回 值】0：代表语音模块初始化失败
4：代表 SACM_MS01 初始化成功。

【备　　注】

① 该函数初始化 MS01 的译码器，以及系统时钟（System clock）TimerA、TimerB、DAC 并且以 16/20/24KHz 采样率触发 FIQ_TMA 中断。

② 初始化后会接通所有播放通道（0~5）。

③ 对于 SACM_MS01 模块，FIQ 中断服务子程序用于从前台程序（SACM_MS01_ServiceLoop）的执行过程中获取乐曲译码资料；若未来事件不是音符而是由鼓点节奏引起，则其自适应音频脉冲编码方式（ADPCM）资料将被传入 IRQ2 进行译码，然后将二者混合在一起送出 DAC 通道播放。

2) 【API 格式】 C：void SACM_ MS01_ServiceLoop(void)
ASM：Call F_ SACM_MS01 _ServiceLoop

【功能说明】从资源中获取 SACM_ MS01 语音资料，并将其填入译码队列自动译码。

【参　　数】无。

【返 回 值】无。

3）【API 格式】

C：int SACM_ MS01_Play(int Speech_Index, int Channel, int Ramp_Set);
ASM：R1=[Speech _Index]
R2=[Channel]
R3=[Ramp_Set]
Call SACM_ MS01_Play

【功能说明】开始播放一种 SACM_ MS01 音调。

【参　　数】Speech _Index 表示语音索引号。

Channel：1：通过 DAC1 通道播放；
2：通过 DAC2 通道播放；
3：通过 DAC1 和 DAC2 双通道播放。

Ramp_Set：0：禁止音量增/减调节；
1：仅允许音量增调节；
2：仅允许音量减调节；

3：允许音量增/减调节。

【返 回 值】无。

【备　　注】

①SACM_MS01 的数据率有 16Kbps\20Kbps\24Kbps 三种，可在同一模块的几种算法中自动选择一种。

② Speech_Index 是定义在 resource.inc 文件中资源表（T_SACM_MS01_SpeechTable）的偏移地址。

③中断服务子程序中 F_FIQ_Service_ SACM_MS01 必须放在 TMA_FIQ 中断向量上。

程序7-5　采用 SACM_MS01 自动方式播放一段音调：(见光盘)

前台程序：

```
#define     PWM              0
#define     DAC_24K          1
#define     DAC_20K          2
#define     DAC_16K          3
#define     DAC1             1
#define     DAC2             2
#define     Ramp_UpDn_Off    0
#define     Ramp_Up_On       1
#define     Ramp_Dn_On       2
#define     Ramp_UpDn_On     3

Main ()
{
    SACM_MS01_Initial(1);                  //MS01 初始化
    SACM_MS01_Play(Speech_1, DAC1+DAC2, Ramp_UpDn_On);     //放音
    while(SACM_MS01_Status()&0x01)   //是否播放完一段语音
  {
     SACM_MS01_ServiceLoop();        //解码并填充队列
  }
}
```

背景程序：

```
_FIQ:
        PUSH    registers
        CALL    F_FIQ_Service_SACM_MS01;     //4 通道的也配器服务程序
        CLEAR   interrupt flag
        POP     registers
        RETI;
_IRQ2:
        PUSH    registers
        CALL    F_IRQ2_Service_SACM_MS01;    //2 通道鼓点节奏服务程序
        CLEAR   interrupt flag
```

```
        POP     registers
        RETI;
_IRQ4:
      PUSH    registers
      CALL    F_IRQ4_Service_SACM_MS01;    //控制节拍服务程序
      CLEAR   interrupt flag
      POP     registers
      RETI;
```

4）【API 格式】C：void SACM_ MS01_Stop(void)；
ASM：Call F_ SACM_ MS01_Stop【功能说明】停止播放 SACM_ MS01 乐曲。
【参　　数】无。
【返 回 值】无。

5) 【API 格式】C：void SACM_ SACM_ MS01_Pause (void)；
ASM：Call F_ SACM_ MS01_Pause
【功能说明】暂停播放 SACM_MS01 乐曲。
【参　　数】无。
【返 回 值】无。

6) 【 API 格式】C：void SACM_ MS01_Resume(void)；
ASM：Call F_ SACM_ MS01_Resume
【功能说明】恢复暂停播放的 SACM_ MS01 语音或乐曲。
【参　　数】无。
【返 回 值】无。

7) 【API 格式】C：void SACM_ MS01_Volume(int Volume_Index)；
ASM：R1=[Volume_Index]
Call F SACM_ MS01_Volume
【功能说明】在播放 SACM_ MS01 语音时改变主音量。
【参　　数】Volume_Index 为音量数，音量从最小到最大可在 0~15 之间选择。
【返 回 值】无。

8) 【API 格式】C：unsigned int SACM_ MS01_Status(void)；
ASM：Call F_ SACM_ MS01_ Status
[Return_Value]=R1
【功能说明】获取 SACM_ MS01 合成音乐播放的状态。
【参　　数】无。
【返 回 值】 当 R1 中 bit0=0，表示语音播放结束；bit0=1，表示语音在播放中。
还有其它状态位（bit8~ bit13），见图 7.10。

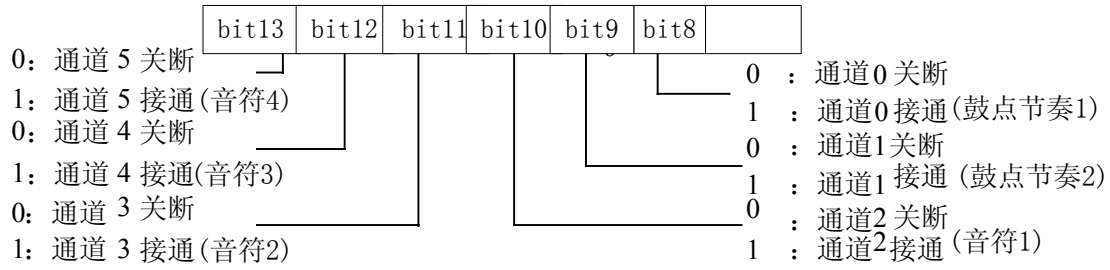

图7.10 SACM_MS01 部分状态返回值

9) 【 API 格式】C：void SACM_MS01_ChannelOn(int Channel)；

ASM：R1=[Channel]

Call F_SACM_MS01_ChannelOn

【功能说明】将 SACM_MS01 乐曲播放通道之一接通。

【参　　数】Channel 为 0~5 之间整数，其中 0，1 代表鼓点节奏通道，2~5 则代表音符通道。

【返 回 值】无。

10) 【 API 格式】C：void SACM_MS01_ChannelOff(int Channel)；

ASM：R1=[Channel]

Call F_SACM_MS01_ChannelOff

【功能说明】将 SACM_MS01 乐曲播放通道之一关断。

【参　　数】Channel 为 0~5 之间整数，其中 0，1 代表鼓点节奏通道，2~5 则代表音符通道。【返 回 值】无。

程序7-6

```
int main()
{
    SACM_MS01_Initial(DAC_24K);        //选择 DAC 24K 播放方式,且 6 个通道都接通
    SACM_MS01_ChannelOff(0);           //通道 0 被关断(鼓点节奏 1)
    SACM_MS01_ChannelOff(5);           //通道 5 被关断(音符 4,频率 Tone 方式)
    SACM_MS01_Play(SongIndex,DAC1+DAC2);        //开始播放乐曲
    while(SACM_MS01_Status()&0x01)              //若状态返回值的 bit0=1
    {
      SACM_MS01_ServiceLoop();         // SACM_MS01 乐曲播放的译码程序
    }
}
```

11) 【API 格式】

C：void SACM_MS01_SetInstrument(int Channel, int Instrument, int Mode);

ASM：R1=[Channel]

R2=[Instrument]
R3=[Mode]
Call F_SACM_MS01_SetInstrument

【功能说明】在 SACM_MS01 的一个播放通道上改变乐曲配器类型。

【参　　数】Channel=0~5；其中 0，1 代表鼓点节奏通道，2~5 则代表音符通道。
对于通道 0、1，Instrument=0~Max_Drum#；表不同的鼓点节奏；
对于通道 2~5，Instrument=0~34；表不同的乐曲配器类型。
Mod=0，1；分别代表配器类型可以或不可通过乐曲事件改变。

【返 回 值】无。

12）【API 格式】ASM：Call F_FIQ_Service_ SACM_ MS01
ASM：Call F_IRQ2_Service_ SACM_ MS01
ASM：Call F_IRQ4_Service_ SACM_ MS01

【功能说明】 SACM_MS01 模块，FIQ 中断服务子程序用于从前台程序（SACM_MS01_ ServiceLoop）的执行过程中获取乐曲译码资料；若未来事件不是音符而是由鼓点节奏引起，则其自适应音频脉冲编码方式（ADPCM）资料将被传入 IRQ2 进行译码，然后将二者混合在一起送出 DAC 通道播放。

【参　　数】无。

【返 回 值】无。

【备　　注】

① SACM_MS01 语音背景子程序只有汇编指令形式

② 中断服务子程序必须在 TMA_FIQ 中断源上。

③ 应将两个额外的中断服务子程序分别安置在 IRQ2_TMB 和 IRQ4_1K 中断源上。

7.3.6 SACM_DVR

SACM-DVR 具有录音和放音功能，并采用 SACM_A2000 的算法，录音时采用 16K 资料率及 8K 采样率获取语音资源，经过 SACM_A2000 压缩后存储在扩展的 SRAM 628128A 里，录满音后自动开始放音。

其相关 API 函数如下所示：

```
int SACM_DVR_Initial(int Init_Index)                    //初始化
void SACM_DVR_ServiceLoop(void)                         //获取资料，填入译码队列
void SACM_DVR_Encode(void)                              //录音
SACM_DVR_StopEncoder();                                 //停止编码
SACM_DVR_InitEncoder(RceMonitorOn)                      //初始化编码器
void SACM_DVR_Stop(void)                                //停止录音
void SACM_DVR_Play(void)                                //开始播放
unsigned int SACM_DVR _Status(void)                     //获取 SACM_DVR 模块的状态
void SACM_DVR _InitDecoder(int Channel)                 //开始译码
void SACM_DVR _Decode(void)                             //获取语音资料并译码，中断播放
SACM_DVR_StopDecoder();                                 //停止解码
```

```
unsigned int SACM_DVR _ TestQueue(void)                    //获取语音队列状态
int SACM_DVR _Fetchqueue(void)                             //获取录音编码数据
void SACM_DVR_FillQueue(unsigned int encoded-data)         //填充资料到语音队列，等待播放
int  GetResource(long Address) ——(Manual)                  // 从资源文件里获取一个字型语音资料
```

中断服务函数：

```
Call F_FIQ_Service_ SACM_DVR                               //playing
Call F_IRQ1_Service_ SACM_DVR                              //recode
```

具体函数如下：

1) 【API 格式】C：void SACM_ DVR _Initial(int Init_Index)

ASM：R1=[Init_Index]

Call F_ SACM_ DVR _Initial

【功能说明】SACM_ DVR 语音播放之前的初始化：设置中断源、定时器以及播放方式（自动、手动）

【参　　数】Init_Index=0 表示手动方式；Init_Index=1 则表示自动方式。

【返 回 值】无

【备　　注】

① 对于 SACM_DVR 模块，需要一些 I/O 口来连接外部的 SRAM，用以存放录音资料。

② 录放音的格式采用 SACM_A2000。

2) 【API 格式】C：void SACM_DVR_ServiceLoop(void)

ASM：Call F_ SACM_DVR _ServiceLoop

【功能说明】在录音期间从 ADC 通道获取录音资料，且将其以 SACM_A2000 格式进行编码后存入外接 SRAM 中；而在播放期间从 SRAM 中获取语音资料，对其进行解码，然后等候中断服务子程序将其送出 DAC 通道。

【参　　数】无。

【返 回 值】无。

3）【API 格式】C：void SACM_DVR_Encode(void);

ASM：Call F_ SACM_DVR_ Encode

【功能说明】开始以自动方式录制声音资料到外接 SRAM 中。

【参　　数】无。

【返 回 值】无。

【备　　注】该函数仅适用于 SACM_DVR 模块，且只有自动方式。

4）【API 格式】C：void SACM_DVR _Stop(void);

ASM：Call F_ SACM_DVR _Stop

【功能说明】 以自动方式停止录音。

【参　　数】无。

【返 回 值】无。

5）【API 格式】

C：int SACM_DVR _Play(int Speech_Index, int Channel, int Ramp_Set)；

ASM：Call SACM_DVR _Play【功能说明】以自动方式播放外接 SRAM 中的录音资料。

【参　　数】无

【返 回 值】无。

【备　　注】该函数仅使用于自动方式下。

6）【API 格式】C：unsigned int SACM_DVR _Status(void)；

ASM：Call F_ SACM_DVR _ Status

[返回值]=R1

【功能说明】获取 SACM_DVR 模块的状态。

【参　　数】无。

【返 回 值】 当 R1 中 bit0=0，表示语音播放结束；bit0=1，表示语音在播放中。

SACM_DVR 模块的状态返回值，如图 7.11 所示。

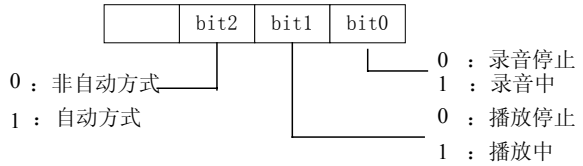

图7.11 SACM_DVR 状态返回值

【备　　注】该函数仅使用于 DVR 的手动方式下。

7）【API 格式】C：void SACM_DVR_InitDecoder(int Channel)；

ASM：Call F_SACM_DVR_Decode

【功能说明】开始对 SACM_DVR 语音资料以非自动方式（编程控制）进行译码。

【参　　数】Channel=1，2，3；分别表示使用 DAC1、DAC2 通道以及 DAC1 和 DAC2 双通道。

【返 回 值】无。

【备　　注】用户只能通过非自动方式对语音资料解压缩。

8）【API 格式】C：void SACM_DVR_Decode(void)；

ASM：Call F_ SACM_DVR_Decode

【功能说明】从语音队列里获取的 SACM_DVR 语音资料，并进行译码，然后通过中断服务子程序将其送入 DAC 通道播放。

【参　　数】无。

【返 回 值】无。

【备　　注】用户仅能通过非自动方式对语音资料进行译码。

9) 【 API 格式】C：unsigned int SACM_DVR_TestQueue(void)；
ASM：Call F_SACM_DVR_TestQueue
[返回值]=R1
【功能说明】获取语音队列的状态。
【参　　数】无。
【返 回 值】R1=0，语音队列不空不满
=1，语音队列满
=2；语音队列空。
【备　　注：用户仅能通过非自动方式测试语音队列状态。

10) 【API 格式】C：int SACM_DVR _FetchQueue(void)；
ASM：Call F_SACM_DVR _FetchQueue
[Return_Value]=R1
【功能说明】获取录音编码（SACM_A2000）数据。
【参　　数】无。
【返 回 值】16 位录音资料。
【备　　注】
① 采用--SACM_A2000 编码格式编码
② 仅用于非自动方式下

程序7-7 外扩 SRAM 需要 SRAM 写初始化（InitWriteSRAM）,及写（WriteSRAM）子函数。

```
SACM_DVR_Initial(Manual);
Addr=0;
InitWriteSRAM();                          //用户需外接 SRAM
SACM_DVR_InitEncoder();
while(Addr<SRAM_Size)
{
    SACM_DVR_Encode();                    //获取数据并译码
    If(SACM_DVR_TestQueue()!=Empty)       //若队列不空,则存储资料
    {
      ret=SACM_DVR_FetchQueue();          //从队列中得到资料
      writeSRAM(Addr,ret);                //存入用户存储区
      Addr+=2;                            //两个 8 位 SRAM 存储一个 16 位资料
    }
}
```

11）【API 格式】C：void SACM_DVR _FillQueue(unsigned int encoded-data)；
ASM：R1=[语音编码资料]
Call F_ SACM_DVR _FillQueue
【功能说明】 填充 SACM_A2000 语音资料到 DVR 译码器等待播放
【参　　数】encoded-data 为语音编码资料。
【返 回 值】无。

【备　　注】
① 语音资料格式为--SACM_A2000 编码格式。
② 从语音队列里至少每 48ms 获取 48 个字资料（16K 资料采样率）。
③ 仅用于非自动方式下。

12）【API 格式】C：int GetResource(long Address)；
【功能说明】从资源文件里获取一个字型语音资料。
【参　　数】无。
【返 回 值】一个字型语音资料。

13）【API 格式】ASM：Call F_FIQ_Service_ SACM_DVR
ASM：Call F_IRQ1_Service_SACM_DVR
【功能说明】　用作 SACM_DVR 语音背景程序的中断服务子程序。通过前台子程序（自动方式的 SACM_DVR _ServiceLoop 及手动方式的 SACM_DVR _Decode）对语音资料进行译码，然后将其送入 DAC 通道播放。即 FIQ 中断服务子程序用于声音播放的背景程序；而 IRQ1 中断服务子程序则用于声音录制的背景程序。
【参　　数】无。
【返 回 值】无。
【备　　注】SACM_DVR 语音背景子程序只有汇编指令形式，且应将此子程序安置在 TMA_FIQ 中断源上。额外的中断服务子程序安置在 IRQ1_TMA 中断源上。

程序7-8 DVR 以自动方式录放音

前台程序：

```
int key;
Main()
{
System_Initial();                                    //键盘初始化
        SACM_DVR_Initial(Auto);
        while(1)
    {
            Key = SP_GetCh();                        //获取键值
            switch(Key)
            {
                case 0x0000:
                    break;
                case 0x0001:
                    SACM_DVR_Record();    //录音，存储资料到 SRAM
                    break;
                case 0x0002:
                    SACM_DVR_Stop();      //停止录/放音
                    break;
```

```
                    case 0x0004:
                        SACM_DVR_Play();        //从 SRAM 中取出语音资料并播放
                        break;
                    default:
                        break;
                }
                System_ServiceLoop();           // 键扫描
                SACM_DVR_ServiceLoop();
        }                                       // while(1)结束
    }
```

背景程序：

```
_FIQ:
        PUSH registers;
        CALL    F_FIQ_Service_SACM_DVR;         // 放音
        CLEAR   interrupt flag
        POP registers;
        RETI;
_IRQ1:
        PUSH registers
        CALL    F_IRQ1_Service_SACM_DVR;        //录音
        CLEAR   interrupt flag
        POP registers;
reti;
```

注：对于函数 System_Initial()、System_ServiceLoop()和 SP_GetCh()详见具体的实验。

程序7-9 非自动方式：(见光盘)

DVR 手动方式主程序流程见图 7.12：

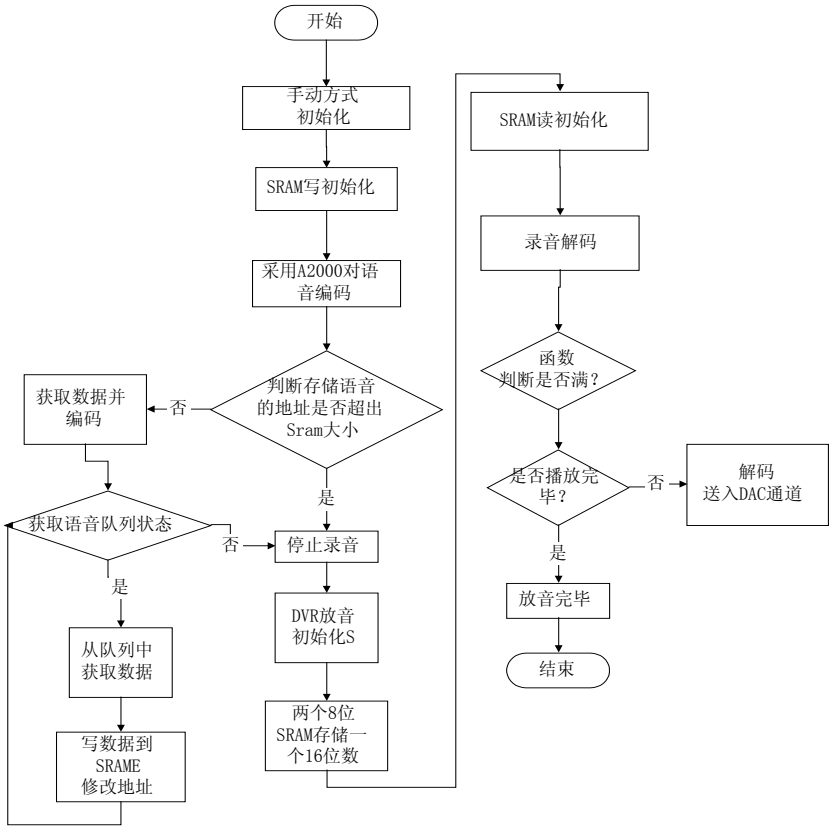

图7.12 DVR 手动方式主程序流程

```
#DEFINE      Manual              0
#DEFINE  Auto                    1

#DEFINE  SRAM_Size               0xffff-2

#DEFINE  RceMonitorOff           0
#DEFINE  RceMonitorOn            1

#DEFINE  DAC1                    1
#DEFINE  DAC2                    2

#DEFINE  Full                    1
#DEFINE  Empty                   2
**************录音**************
main(){
      SACM_DVR_Initial(Manual);              //手动方式初始化
```

```
        Addr = 0;                                   //定义语音存放的首址变量
        InitWriteSRAM();
         SACM_DVR_InitEncoder(RceMonitorOn);    //开始对 A2000 的语音资料以非自动方式编
码
    while(Addr<SRAM_Size)                           //判断存储语音的地址是否超出存储单元的
大小
    {
         SACM_DVR_Encode ();                        //获取数据并编码
         if(SACM_DVR_TestQueue()!= Empty)
         {
                  Ret=SACM_DVR_FetchQueue();    //从队列中获取资料
                  WriteSRAM(Addr,Ret);          //  存入用户定义的存储单元区
                  Addr+=2;                      //两个 8 位 SRAM 存储一个 16 位资料
         }
    }
                  SACM_DVR_StopEncoder();
    }
     *********放音*********
    main()
{
    SACM_DVR_Initial(Manual);                       //非自动方式播放的初始化

     InitReadSRAM();

     Addr=0;

        SACM_DVR_InitDecoder(DAC1);         //开始对 A2000 的语音资料以非自动方式译码
    while(1)
    {
     if(SACM_DVR_TestQueue()!=Full)         //测试并获取语音队列的状态
         {
              Ret =ReadSRAM(Addr);          //从存储区里获取一个字型语音资料
             SACM_DVR_FillQueue(Ret);       //获取语音编码资料并填入语音队列等候译码

             Addr+=2;
         }
          if(Addr<SRAM_Size)                //如果该段语音播完，即到达末地址时
          SACM_DVR_Decode ();               //获取资源并进行译码，再通过中断服务子程序
送入 DAC 通道播放
          else
          SACM_DVR_StopDecoder();           //否则，停止播放
         }
       }
```

背景程序：

```
_FIQ:
        PUSH R1,R5 to [sp];
        CALL     F_FIQ_Service_SACM_DVR; //语音播放中断
        R1=0xa800
        [P_INT_Clear]=R1
        POP R1,R5 from [sp];
        RETI;

_IRQ1:
        PUSH R1,R5 to [sp];
        CALL     F_IRQ1_Service_SACM_DVR; //语音录制中断
        R1=0x1000
        [P_INT_Clear]=R1
        POP R1R5 from [sp];
        RETI;
```

7.4　语音压缩方法

对于常用的 SACM_A2000 和 SACM_S480 两种放音算法要涉及到语音资源的添加问题，即将 WAV 文件按照我们需要的压缩比进行压缩，变成资源表形式在程序中调用。这里介绍两种语音压缩的方法：DOS 下和 WINDOWS 下（我们建议用户使用 WINDOWS 方式压缩，因为它操作比较方便，不容易出错）。

(1)DOS 下的压缩：

SACM_A2000：

1) PC 机采用 8K16 位单声道录制一个 WAV 语音文件

2) 用 A2000 压缩生成 16k(或 20k,24k)压缩率的文件

4) 在 MS-DOS 下：

e:\>sacm2000.exe 16 *.wav *.out *.16k

或 (e:\>sacm2000.exe 20 *.wav *.out *.20k

e:\>sacm2000.exe 24 *.wav *.out *.24k)

SACM_S480：

1) PC 机采用 8K16 位单声道录制一个 WAV 文件

2) 用 s480 压缩生成 4.8k（或 7.2k）压缩率的文件

3) 在 MS-DOS 下：

e:\>sacm.exe *.wav *.48k *.out –s48

或（e:\>sacm.exe *.wav *.72k *.out –s72）

图 7.13 是凌阳音频压缩编码(SACM)方法的流程：

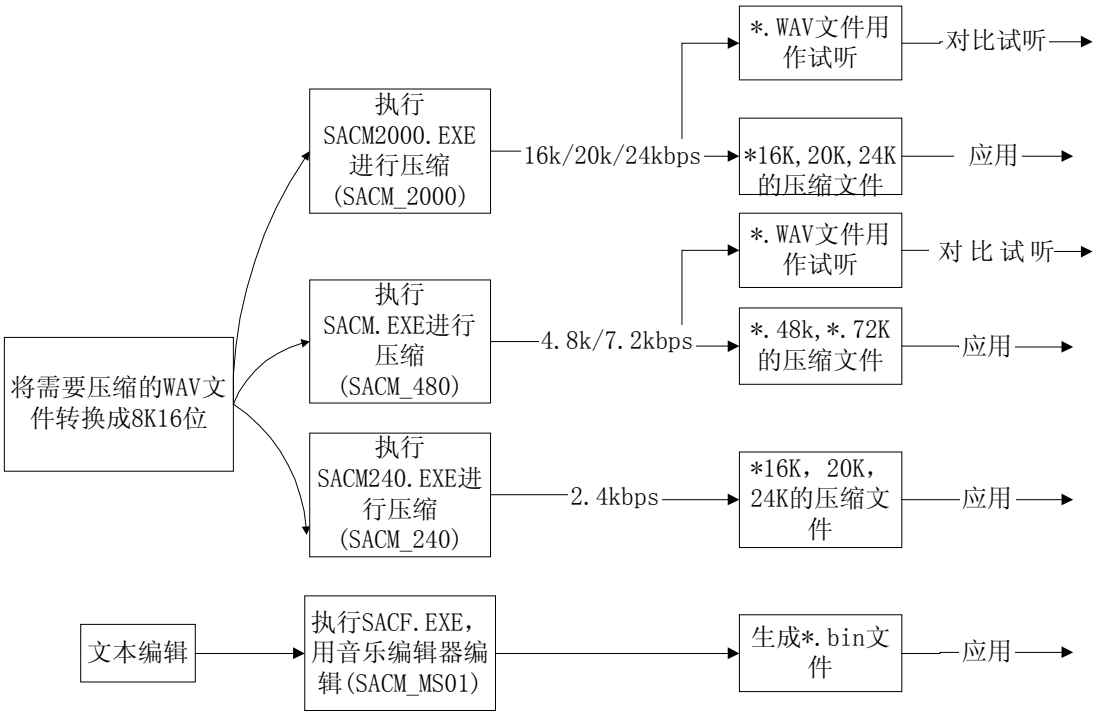

图7.13 凌阳音频压缩编码(SACM)流程

(2) WINDOWS 下的压缩：

0 是用于压缩的 windows 工具，可以选择一个或多个 WAV 文件进行压缩，具体步骤可根据提示来操作。

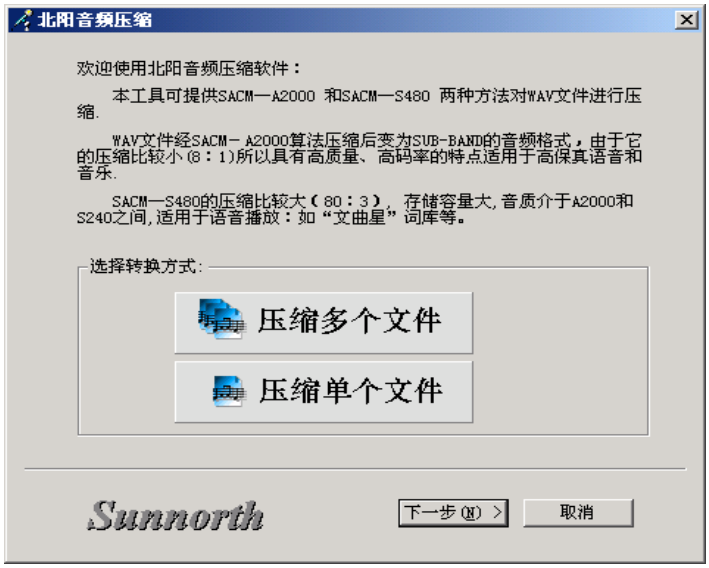

图7.14 用于压缩的 windows 工具

7.5 键控放音程序介绍

首先介绍一下程序模块，程序模块中的文件大致分为两类：一类是程序，另一类则是程序接口。所谓接口一般是针对高一级的程序模块而言。通过接口，高一级的程序模块可以调用本级程序模块中的子程序或函数，或者使用本级模块中定义的全局变量。这样做无疑会大大增加软件的可维护性。而程序既可以用 C 语言编写，亦可用μ'nSP™的汇编语言编写。

随着对语音编程越来越熟悉，可以考虑加入一些模块化程序，如键盘：这样可以按键控制语音播放、停止、暂停、恢复以及音量的大小等，这里我们为用户提供了一个资源使用模块，即接口文件 Key.inc，其中定义了供系统级调用的与键扫描相关的一些子程序。如：键扫描初始化子程序、键扫描子程序、键扫描防抖动处理子程序以及获得键码子程序等等。模块中的 Key.asm 文件中则是上述接口中定义的各子程序的程序实体，以及定义出程序中所需要的全局或局部变量。具体结构见下图。

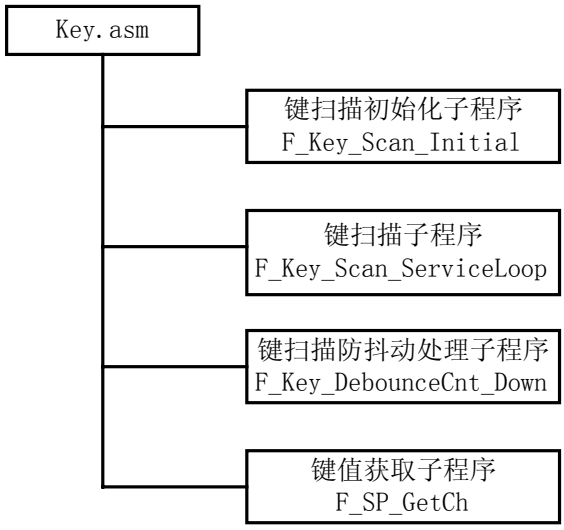

图7.15 键盘程序结构

下面我们具体介绍一下几个常用的键盘 API 函数：

```
F_ Key_Scan_Initial                    //键初始化
F_Key_Scan_ServiceLoop                 //键盘扫描
F_Key_DebounceCnt_Down                 //键盘防抖动处理
F_ SP_GetCh()                          //获取键值
```

1）【API 格式】C：void Key_Scan_Initial（void）
ASM：Call F_ Key_Scan_Initial
【功能说明】 键盘扫描初始化。
【参　　数】无。
【返 回 值】无。

2）【API 格式】C：void Key_Scan_ServiceLoop（void）
ASM：Call　F_Key_Scan_ServiceLoop
【功能说明】键盘扫描服务循环。
【参　　数】无。
【返 回 值】无。

3）【API 格式】C：void Key_DebounceCnt_Down（void）
ASM：Call F_Key_DebounceCnt_Down
【功能说明】键盘扫描过程中对键抖动的处理。
【参　　数】无。
【返 回 值】无。

4）【API 格式】C：unsigned int SP_GetCh（void）
ASM：Call F_SP_GetCh()
[返回值]=R1
【功能说明】从扫描缓冲区内获得键值，并将缓冲区的键值清零。
【参　　数】无。
【返 回 值】被按下的键值。

然而在介绍了键盘 API 函数后我们就会想到要在程序中哪里去调用，这里为了使程序更具模块化我们又为大家提供了一个系统资源模块：它是建立在上述诸多模块之上的一个模块，其中许多子程序都调用了在它之下模块中的子程序。为了让上一级的程序调用本模块 System.asm 文件中的子程序，有一个接口文件 System.inc。这里我们一般只需要三条调用语句，当然用户也可以在这里面添加一些模块，具体实现可以参考下面模块框图：

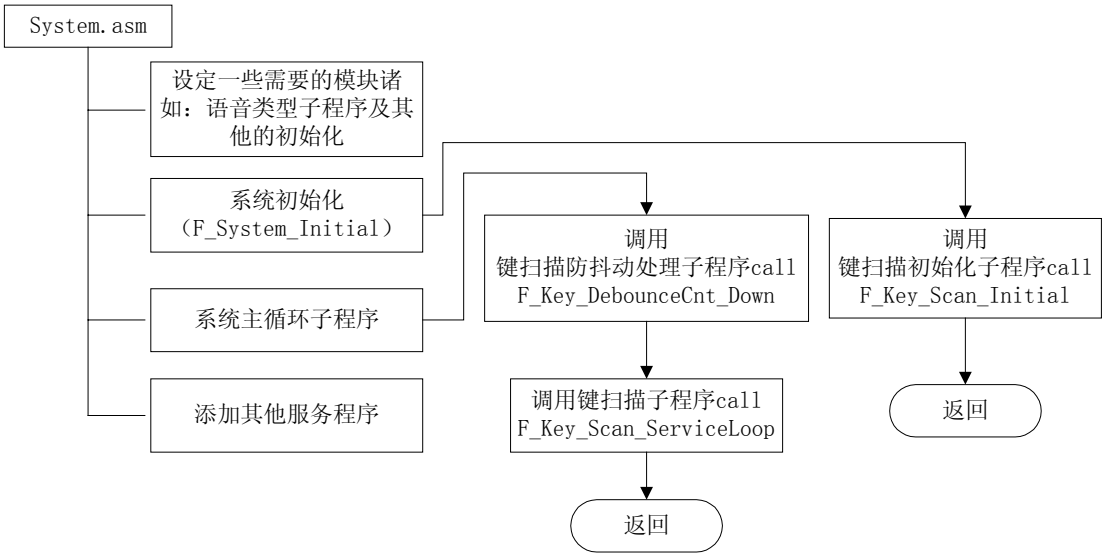

图7.16 调用键盘程序的系统程序结构

这样，对于我们的一个含键盘程序的两个重要模块介绍完了，下面的问题是如何在主程序中实现整体调用，见下图：

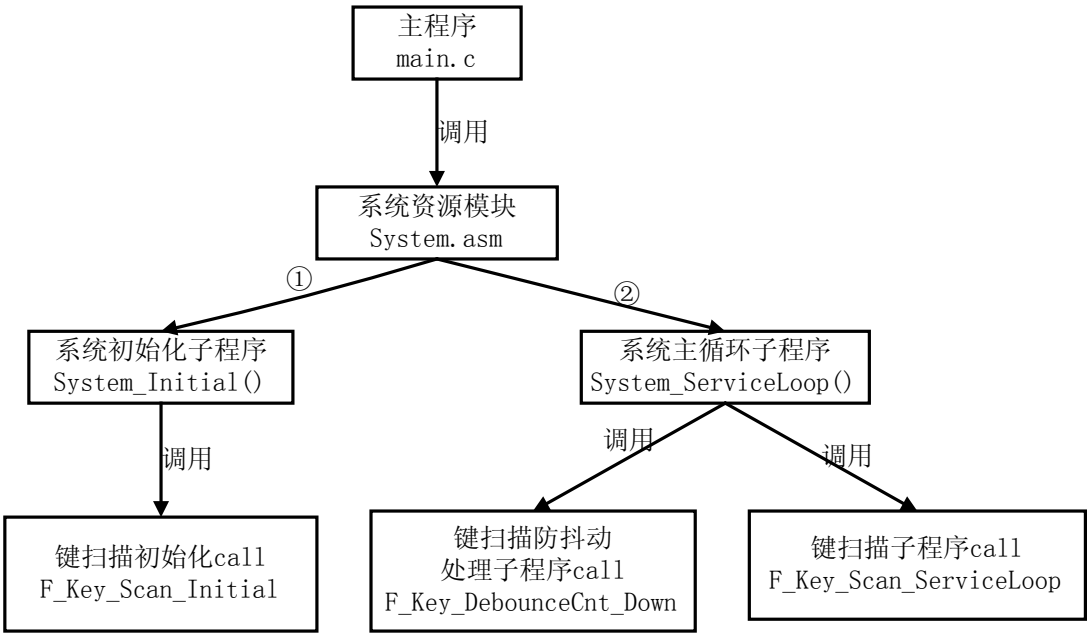

图7.17 主程序结构

程序7-10 (见光盘)

同前面放音一样在最后我们通过一个 SACM_A2000 按键控制自动放音的例子来熟悉一下整个程序的执行过程：

```
********主程序(main.c)**********
main()
{
    int Key = 0;                                    // 初始化键值
    int SpeechIndex = 0;                            // 初始化语音目录索引号
    int VolumeIndex = 7;                            //初始化音量
    Ret = System_Initial();
        Ret = SACM_A2000_Initial(Auto);
        SACM_A2000_Play(SpeechIndex,DAC1+DAC2,Ramp_UpDn_On);
        //播放
        while(1)
{
    Key = SP_GetCh();
        switch(Key)
{
        case 0x00:
          break;
        case 0x01:
          SACM_A2000_Play(SpeechIndex,DAC1+DAC2, Ramp_UpDn_On);
            // 播放
          break;
        case 0x02:
          SACM_A2000_Stop();                        //停止放音
          break;
        case 0x04:
          SACM_A2000_Pause();                       //暂停放音
          break;
       case 0x08:
          SACM_A2000_Resume();                      // 暂停后的恢复
          break;
        case 0x10:
          VolumeIndex++;
          if(VolumeIndex > MaxVolume)
          VolumeIndex = MaxVolume;
          SACM_A2000_Volume(VolumeIndex);           //音量增加
          break;
        case 0x20:
          if(VolumeIndex == 0)
```

```
            VolumeIndex = 0;
          else
            VolumeIndex--;
            SACM_A2000_Volume(VolumeIndex);          //音量减
            break;
          case 0x40:
            SpeechIndex++;                           // 播放下一首
            if(SpeechIndex == MaxSpeechNum)
            SpeechIndex = 0;
            SACM_A2000_Play(SpeechIndex,DAC1+DAC2, Ramp_UpDn_On);
            break;
          case 0x80:
            if(SpeechIndex == 0)                     //播放前一首
            SpeechIndex = MaxSpeechNum;
            SpeechIndex--;
            SACM_A2000_Play(SpeechIndex,DAC1+DAC2, Ramp_UpDn_On);
            break;
          default:
           break;
      }
          System_ServiceLoop();                      //调用系统初始化
          SACM_A2000_ServiceLoop();
          // 获取 A2000 资料并填入译码队列等待播放
}
}
**********系统子程序（System.asm）****************
.public _System_Initial;
.public    F_System_Initial;
_System_Initial: .PROC
F_System_Initial:
          CALL     F_Key_Scan_Initial;             // 键盘扫描
                                          //可以添加一些语音类型子程序或初始化内容
          RETF
          .ENDP;.
.public _System_ServiceLoop;
.public    F_System_ServiceLoop;
_System_ServiceLoop: .PROC
F_System_ServiceLoop:
          CALL     F_Key_DebounceCnt_Down;         // 调用键扫描防抖动处理子程序
          CALL     F_Key_Scan_ServiceLoop;         //调用键扫描子程序
          // 可在次添加其它服务程序
          RETF;
          .ENDP;
**********键盘子程序（Key.asm）**********
```

```
.ram
.var R_DebounceReg;
.DEFINE   C_DebounceCnt         0x0002;
. var      R_DebounceCnt;
. var      R_KeyBuf;
. var      R_KeyStrobe;
.CODE
***键扫描初始化*********************************
F_Key_Scan_Initial:
          R1 = 0x0000;
          [R_DebounceReg] = R1;                    // R_DebounceReg 初始化为 0
          [R_KeyBuf] = R1;                         // R_KeyBuf 初始化为 0
          [R_KeyStrobe] =R1;                       // R_KeyStrobe 初始化为 0
          r1 = C_DebounceCnt;
          [R_DebounceCnt] = R1;                    //设定记数初值
          RETF;
***键扫描*************************************
F_Key_Scan_ServiceLoop:
          R1 = [P_IOA_Data];                       // 由 IOA 口获取键值
          R1 = R1 and 0xff;                        //保留键值
          R2 = [R_DebounceReg];                    //将上次获取的键值送给 R2
          [R_DebounceReg] = R1;                    //将当前键值送给 R_DebounceReg
          CMP R2,[R_DebounceReg];                  //比较两次采样的键值是否相同
          JE   L_KS_StableTwoSample;               //是，则转
          R1 = C_DebounceCnt;                      //否，则设定记数的时间
          [R_DebounceCnt] = R1;
          RETF;
L_KS_StableTwoSample:
          R1 = [R_DebounceCnt];                    //判断记数值是否为零
          JZ   L_KS_StableOverDebounce;            //是，则转
          RETF;

L_KS_StableOverDebounce:
      R2 = [R_DebounceReg];                        //将当前采样的键值送给 R2
      R1 = [R_KeyBuf];                             //暂存上次 R_KeyBuf 里的键值到 R1
      [R_KeyBuf] = R2;                             //当前键值送给 R_KeyBuf
      R1 = R1 xor 0x00ff;                          //R1 低 8 位取返
     R1 = R1 and [R_KeyBuf]                        //R1 低 8 位取返后和当前的键值与
      R1 = R1 and 0x00ff;                          //保留与的结果既最后确定的键值
      R1 = R1 OR [R_KeyStrobe];
      [R_KeyStrobe] = R1;                          //送键值到 R_KeyStrobe 单元
          RETF;
***键扫描防抖动处理*****************************
F_Key_DebounceCnt_Down:
```

```
        R1 = [R_DebounceCnt];
        // 读取记数单元 R_DebounceCnt 的值
        JZ   L_DebounceCntZero;
        // 如果 R_DebounceCnt 单元为零则停止记数
        R1 -= 0x0001;
        [R_DebounceCnt] = R1;
L_DebounceCntZero:
         RETF;
*******取键值**********************************
_SP_GetCh:
        F_SP_GetCh:
        R1 = [R_KeyStrobe];                          // 取键值
        R2 = 0x0000;                                 // 清零 R_KeyStrobe 单元
        [R_KeyStrobe] = R2;
        RETF;
```

（注意：以上程序模块只介绍主要程序部分，未加入伪指令部分内容，仅供学习参考）

7.6 语音辨识

在前面我们已经介绍过语音辨识的一些相关的内容，在这里我们给出 SPCE061 的特定语者辨识 SD（Speaker Dependent）的一个例子以供有兴趣者参考。SD 即语音样板由单个人训练，也只能识别训练某人的语音命令，而他人的命令识别率较低或几乎不能识别。

图 7.18 是语音辨识的一个整体框图：

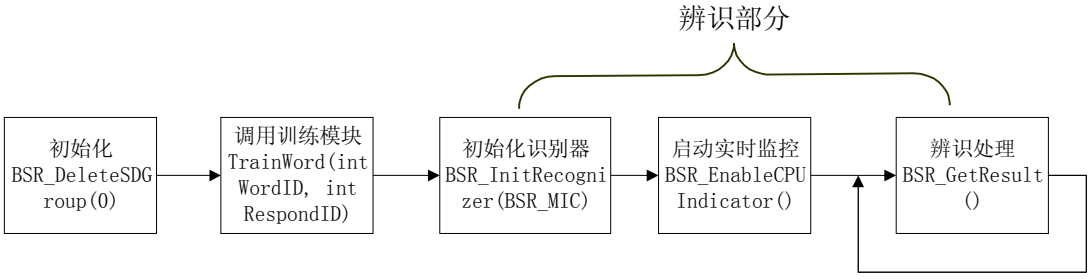

图7.18 语音辨识原理框图

同样语音辨识也将其一些功能作成模块，并通过 API 调用来实现这些功能，在这里我们为大家介绍一些常用的 API 函数，如果有兴趣者可以登陆我们的网站去获得更

多的相关内容
初始化：

【API 格式】C：　int BSR_DeleteSDGroup(0)；
ASM：F_BSR_DeleteSDGroup(0)
【功能说明】SRAM 初始化。
【参　　数】该参数是辨识的一个标识符，0 代表选择 SRAM,并初始化。
【返 回 值】当 SRAM 擦除成功返回 0，否则，返回－1。

训练部分：

1) 【API 格式】C：int　BSR_Train　(int　CommandID, int TraindMode)；
ASM：F_BSR_Train
【功能说明】训练函数。
【参　　数】
CommandID：命令序号，范围从 0x100 到 0x105,并且对于每组训练语句都是唯一的。
TraindMode：训练次数，要求使用者在应用之前训练一或两遍：
BSR_TRAIN_ONCE:要求训练一次。
BSR_TRAIN_TWICE 要求训练两次。
【返 回 值】训练成功，返回 0；没有声音返回－1；训练需要更多的语音数据来训练，返回－2；当环境太吵时，返回－3；当数据库满，返回－4；当两次输入命令不通，返回－5；当序号超出范围，返回－6。
【备　　注】
① 在调用训练程序之前，确保识别器正确的初始化。
② 训练次数是 2 时，则两次一定会有差异，所以一定要保证两次训练结果接近
③ 为了增强可靠性，最好训练两次，否则辨识的命令就会倾向于噪音
④ 调用函数后，等待 2 秒开始训练，每条命令只有 1 .3 秒，也就是说，当训练命令超出 1.3 秒时，只有前 1.3 秒命令有效。

辨识部分：

1）【API 格式】C：　void BSR_InitRecognizer(int AudioSource)
ASM：F_BSR_InitRecognizer
【功能说明】辨识器初始化。
【参　　数】 定义语音输入来源。通过 MIC 语音输入还是 LINE_IN 电压模拟量输入。
【返 回 值】无。

2）【API 格式】C：int　BSR_GetResult()；
ASM：F_ BSR_GetResult
【返回值】=R1
【功能说明】辨识中获取数据。
【参　　数】 无。
【返 回 值】

当无命令识别出来时，返回 0；
识别器停止未初始化或识别未激活返回－1；
当识别不合格时返回－2；
当识别出来时返回命令的序号。

【备　　注】 该函数用于启动辨识，BSR_GetResult()；

3）【API 格式】C： void BSR_StopRecognizer(void)；
ASM：F_ BSR_StopRecognizer
【功能说明】停止辨识。
【参　　数】无。
【返 回 值】 无。
【备　　注】该函数是用于停止识别，当调用此函数时，FIQ_TMA 中断将关闭。

中断部分：
【API 格式】 ASM：_BSR_InitRecognizer
【功能说明】 在中断中调用，并通过中断将语音信号送 DAC 通道播放。
【参　　数】无。
【返 回 值】无。
【备　　注】
① 该函数在中断 FIQ_TMA 中调用
② 当主程序调用 BSR_InitRecognizer 时，辨识器便打开 8K 采样率的 FIQ_TMA 中断并开始将采样的语音数据填入辨识器的数据队列中。
③ 应用程序需要设置一下程序段在 FIQ_TMA 中：

```
.PUBLIC _FIQ
.EXTERNAL _BSR_FIQ_Routine                    //定义全局变量
.TEXT
_FIQ:
    PUSH R1,R4 to [SP]                        //寄存器入栈保护
    R1 = [P_INT_Ctrl]
    CALL _BSR_FIQ_Routine                     //调用子程序
    R1 = 0x2000                               //清中断标志位
    [P_INT_Clear] = R1
    POP R1,R4 from [SP];                      //寄存器组出栈
    RETI
END
```

其中实时监控是用来观察辨识是否正常工作，如果辨识正常则会产生一 16ms 连续稳定方波如图 7.19 否则如果 CPU 超载则会产生不稳定波形如图 7.20，此时需要删除命令，或是优化程序否则会丢失语音数据产生辨识出现错误的信息 .

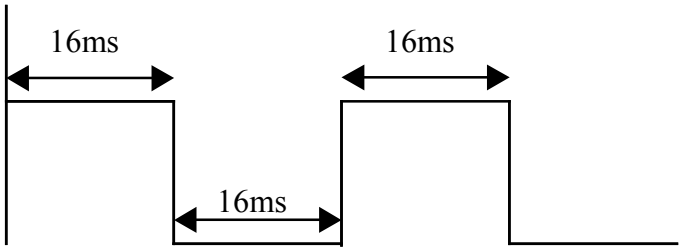

图7.19 辨识正常产生的方波

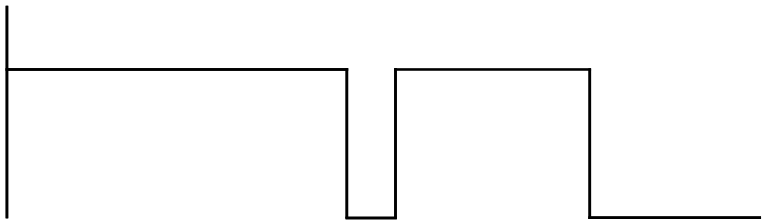

图7.20 CPU 超载产生的波形

程序7-11 (见光盘)

以下是特定人辨识的一个范例：

在程序中我们通过三条语句的训练演示特定人连续音识别，其中第一条语句为触发名称。另外两条为命令，训练完毕开始辨识当识别出触发名称后，开始发布命令，则会听到自己设置的应答，具体命令如下：

**********训练**********

提示音	输入语音
"请输入触发名称"	"警卫"
"请输入第一条命令"	"开枪"
"请输入第二条命令"	"你在干什么？"

"请再说一遍"（以上提示音每说完一遍出现此命令）

"没有听到任何声音"（当没有检测到声音时出现此命令）

"两次输入名称不相同"（当两次输入的名称不同时出现此命令）

"两次输入命令不相同"（当两次输入的命令有差异时出现此命令）

"准备就绪，请开始辨识"（以上三条语句全部训练成功时，进入识别）

**********识别**********

发布命令	应答
"警卫"	"在"/"长官"
"开枪"	"枪声"
"你在干什么？"	"我在巡逻"/"我在休息"/"我在等人"

注意：在每次提示音结束后 2-3 秒再输入命令或当上次应答结束 2-3 秒后再发布命

令

主程序流程如 0

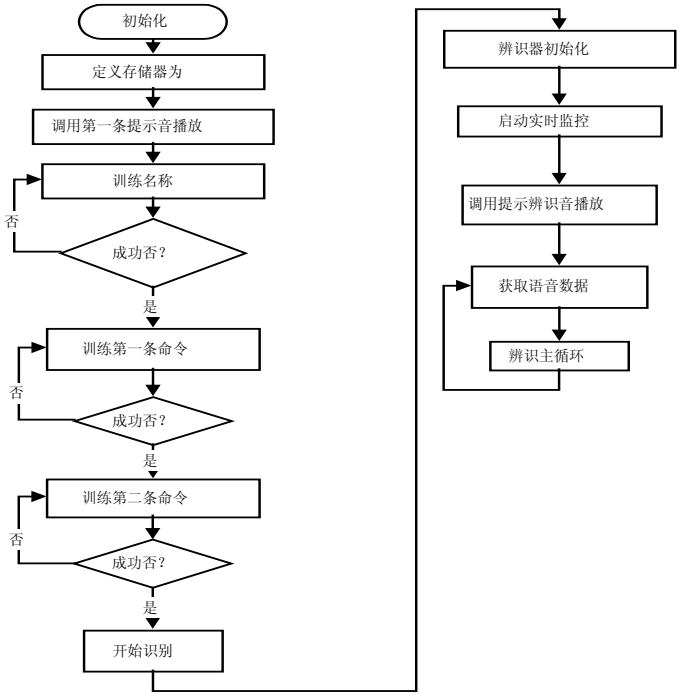

图7.21 主程序流程

```
#INCLUDE "bsrsd.h"
#DEFINE NAME_ID                 0x100
#DEFINE COMMAND_ONE_ID          0x101
#DEFINE COMMAND_TWO_ID          0x102
#DEFINE RSP_INTR                0
#DEFINE RSP_NAME                1
#DEFINE RSP_FIRE                2
#DEFINE RSP_GUARD               3
#DEFINE RSP_AGAIN               4
#DEFINE RSP_NOVOICE             5
#DEFINE RSP_NAMEDIFF            6
#DEFINE RSP_CMDDIFF             7
#DEFINE RSP_STAR                8
#DEFINE RSP_MASTER              9
#DEFINE RSP_HERE                10
#DEFINE RSP_GUNSHOT             0
#DEFINE RSP_PATROL              11
#DEFINE RSP_READY               12
```

```
#DEFINE RSP_COPY                    13
#DEFINE RSP_NOISY                   14
//.....全程变量.....
int gActivated = 0;
                    //该变量用于检测是否有触发命令，当有识别出语句为触发名称则该位置 1
int gTriggerRespond[] = {RSP_MASTER, RSP_HERE, RSP_MASTER};
                    //第一条命令应答
int gComm2Respond[] = {RSP_PATROL, RSP_READY, RSP_COPY};
                    //第二条命令应答
extern void ClearWatchDog();
int PlayFlag = 0;
void PlayRespond2(int Result)
                    //枪声放音子程序
{
    BSR_StopRecognizer();
    SACM_A2000_Initial(1);
    SACM_A2000_Play(Result, 3, 3);
    while((SACM_A2000_Status()&0x0001) != 0)
    {
        SACM_A2000_ServiceLoop();
        ClearWatchDog();
    }
    SACM_A2000_Stop();
    BSR_InitRecognizer(BSR_MIC);
    BSR_EnableCPUIndicator();
}
void PlayRespond(int Result)                    //放音子程序
{
    BSR_StopRecognizer();
    SACM_S480_Initial(1);
    SACM_S480_Play(Result, 3, 3);
    while((SACM_S480_Status()&0x0001) != 0)
    {
        SACM_S480_ServiceLoop();
        ClearWatchDog();
    }
    SACM_S480_Stop();
    BSR_InitRecognizer(BSR_MIC);
    BSR_EnableCPUIndicator();                   //启动实时监控
}
int TrainWord(int WordID, int RespondID)        //命令训练
{
    int res;
    PlayRespond(RespondID);
```

```
    while(1)
    {
        res = BSR_Train(WordID,BSR_TRAIN_TWICE);
        if(res == 0) break;
        switch(res)
        {
        case -1:                                    //没有检测出声音
            PlayRespond(RSP_NOVOICE);
            return -1;
        case -2:                                    //需要重新训练一遍
            PlayRespond(RSP_AGAIN);
            break;
        case -3:                                    //环境太吵
            PlayRespond(RSP_NOISY);
            return -1;
        case -4:                                    //数据库满
            return -1;
        case -5:                                    //检测出声音不同
        if(WordID == NAME_ID)
PlayRespond(RSP_NAMEDIFF);                          //两次输入名称不同
else
PlayRespond(RSP_CMDDIFF);//两次输入命令不同
            return -1;
        case -6:                                    //序号错误
            return -1;
        }
    }
    return 0;
}
int main()
{
    int res, timeCnt=0, random_no=0;
    BSR_DeleteSDGroup(0);                           // 初始化存储器为 RAM
    PlayRespond(RSP_INTR);                          //播放开始训练的提示音
                        //.....训练名称.....
    while(TrainWord(NAME_ID,1) != 0) ;
                        //.....训练第一条命令.....
    while(TrainWord(COMMAND_ONE_ID,2) != 0) ;
                        //.....训练第二条命令.....
    while(TrainWord(COMMAND_TWO_ID,3) != 0) ;
                        //.....开始识别命令.....
    BSR_InitRecognizer(BSR_MIC);                    //辨识器初始化
    BSR_EnableCPUIndicator();
    PlayRespond(RSP_STAR);                          // 播放开始辨识的提示音
```

```
while(1)
{
    random_no ++;
    if(random_no >= 3) random_no = 0;
    res = BSR_GetResult();
    if(res > 0)                                    //识别出命令
    {
        if(gActivated)
        {
            timeCnt = 0;
            switch(res)
            {
            case NAME_ID:
                PlayRespond(gTriggerRespond[random_no]);
                break;
            case COMMAND_ONE_ID:
                PlayFlag = 1;
                PlayRespond2(RSP_GUNSHOT);
                PlayFlag = 0;
                gActivated = 0;
                break;
            case COMMAND_TWO_ID:
                PlayRespond(gComm2Respond[random_no]);
                gActivated = 0;
            }
        }
        else
        {
            if(res == NAME_ID)
            {PlayRespond(gTriggerRespond[random_no]);
                gActivated = 1;
                timeCnt = 0;
            }
        }
    }
    else if (gActivated)
    {
        if (++timeCnt > 450)                       //超出定时
        {PlayRespond(RSP_NOVOICE);          //在设定时间内没有检测出声音
            gActivated = 0;
            timeCnt = 0;
        }
    }
}
```

}

中断流程见图 7.22：

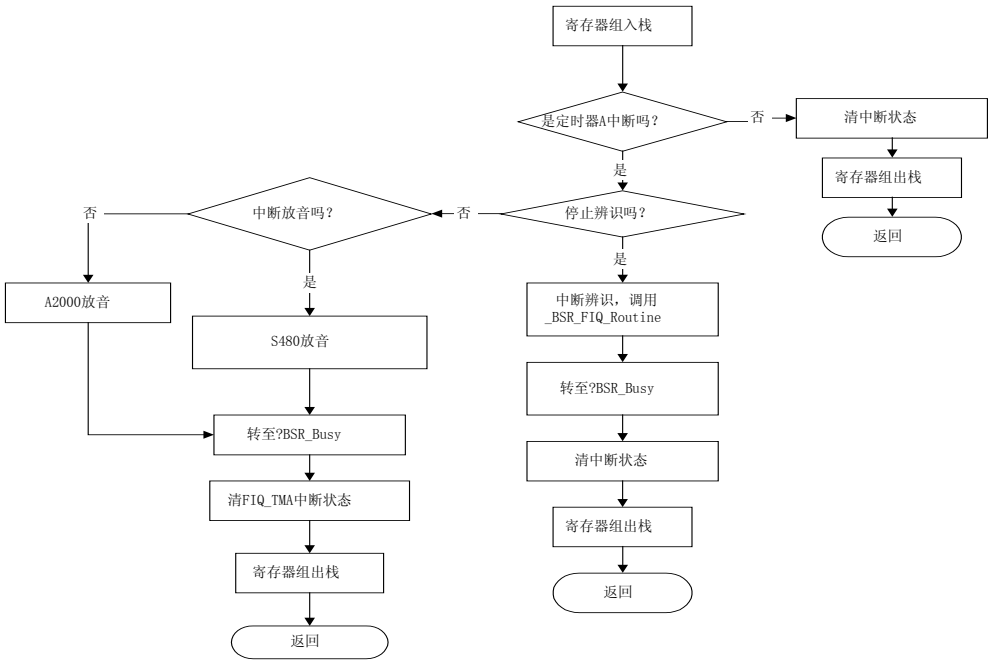

图7.22 特定人连续语音识别中断程序流程

```
.PUBLIC _FIQ
.EXTERNAL _BSR_FIQ_Routine
.EXTERNAL __gIsStopRecog                    //变量值 = 0 辨识器忙
                                            //      = 1 辨识器停止


.PUBLIC _BREAK,_IRQ0, _IRQ1, _IRQ2, _IRQ3, _IRQ4, _IRQ5, _IRQ6, _IRQ7
.EXTERNAL _PlayFlag
.INCLUDE s480.inc;
.INCLUDE A2000.inc;
.INCLUDE resource.inc
.INCLUDE hardware.inc

.TEXT
_FIQ:
    push R1,R4 to [SP]
    R1 = [P_INT_Ctrl]
    R1 &= 0x2000
    jz ?notTimerA                           //当不为 TIQ_TMA，则转
    R1 = [__gIsStopRecog]
    jnz ?BSR_NotBusy
    //[__gIsStopRecog]为 1 则转至放音处理
    call _BSR_FIQ_Routine                   //为 0，调用辨识子程序
    jmp ?BSR_Busy                           //返回中断
?BSR_NotBusy:                               //放音处理
    R2 = [_PlayFlag]
    jnz ?Play2000                           //[_PlayFlag]为 1 则是播放 2000
    call F_FIQ_Service_SACM_S480;           //为 0，播放 480
    jmp ?BSR_Busy                           //返回中断
?Play2000:                                  //2000 播放子程序
    call F_FIQ_Service_SACM_A2000;
?BSR_Busy:                                  //返回中断
    R1 = 0x2000
    [P_INT_Clear] = R1
    pop R1,R4 from [SP];
    reti;
?notTimerA:
    R1 = 0x8800;
   [P_INT_Clear] = R1;
    pop R1,R4 from [SP];
    reti;
.END
```

7.7 本章小结

本章主要向大家介绍了表 7.3 中的各种压缩算法的 API（**Application Programming Interface**）函数功能及各自的应用，并进一步介绍了两个接口文件 Key.asm 和 System.asm 的具体内容及应用方法。还举出了使用 SPCE061A 进行辨识的一个应用实例。

表7.3 凌阳语音压缩算法及其编码

凌阳压缩算法	编码方法	编码率
SACM_A2000	SUB-BAND	16/20/24Kbps
SACM_S480	CELP	4.8/7.2Kbps
SACM_S240	LPC	1.2/2.4Kbps
SACM_MS01	FM	音乐合成
SACM_DVR（A2000 压缩）	SUB-BAND	16K 资料率/8K 采样率录音

1）通过本章介绍大家都了解到自动方式放音包括 SACM_A2000、SACM_S480 和 SACM_S240 三种方式，而且也发现每种算法除了压缩比不同外程序结构基本相同，所以非常容易掌握并应用，在这里我们巩固一下前面的内容，图 7.23 是自动方式下整个程序的流程：

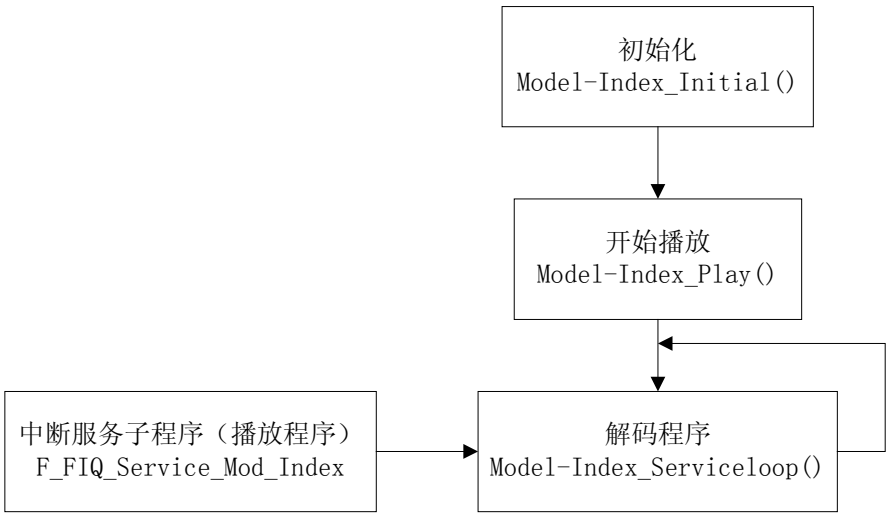

图7.23 自动方式下程序的流程

2）另外 SACM_A2000 和 SACM_DVR 除了自动方式外还有非自动方式，图 7.24 是在手动方式下声音录制与播放流程（当然 SACM_A2000 只有声音播放部分）：

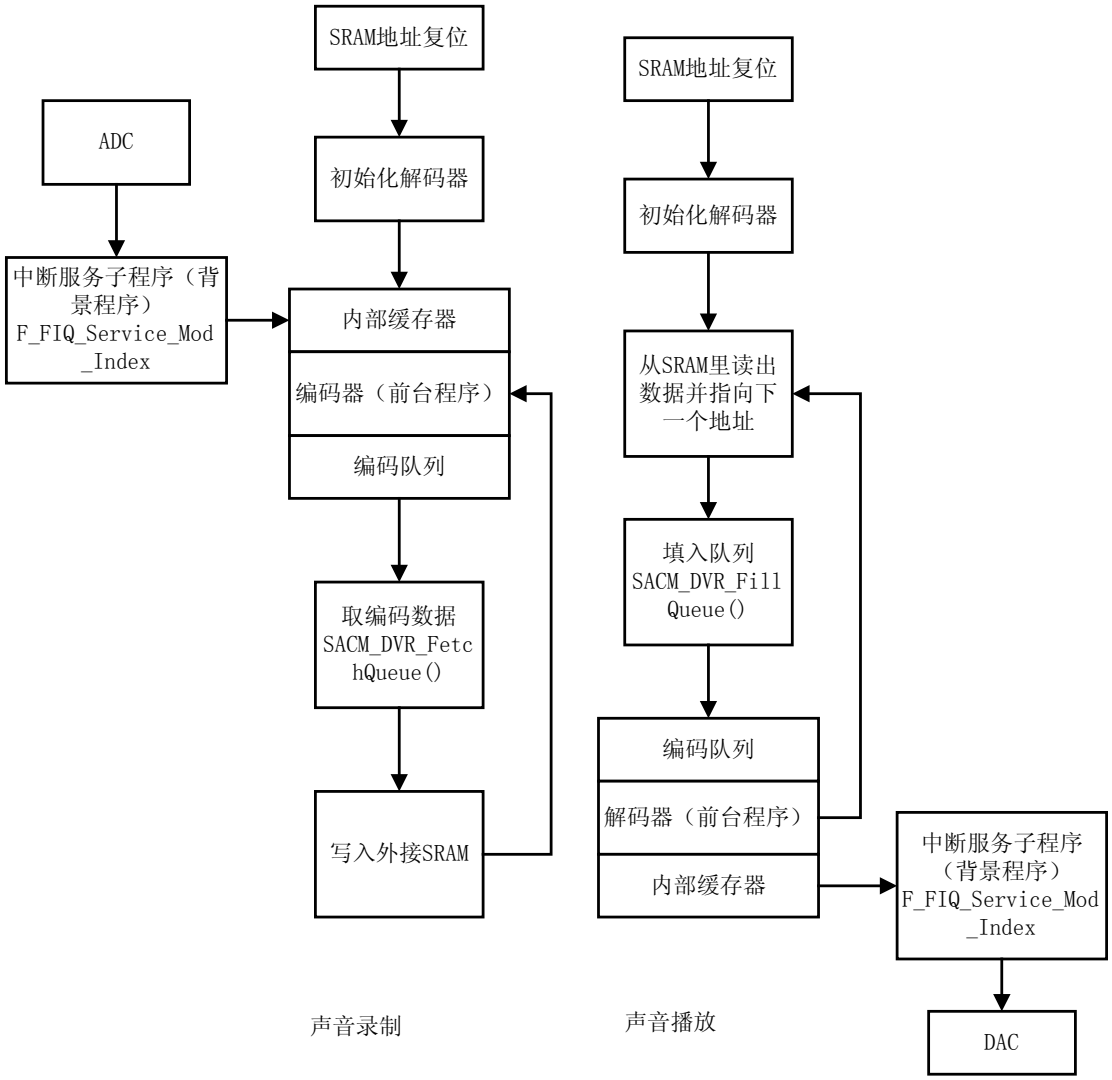

图7.24 手动方式下声音录制与播放流程

至于键控部分，如果同学们感兴趣的话可以结合 7.5 小节内容自己尝试做一些键控放音的实验以进一步更好的掌握语音编程。

备注：

有时候我们在一个较复杂的程序里，可能不只用到语音，还有 A/D、中断、定时等，这样，在放音过程中很有可能会和程序中的一些寄存器发生冲突，所以了解 API 函数中占用了哪些寄存器就很有必要的，列出了各个函数所占用的寄存器。

表7.4 API 函数中所占用的寄存器

函数	使用寄存器
SACM_A2000_Initial(int Init_Index) SACM_S480_Initial(int Init_Index) SACM_S240_Initial(int Init_Index) SACM_MS01_Initial(int Init_Index) SACM_DVR_Initial(int Init_Index)	[P-_SystemClock] [P_TimerA_Ctrl] [P_TimerA_Data] [P_DAC_Ctrl] [P_INT_Clear] [P_TimerB_Ctrl] [P_TimerB_Data]
SACM_ A2000_Play() SACM_A2000_InitDecoder()[Manual Mode] SACM_ S480_Play() SACM_ S240_Play() SACM_ MS01_Play() SACM_ DVR_Play() SACM_ DVR_InitDecoder()[Manual Mode]	[P_INT_Clear] [P_TimerA_Data]
SACM_ DVR_Record() SACM_ DVR_InitDecoder()	[P_ADC_Ctrl] [P_TimerA_Data] [P_INT_Ctrl]
SACM_ A2000_Stop() SACM_ A2000_Stop Decoder() SACM_ S480_Stop() SACM_ S240_Stop() SACM_ MS01_Stop() SACM_ DVR_Stop() SACM_ Stop Decoder() SACM_ StopEncoder()	[P_INT_Ctrl] [P_INT_Clear] [P_ADC_Ctrl]
SACM_ A2000_ServiceLoop() SACM_ A2000_ Decoder()[Manual Mode] SACM_ S480_ServiceLoop() SACM_ S240_ServiceLoop() SACM_ MS01_ServiceLoop() SACM_ DVR_ServiceLoop() SACM_ DVR _ Encoder()[Manual Mode]	[P_INT_Ctrl] [P_INT_Clear]
F_FIQ_Service_ SACM_A2000 F_FIQ_Service_ SACM_S480 F_FIQ_Service_ SACM_S240 F_FIQ_Service_ SACM_MS01 F_FIQ_Service_ SACM_DVR	[P_DAC1] [P_DAC2]
F_IRQ1_Service_ SACM_DVR	[P_ADC]