第 9 章 凌阳音频的 C 语言程序设计 93
9.1 压缩分类和压缩算法 93
9.1.1 凌阳音频压缩算法的编码标准 93
9.1.2 压缩分类 93
9.1.3 凌阳常用的音频形式和压缩算法 93
9.2 常用的应用程序接口 API 的功能介绍及应用 94
9.2.1 概述 94
9.2.2 SACM_A2000 95
9.2.3 SACM_S480 101
9.2.4 SACM_S240 104
9.2.5 SACM_MS01 109
9.2.6 SACM_DVR 115
9.3 语音辨识 118
9.4 本章 API 函数中所占用的寄存器 127

第9章 凌阳音频的 C 语言程序设计

9.1 压缩分类和压缩算法

9.1.1 凌阳音频压缩算法的编码标准

表 9.1 是不同音频质量等级的编码技术标准（频响）：

表9.1 编码技术标准

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。

9.1.2 压缩分类

凌阳音频压缩算法根据不同的压缩比分为以下几种：

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

9.1.3 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。

9.2 常用的应用程序接口 API 的功能介绍及应用

9.2.1 概述

表 9.2 所列出的是凌阳音频的几种算法：

表9.2 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 信道录音功能	16KHz

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 9.1 所示。凌阳提供了 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，你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。SACM-LIB 目前主要有 sacmv25.lib、sacmv26e.lib、sacmv32.lib，这三个库在下面的例程都有用到，请读者注意使用这三个库有什么不同。再次提醒，本书的所有例程都可以到 http://www.unsp.com.cn 下载。

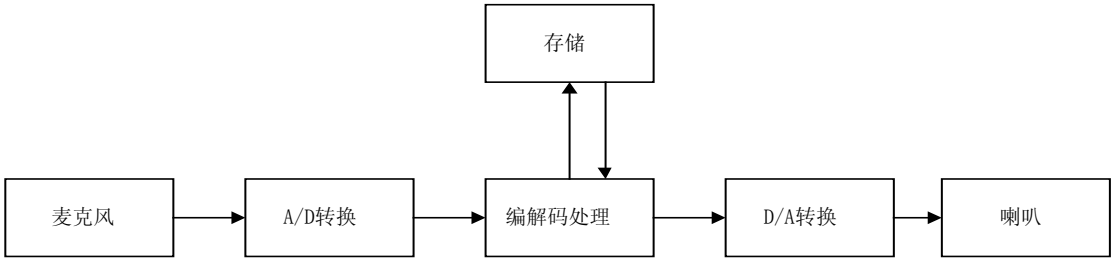

图9.1 单片机对语音处理过程

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。

9.2.2 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 F_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，从 0 开始计数。

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。

函数允许 TimerA 以所选的的数据采样率（计数溢出）中断。

例9-1 以自动方式播放一段 SACM_A2000 语音，并自动结束。

SACM_A2000 自动方式主程序流程图：

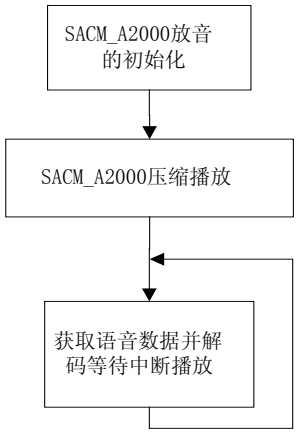

图9.2 A2000 自动方式主程序流程

主程序：

```
#define        DAC1              1
#define        DAC2              2
#define        Ramp_UpDn_On      3
#include   "SPCE061V004.H"
```

```
main ()
{
 SACM_A2000_Initial(1);
 SACM_A2000_Play(0, DAC1+DAC2, Ramp_UpDn_On);          //放音
 while(SACM_A2000_Status()&0x01)
 {
      SACM_A2000_ServiceLoop();
      *P_Watchdog_Clear=C_WDTCLR;
 }
 while(1)                                              //死循环
      *P_Watchdog_Clear=C_WDTCLR;
}
```

中服务断函数

```
#include      "SPCE061V004.H"
__asm(".external      F_FIQ_Service_SACM_A2000");
void FIQ(void) __attribute__ ((ISR));
void FIQ(void)
{
     if(*P_INT_Ctrl&0x2000)                           //定时器 A 中断
     {
          *P_INT_Clear=C_FIQ_TMA;
          __asm("call F_FIQ_Service_SACM_A2000");   // A2000 中断服务函数
     }
     else if(*P_INT_Ctrl&0x0800)                      //定时器 B 中断
          *P_INT_Clear=C_FIQ_TMB;
     else                                             //PWM 中断
          *P_INT_Clear=C_FIQ_PWM;
}
```

注：播放语音文件中数据，当出现 FF FF FFH 数据时便停止播放。
本例要链接语音库 sacmv25.lib，以及和该库对应的 hardware.asm。

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 语音或乐曲的播放。
【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。如果要在 C 语言中使用，可以使用嵌入式汇编的办法：__asm("call F_FIQ_Service_SACM_A2000")。

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 信道以及 DAC1 和 DAC2 双通道。
【返 回 值】无。
【备　　注】只能用于非自动方式

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 信道播放。

【参　　数】无。

【返 回 值】无。

【备　　注】只能用于非自动方式

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】只能用于非自动方式

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；R1=0 表示语音队列不空不满，R1=1 表示语音队列满，R1=2 表示语音队列空。

【备　　注】只能用于非自动方式

例9-2 SACM_A2000 非自动方式（编程控制）播放语音

SACM_A2000 非自动方式主程序流程见图 9.3：

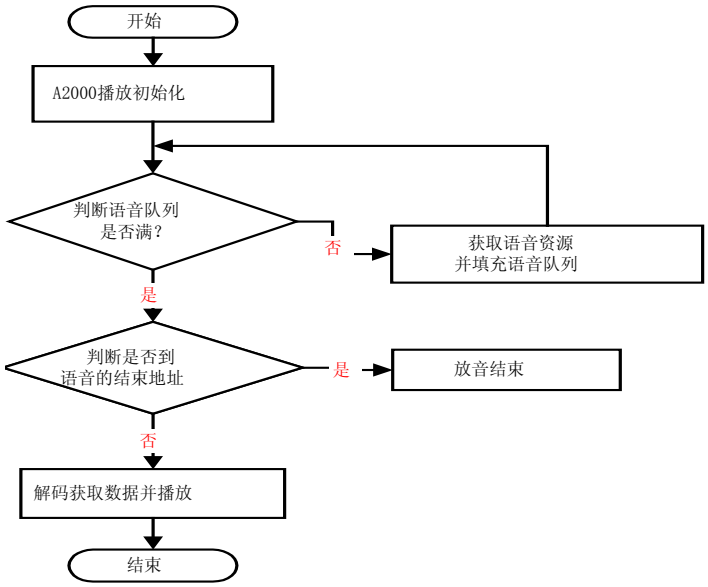

图9.3 SACM_A2000 非自动方式主程序流程

中断服务子程序流程见图 9.4：

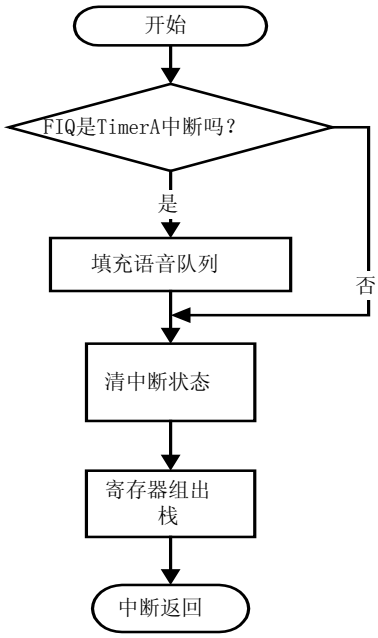

图9.4 SACM_A2000 中断服务子程序流程

本例的中断服务程序和上例完全相同，不再列出。主程序如下：

```
#define          DAC1                1
#define          Full                1
#include   "A2000.h"
#define P_Watchdog_Clear          (volatile unsigned int *)0x7012
main()
{
  extern long     RES_D1_24K_SA,RES_D1_24K_EA;      //语音资源的首末地址标号
  long    int Addr;                                 //定义地址变量
  int Ret = 0;

 Addr=RES_D1_24K_SA;                                //送入语音队列的首址

 SACM_A2000_Initial(0);                             //非自动方式播放的初始化

 SACM_A2000_InitDecoder(DAC1);                      //以非自动方式解码

while(1)
{
      *P_Watchdog_Clear=0x0001;
```

```
        if(SACM_A2000_TestQueue()!=Full)
        {
             Ret =SP_GetResource(Addr);
            SACM_A2000_FillQueue(Ret);
            Addr++;
        }

        if(Addr< RES_D1_24K_EA )                    //如果该段语音未播完
            SACM_A2000_Decoder();                   //获取资源并进行解码，再通过中
                                                    //断服务程序送入 DAC 通道播放
        else
            SACM_A2000_Stop();                      //否则，停止播放
       }
    }
```

注：

1）文件的结束是由用户位址变量控制的。

2）本例要链接语音库 sacmv25.lib，以及和该库对应的 hardware.asm。

9.2.3 SACM_S480

该压缩算法压缩比较大，为 80:3,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：

```
void 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：viod　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】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_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】

① SACM_S480 的数据率有 4.8Kbps\7.2Kbps 两种可选。

②Speech_Index 是定义在 resource.asm 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。

④函数允许 TimerA 以所选的的数据采样率（计数溢出）中断。

例9-3 以自动方式播放一段 SACM_S480 语音，并自动结束

SACMV25.lib 中的 SACM_S480 只有自动播放方式,在 SACMV32.lib 中的 SACM_S480 既有自动播放方式，又有手动播放方式。本例采用 SACMV25.lib，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。

主程序如下：

```
#define     DAC1            1
#define     DAC2            2
#define     Ramp_UpDn_On    3
#define     Auto        1
```

```
#define P_Watchdog_Clear          (volatile unsigned int *)0x7012
#include  "s480.h"
 main()
{
      int SpeechIndex=0;

      SACM_S480_Initial(Auto);
      SACM_S480_Play(SpeechIndex,DAC1+DAC2, Ramp_UpDn_On);

      while(1)
      {
            SACM_S480_ServiceLoop();
            *P_Watchdog_Clear=0x0001;
      }
}
```

中断服务程序如下：

```
#include  "SPCE061V004.H"
__asm(".external    F_FIQ_Service_SACM_S480");
void FIQ(void) __attribute__ ((ISR));
void FIQ(void)
{
      if(*P_INT_Ctrl&0x2000)                          //定时器 A 中断
      {
            *P_INT_Clear=C_FIQ_TMA;
            __asm("call F_FIQ_Service_SACM_S480"); //调用 A2000 中断服务函数
      }
      else if(*P_INT_Ctrl&0x0800)                     //定时器 B 中断
            *P_INT_Clear=C_FIQ_TMB;
      else                                            //PWM 中断
            *P_INT_Clear=C_FIQ_PWM;
}
```

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。本例要链接语音库 sacmv25.lib，以及和该库对应的 hardware.asm。

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 语音或乐曲。
【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 语音或乐曲。

【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 语音或乐曲的播放。

【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_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。

9.2.4 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】

【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：void 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】

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。

④ 函数允许 TimerA 以所选的的数据采样率（计数溢出）中断。

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 语音或乐曲。
【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_ 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 信道播放。
【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。

例9-4 以自动方式播放一段 SACM_ S240 语音，并自动结束。

注意，本例使用的是 sacmv32.lib，所以需要加入和 sacmv32.lib 对应的 sacmv32.asm/sp_lib.asm/system.asm。

把 A 口低 8 位接 1×8 键盘。各个按键的意思如下：

K0：放音；
K1：停止放音；
K2：暂停放音；
K3：继续放音；
K4：音量增加；
K5：音量减小；
K6：后一段；
K7：前一段。

SACM_S240 在自动方式下的主程序如下：

```
#include "sp_lib.h"
#include "sacmv32.h"
#define MaxSpeechNum     3                  // 最大语音资源数目
#define MaxVolume        15                 // 声音的最大值
int   Ret = 0;
extern long RES_PK1_SA;
long Addr;
int main(){
    int Key = 0;
    int SpeechIndex = 0;
    int VolumeIndex = 7;                                  // 中等音量
    int Mode;

    Mode = Auto;
    if(Mode == Auto) {
        Ret = SACM_S240_Initial(Auto);
        SACM_S240_Play(SpeechIndex,DAC1+DAC2, Ramp_UpDn_On); // 放音
        while(1){
            Key = SP_GetCh();
            switch(Key){
            case 0x00:
                break;
            case 0x01:
                SACM_S240_Play(SpeechIndex,DAC1+DAC2, Ramp_UpDn_On);
            break;
            case 0x02:
```

```
                SACM_S240_Stop();                   // 停止放音
                break;
            case 0x04:
                SACM_S240_Pause();                  // 暂停放音
                break;
            case 0x08:
                SACM_S240_Resume();                 // 继续放音
                break;
            case 0x10:
                VolumeIndex++;
                if(VolumeIndex > MaxVolume)
                    VolumeIndex = MaxVolume;
                SACM_S240_Volume(VolumeIndex);// 音量增加
                break;
            case 0x20:
                if(VolumeIndex == 0)
                    VolumeIndex = 0;
                else
                    VolumeIndex--;
                SACM_S240_Volume(VolumeIndex); // 音量减小
                break;
            case 0x40:                              // 下一段
                SpeechIndex++;
                if(SpeechIndex == MaxSpeechNum)
                    SpeechIndex = 0;
                SACM_S240_Play(SpeechIndex,DAC1+DAC2, Ramp_UpDn_On);
                break;
            case 0x80:                              // 前一段
                if(SpeechIndex == 0)
                    SpeechIndex = MaxSpeechNum;
                SpeechIndex--;
                SACM_S240_Play(SpeechIndex,DAC1+DAC2, Ramp_UpDn_On);
            break;
            default:
                break;
            }
            System_ServiceLoop();                   // 键盘扫描函数
            SACM_S240_ServiceLoop();                // SACM S240 服务函数
        }
    }
    return 0;
}
```

中断服务程序如下：

中断服务程序如下：

```
#include  "SPCE061V004.H"
__asm(".external    F_FIQ_Service_SACM_S240");
void FIQ(void) __attribute__ ((ISR));
void FIQ(void)
{
    if(*P_INT_Ctrl&0x2000)                          //定时器 A 中断
    {
        *P_INT_Clear=C_FIQ_TMA;
        __asm("call F_FIQ_Service_SACM_S240"); //调用 A2000 中断服务函数
    }
    else if(*P_INT_Ctrl&0x0800)                     //定时器 B 中断
        *P_INT_Clear=C_FIQ_TMB;
    else                                            //PWM 中断
        *P_INT_Clear=C_FIQ_PWM;
}
```

9.2.5 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)                    //接通信道
void SACM_ MS01_ChannelOff(int Channel)                   //关闭信道
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：int 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：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】

【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.asm 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。

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】恢复播放暂停的 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 9.5。

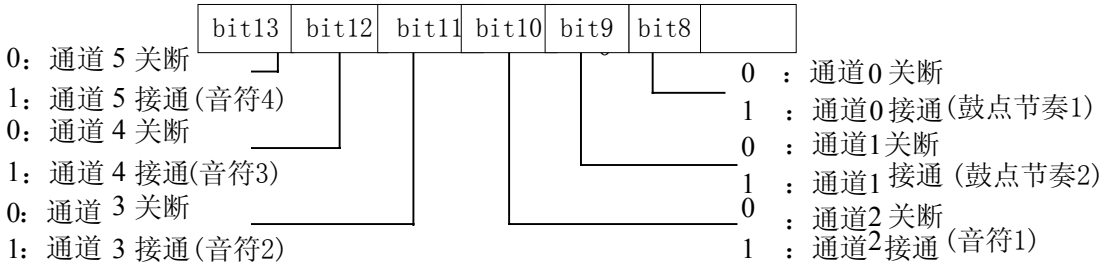

图9.5 SACM_MS01 部分状态返回值

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 代表鼓点节奏信道，2~5 则代表音符通道。

【返 回 值】无。

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 代表鼓点节奏信道，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。

例9-5　采用 SACM_MS01 自动方式播放一段乐曲

本例使用的是 sacmv32.lib，所以需要加入和 sacmv32.lib 对应的 sacmv32.asm /sp_lib.asm/system.asm。

把 A 口低 8 位接 1×8 键盘。各个按键的意思如下：

K0：放音；
K1：停止放音；
K2：暂停放音；
K3：继续放音；
K4：音量增加；
K5：音量减小；
K6：后一段；
K7：前一段。

主程序如下：

```
#include "sp_lib.h"
#include "sacmv32.h"
#define MaxSongNum      12          //乐曲资源数目
#define MaxVolume       15          //最大音量
#define Auto            1
int Mode;
int Key = 0;
int SongIndex = 0;
int VolumeIndex = 8;
int main()
{
```

```
Mode = Auto;
while(Mode == Auto)
{
    System_Initial();
    SACM_MS01_Initial(DAC_16K,Auto);                    // 打开所有通道
    SACM_MS01_Play(SongIndex,DAC1+DAC2,Ramp_Up_On+Ramp_Dn_On);
    while(1)
    {
        Key = SP_GetCh();
        switch(Key) {
        case 0x00:
            break;
        case 0x01:
    SACM_MS01_Play(SongIndex,DAC1+DAC2,Ramp_Up_On+Ramp_Dn_On);
        break;
        case 0x02:
            SACM_MS01_Stop();                           // 停止
            break;
        case 0x04:
            SACM_MS01_Pause();                          // 暂停
            break;
        case 0x08:
            SACM_MS01_Resume();                         // 继续
            break;
        case 0x10:
            VolumeIndex++;
            if(VolumeIndex > MaxVolume)
                VolumeIndex = MaxVolume;
            SACM_MS01_Volume(VolumeIndex);              // 音量增加
            break;
        case 0x20:
            if(VolumeIndex == 0)
                VolumeIndex = 0;
            else
                VolumeIndex--;
            SACM_MS01_Volume(VolumeIndex);              // 音量减小
            break;
        case 0x40:                                      // 下一曲
            if( ++SongIndex == MaxSongNum)
                SongIndex = 0;
SACM_MS01_Play(SongIndex,DAC1+DAC2,Ramp_Up_On+Ramp_Dn_On);
            break;
```

```
                case 0x80:                                          //上一曲
                    if( --SongIndex < 0)
                        SongIndex = MaxSongNum-1;
            SACM_MS01_Play(SongIndex,DAC1+DAC2,Ramp_Up_On+Ramp_Dn_On);
                    break;
                default:
                    break;
                }
                System_ServiceLoop();                               // 键盘扫描
                SACM_MS01_ServiceLoop();                            // MS01 服务函数
            }
        }
        return 0;
}
```

9.2.6 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 压缩后存储起来。

其相关 API 函数如下所示：

```
void  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  _InitDecode(void)                          //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)                //从资源文件里取一个字型语音资料
```

中断服务函数：

```
Call  F_FIQ_Service_SACM_DVR                    //播放
Call  F_IRQ1_Service_SACM_DVR                   //录制
```

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。
【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 信道以及 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

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。

这里没有给出 DVR 的例程，但本书的第十章有一个实现简易录音笔的例子，DVR 在其中占了很大的篇幅。读者可以通过该实例来了解 DVR 的使用。

9.3 语音辨识

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 9.6 是语音辨识的一个整体框图：

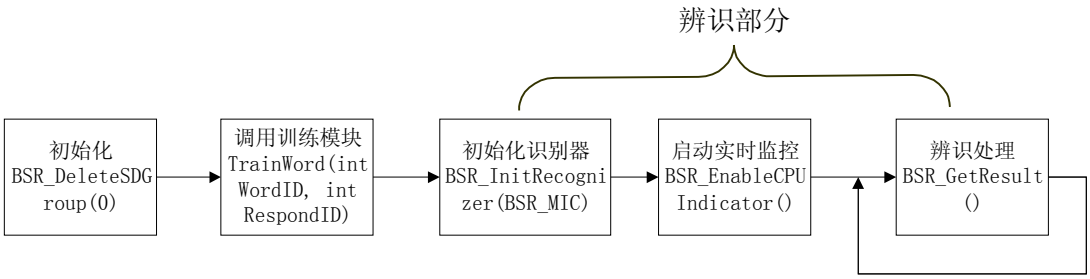

图9.6 语音辨识原理框图

下面为大家介绍一些 SACM 库中的常用语音识别 API 函数，有兴趣者可以登陆大学计划网站 http://www.unsp.com.cn 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 0 x FFF,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
```

例9-6　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）

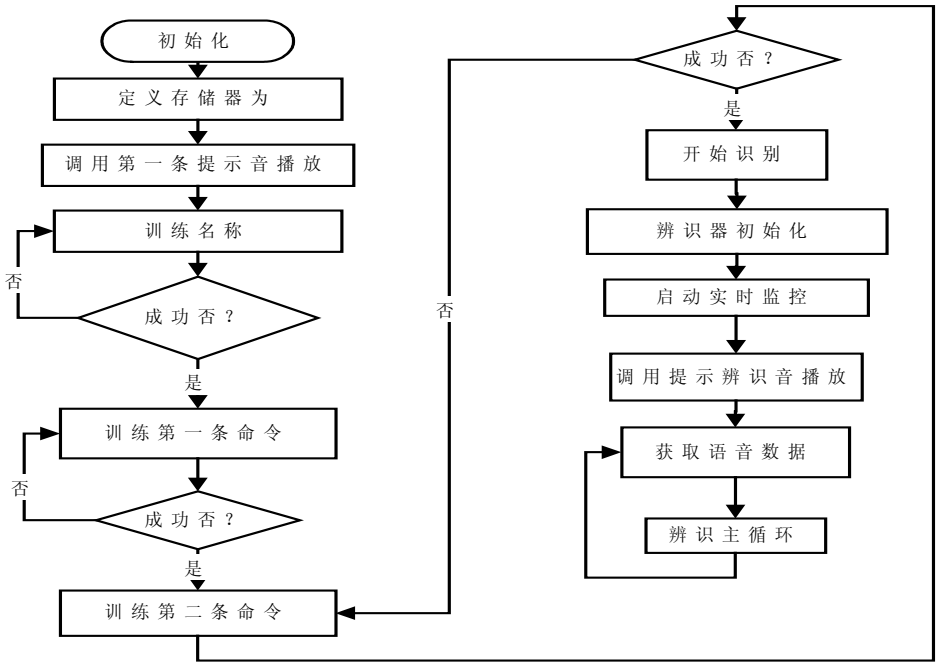

图9.7 主程序流程

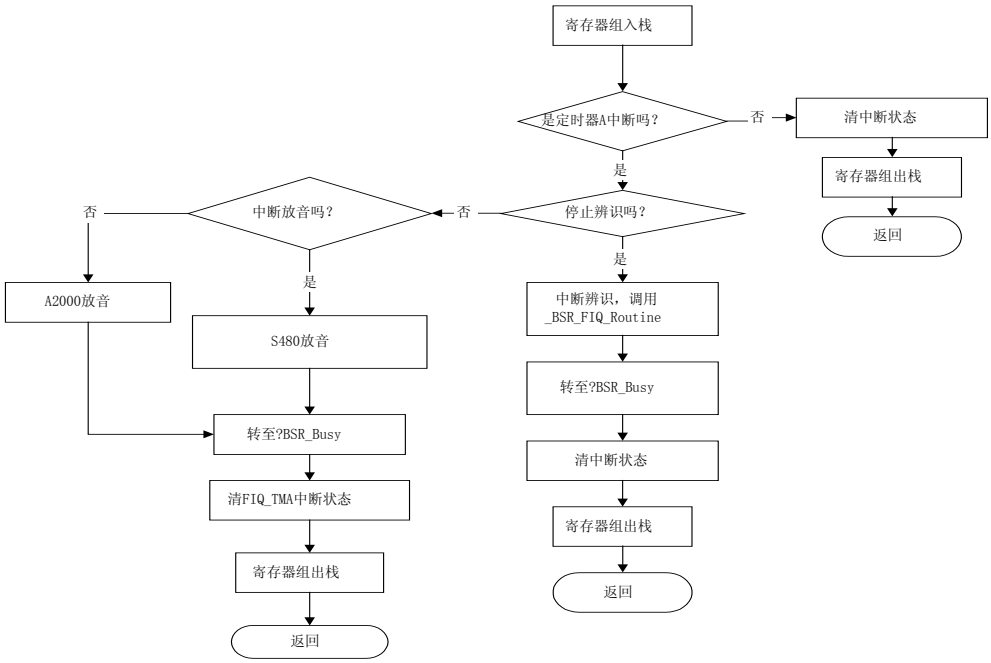

图9.8 特定人连续语音识别中断程序流程

**********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"

本例使用了 sacmv26e.lib 和 bsrv222SDL.lib 这两个库。

主程序如下：

```
#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
int gActivated = 0;      //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
int PlayFlag = 0;
extern void ClearWatchDog();

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);                       //辨识器初始化
    BSR_EnableCPUIndicator();                          //启动实时监控
}

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);                       //辨识器初始化
    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 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();                       //启动实时监控

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;
            }
        }
    }
}
```

以下是中断服务程序，用汇编实现的：

```
.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 A2000
    call F_FIQ_Service_SACM_S480;           //为 0，播放 S480
    jmp ?BSR_Busy                           //返回中断
?Play2000:                                  //A2000 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
```

9.4 本章 API 函数中所占用的寄存器

本章主要向大家介绍了凌阳语音库中各种压缩算法的 API 函数功能及各自的应用。还举出了使用 SPCE061A 进行辨识的一个应用实例。

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，下表列出了各个函数所占用的寄存器。

表9.3 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()	[P_INT_Ctrl] [P_INT_Clear]

SACM_ S480_Stop() SACM_ S240_Stop() SACM_ MS01_Stop() SACM_ DVR_Stop() SACM_ Stop Decoder() SACM_ StopEncoder()	[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]