

# SACMv41dx 语音函数库

---

## 使用说明书

V1.0 – 2006.11.18

凌阳科技大学计划教育推广中心  
北京海淀上地信息产业基地中黎科技园 1 号楼 3 层

TEL: 010-62981113    FAX: 010-62986660-2992    E-mail: [unsp@sunplus.com.cn](mailto:unsp@sunplus.com.cn)    <http://www.unsp.com>

## 版权声明

凌阳科技股份有限公司保留对此文件修改之权利且不另行通知。凌阳科技股份有限公司所提供之信息相信为正确且可靠之信息，但并不保证本文件中绝无错误。请于向凌阳科技股份有限公司提出订单前，自行确定所使用之相关技术文件及规格为最新之版本。若因贵公司使用本公司之文件或产品，而涉及第三人之专利或著作权等智能财产权之应用及配合时，则应由贵公司负责取得同意及授权，本公司仅单纯贩售产品，上述关于同意及授权，非属本公司应为保证之责任。又未经凌阳科技股份有限公司之正式书面许可，本公司之所有产品不得使用于医疗器材，维持生命系统及飞航等相关设备。

# 目 录

1	SACMv41dx语音函数库简介 .....	1
1.1	概述 .....	1
1.2	SACM_A1600 简介 .....	1
1.3	SACM_DVR1600 简介 .....	1
1.4	SACM_S720 简介 .....	1
1.5	SACM_MS01 简介 .....	1
1.6	算法性能对比 .....	2
2	程序架构 .....	3
2.1	语音播放程序架构 .....	3
2.2	语音录制程序架构 .....	5
3	库函数说明 .....	7
3.1	语音播放相关函数 .....	7
3.1.1	SACMv41dx_061A.lib中的播放相关函数 .....	7
3.1.2	SACM_xxxx.asm中的播放相关函数 .....	11
3.1.3	SACM_xxxx_User_C.c与SACM_xxxx_User_ASM.asm中的播放相关函数 .....	13
3.2	语音录制相关函数 .....	14
3.2.1	SACMv41dx_061A.lib中的录音相关函数 .....	14
3.2.2	SACM_DVR1600.asm中的录音相关函数 .....	16
3.2.3	SACM_DVR1600_User_C.c与SACM_DVR1600_User_ASM.asm中的录音相关函数 .....	17
4	语音压缩工具SunAudior .....	19
4.1	SunAudior简介 .....	19
4.2	使用SunAudior软件进行语音压缩 .....	20
5	应用举例 .....	27
5.1	语音播放 .....	27
5.2	语音录制（DVR1600） .....	35
6	常见问题 .....	40

## 1 SACMv41dx 语音函数库简介

### 1.1 概述

SACMv41dx 语音函数库是凌阳科技大学计划针对 SPCE061A 单片机推出的语音录制和播放解决方案。它包括 SACM\_A1600、SACM\_DVR1600、SACM\_S720 和 SACM\_MS01 四套语音编解码软件包，适合语音播放、语音录制、音乐播放等用途。

### 1.2 SACM\_A1600 简介

SACM\_A1600 可对由 A1600 算法压缩的语音资源进行解码播放。A1600 算法具有 10Kbps、12Kbps、14Kbps、16Kbps、20Kbps 和 24Kbps 六种可选的码率，可将“8KHz, 16 bit”的音频以 12.8:1 ~ 5.3:1 的压缩比进行编码。

A1600 算法具有复杂度较低，保真度较高的特点，适合于各种音频（歌曲、环境音效、高品质语音等）播放。

对于 SACM\_A1600，仅可用来播放已压缩好的语音资源，而不具备实时压缩编码（录音）的功能。

随 SACMv41dx 语音函数库一同提供的 SunAudior 软件可以将 wav 格式的声音文件压缩为 SACM\_A1600 可以播放的语音资源，参见第4章。

### 1.3 SACM\_DVR1600 简介

SACM\_DVR1600 与 SACM\_A1600 类似，都采用 A1600 算法。所不同的是，SACM\_DVR1600 除可播放外，还可以在单片机上进行实时的压缩编码（录音）；但要比 SACM\_A1600 占用更多的 RAM 和 ROM 空间。

### 1.4 SACM\_S720 简介

SACM\_S720 可对由 S720 算法压缩的语音资源进行解码播放。S720 算法具有 4.8Kbps 和 7.2Kbps 两种可选的码率，可将“8KHz, 16 bit”的音频以 26.7:1 或 17.8:1 的压缩比进行编码。

S720 算法的复杂度较高，可实现较大比例的压缩，以节省存储空间；但音质损失比较大，仅适合处理语音，可用于对音质要求不太高的语音提示等场合。

对于 SACM\_S720，仅可以用来播放已压缩好的语音资源，而不具备实时压缩编码（录音）的功能。可利用 SunAudior 软件将 wav 格式的声音文件压缩为 SACM\_S720 可以播放的语音资源。

### 1.5 SACM\_MS01 简介

SACM\_MS01 可用来播放 MS01 格式编码的合成音乐。MS01 是专用于音乐合成的一种算法，可将特定格式的乐谱信息转换为音乐。SACM\_MS01 最大支持同时播放 4 通道的合成音乐和 2 通道的鼓点声音。

MS01 音乐资源的制作方法比较复杂：首先需准备好符合要求的 MIDI 文件（可从网上下载或利用 CakeWalk 等软件制作），然后利用 SACMv41dx 语音库附带的 Midi2Pop 软件将 MIDI 转换为扩展名为 .pop 的乐谱文件，最后利用 Pop2Bin 转换工具转换为 SACM\_MS01 可播放的音乐资源。

MS01 转换工具的使用方法参见 Midi2Pop 和 Pop2Bin 各自的使用说明，本文档仅介绍播放 MS01 音乐的方法。

## 1.6 算法性能对比

SACMv41dx 函数库中包含的这几种算法的性能对比列表如下。

表 1.1 各算法性能对比

	A1600	DVR1600	S720	MS01
音质	好	好	较差	合成乐音
适用范围	语音/音频播放	语音/音频录放	语音播放	音乐播放
采样率	8000 Hz	8000 Hz	8000 Hz	16000 Hz
码率	10~24Kbps	10~24Kbps	4.8/7.2Kbps	--
压缩比	12.8 ~ 5.3	12.8 ~ 5.3	26.7 / 17.8	--
SRAM 占用	451 Words	485 Words	451 Words	463 Words
Flash-ROM 占用	3.9K Words	4.3K Words	2.7K Words	5.0K Words
CPU 占用 (@24MHz)	约 60%	录音约 90% 放音约 60%	约 60%	约 80%
CPU 占用 (@49MHz)	约 30%	录音约 45% 放音约 30%	约 30%	约 40%

## 2 程序架构

### 2.1 语音播程序架构

单片机进行语音播放的原理如图 2.1所示：在播放循环中，程序不断对语音资源进行解码，并填入播放队列，而将解码后的数据由DAC播放出来的操作是在Timer中断服务程序中完成的。

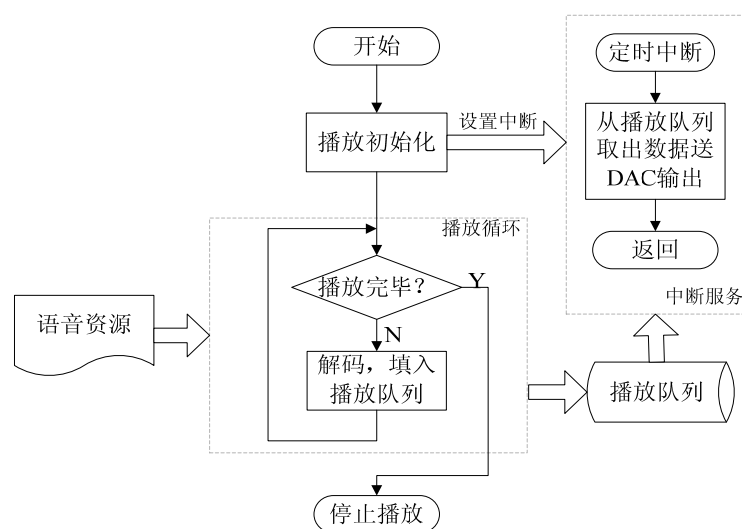





图 2.1 SACMv41dx 语音播放原理

采用 SACMv41dx 库的语音播放程序就是基于上述原理构建出来的。SACM\_A1600、SACM\_DVR1600、SACM\_S720 和 SACM\_MS01 用于播放的程序架构基本相同，都包括以下几个文件（文件名中的“xxxx”表示 A1600、DVR1600、S720 或 MS01）：

表 2.1 语音播放程序文件列表

文件名	用途
SACMv41dx_061A.lib	SACMv41dx 函数库核心文件。用户通过调用该库提供的函数即可实现语音播放。
SACM_xxxx.asm	用户接口函数集，用于对语音播放的初始化、DAC 输出、播放结束时的操作等进行配置。在一般应用中无需修改该文件，采用默认的配置即可。这些接口函数是被 SACMv41dx 库函数自动调用的。
SACM_xxxx_User_C.c 或 SACM_xxxx_User_ASM.asm	用于访问语音资源的函数集，用户可以根据需要对该文件进行修改。SACMv41dx 语音函数库为该函数集提供了 C 语言和汇编语言两套程序模板，可选用其中之一。
xxxx.h	头文件，库函数声明，可被 C 语音编写的程序包含。
xxxx.inc	头文件，库函数声明，可被汇编语言编写的程序包含。

语音播放程序的工作过程如图 2.2所示，图中标示为  的函数是SACMv41dx\_061A.lib中的库函数，

标示为  的是 SACM\_xxxx.asm 文件中定义的用户接口函数，标示为  的是在 SACM\_xxxx\_User\_C.c或SACM\_xxxx\_User\_ASM.asm文件中定义的资源访问函数；粗箭头表示函数的调用关系，单线箭头表示程序工作流程。图中涉及的函数在3.1节有详细的说明。

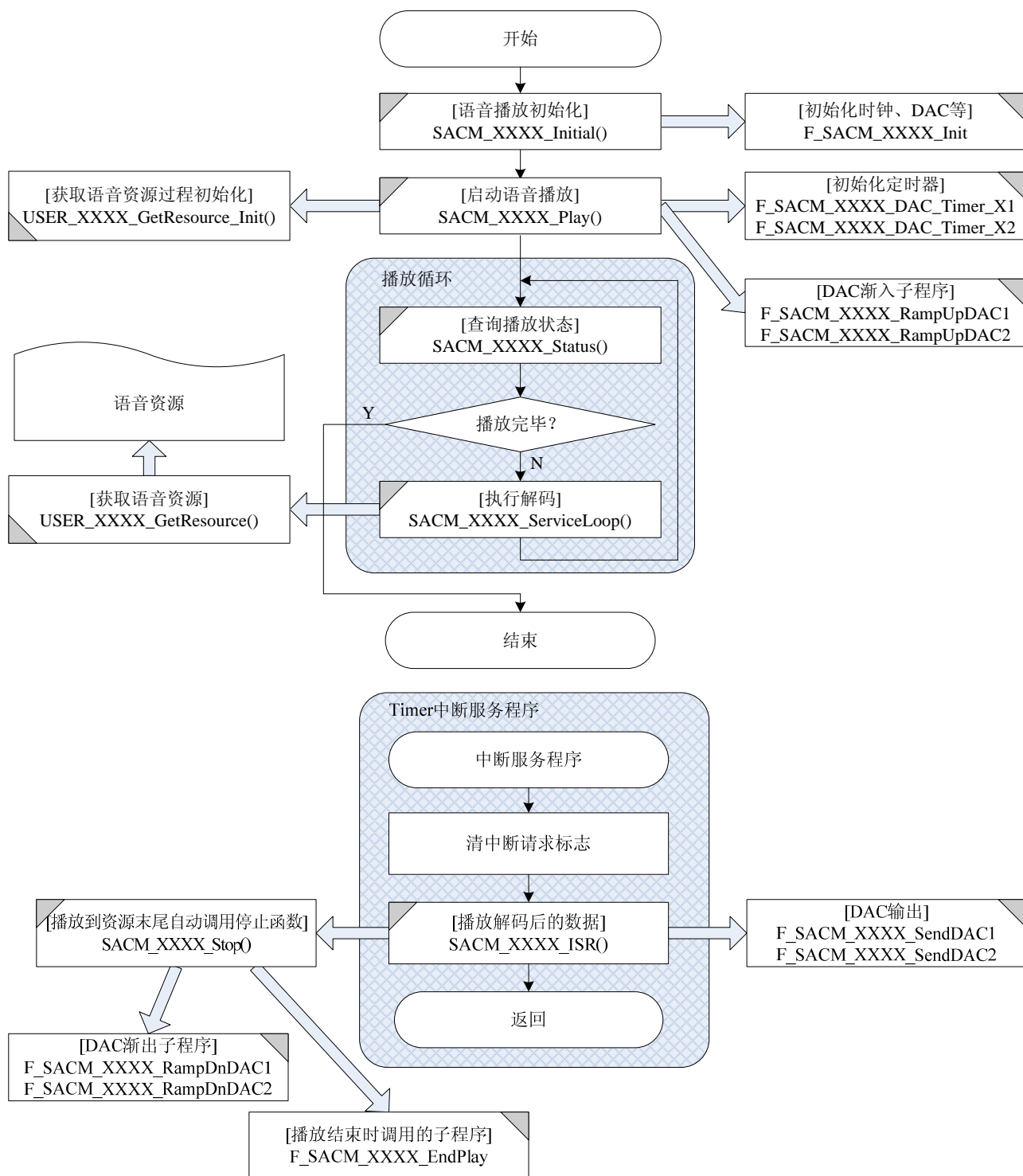


图 2.2 语音播放程序工作示意图

## 2.2 语音录制程序架构

在SACMv41dx语音函数库中，只有SACM\_DVR1600 具有实时压缩编码功能，可用于语音录制。单片机进行语音录制的原理如图 2.3所示：录音初始化时会开启Timer中断，在中断服务程序中定时地进行A/D采样；在录音循环中对采集到的语音样本进行压缩编码，生成语音资源。

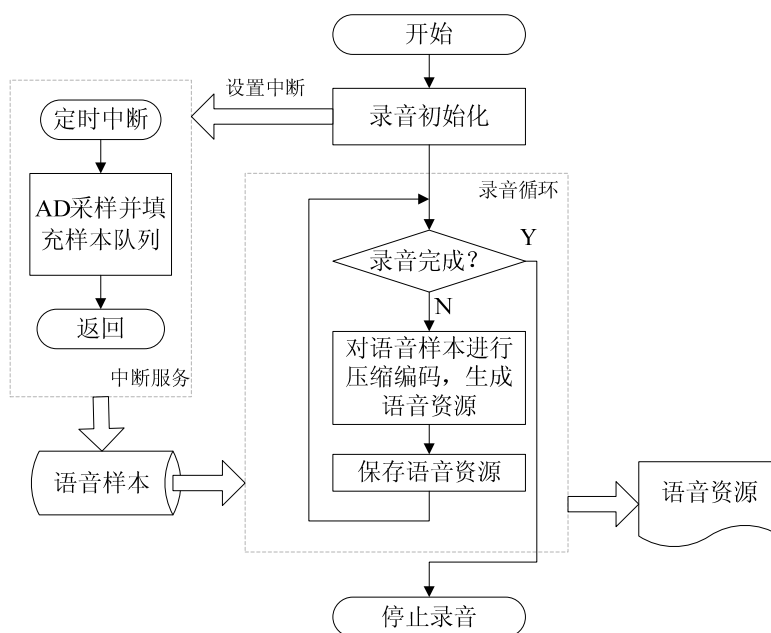





图 2.3 SACMv41dx 语音录制原理

SACM\_DVR1600 录音程序同样使用表 2.1列出的文件，其工作过程如图 2.4所示，图中标示为  的函数是SACMv41dx\_061A.lib中的库函数，标示为  的是SACM\_DVR1600.asm文件中定义的用户接口函数，标示为  的是在SACM\_DVR1600\_User\_C.c或SACM\_DVR1600\_User\_ASM.asm文件中定义的资源访问函数；粗箭头表示函数的调用关系，单线箭头表示程序工作流程。图中涉及的函数在3.2节有详细的说明。



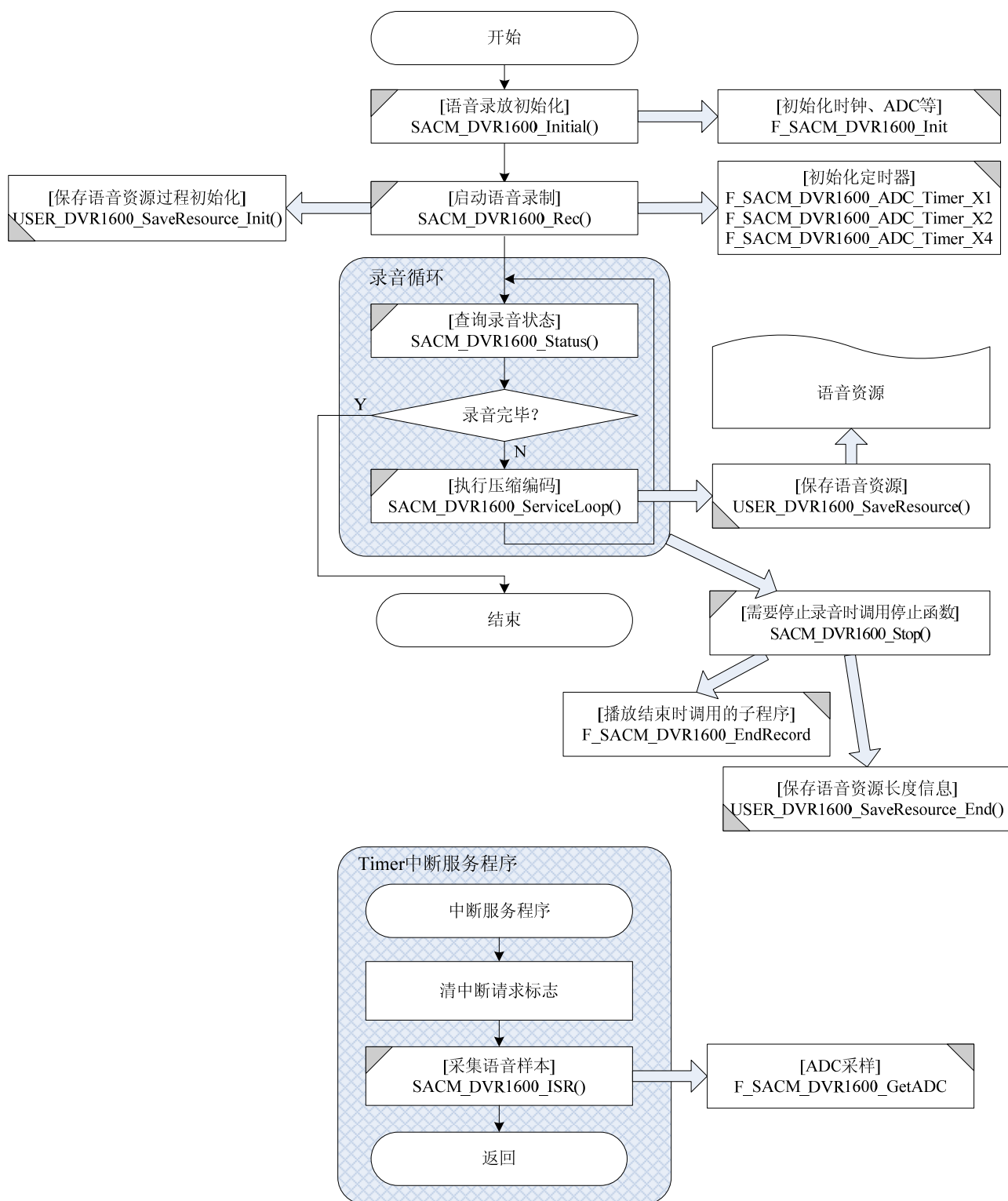


图 2.4 语音录制程序工作示意图

## 3 库函数说明

### 3.1 语音播放相关函数

SACM\_A1600、SACM\_DVR1600、SACM\_S720 和 SACM\_MS01 的语音播放函数形式基本相同，下面的函数名称中的“xxxx”表示 A1600、DVR1600、S720 或 MS01。

#### 3.1.1 SACMv41dx\_061A.lib中的播放相关函数

##### 【SACM\_xxxx\_Initial】

[C 格式]: void SACM\_xxxx\_Initial(void);

[汇编格式]: F\_SACM\_xxxx\_Initial

[实现功能]: 语音算法初始化。在使用 A1600、DVR1600、S720 和 MS01 算法之前要执行一次该函数。

[参数]: 无

[返回值]: 无

[说明]: 执行该函数时，将自动调用 SACM\_xxxx.asm 文件中定义的 F\_SACM\_xxxx\_Init 用户接口函数，用于初始化时钟、中断、DAC、ADC 等。对于 DVR1600 用于录制语音，也要在开始录音之前执行该函数，进行初始化。

##### 【SACM\_xxxx\_Play】

[C 格式]: void SACM\_xxxx\_Play(unsigned int SoundIndex, unsigned int DAC\_Channel, unsigned int RampUpDn);

[汇编格式]: F\_SACM\_xxxx\_Play

[实现功能]: 开始语音播放。

[参数]: SoundIndex (R1): 待播放的语音资源序号

DAC\_Channel (R2): 语音输出的 DAC 通道

1 - DAC1

2 - DAC2

3 - DAC1+DAC2

RampUpDn (R3): 是否在播放起始和结束时采取渐入渐出方式

0 - 不采取渐入渐出方式

1 - 播放起始时采取渐入方式

2 - 播放结束时采取渐出方式

3 - 播放起始时采取渐入方式，播放结束时采取渐出方式

[返回值]: 无

[说明]: 当开始播放一段语音时, 要执行 SACM\_xxxx\_Play 函数。执行该函数将发生下列动作:

1. 自动调用 SACM\_xxxx\_User\_C.c (或汇编语言版的 SACM\_xxxx\_User\_ASM.asm, 下同) 文件中定义的 USER\_xxxx\_GetResource\_Init 函数, 并把 SoundIndex 参数传递过去。
2. 根据 DAC\_FIRType (参见本节中 SACM\_xxxx\_DAC\_FIRType 函数的说明) 自动调用 SACM\_xxxx.asm 中定义的 SACM\_xxxx\_DAC\_Timer\_X1 或 SACM\_xxxx\_DAC\_Timer\_X2 函数, 以初始化 Timer。
3. 如果 RampUpDn 参数的值为 1 或 3, 将根据 DAC\_Channel 参数自动调用 SACM\_xxxx.asm 文件中定义的 F\_SACM\_xxxx\_RampUpDAC1、F\_SACM\_xxxx\_RampUpDAC2 函数。RampUp 的含义是: 在初始状态下, DAC 的默认数字量为 0x0000, 而大多数语音是以数字量 0x8000 作为输出基准的; 当开始播放语音时, DAC 输出由 0x0000 突然变为 0x8000, 将产生轻微的爆音, 影响播放效果。为解决这个问题, 可以选择 RampUp, 即在开始播放语音之前, 逐渐地将 DAC 输出数字量由 0x0000 升高至 0x8000, 以消除播放开始时的爆音。

#### 【SACM\_xxxx\_ServiceLoop】

[C 格式]: void SACM\_xxxx\_ServiceLoop(void);

[汇编格式]: F\_SACM\_xxxx\_ServiceLoop

[实现功能]: 执行解码; 对于 DVR1600 录音程序, 则执行压缩编码。

[参数]: 无

[返回值]: 无

[说明]: SACM\_xxxx\_ServiceLoop 是在播放循环中执行的函数, 用来进行解码操作 (对于 DVR1600 录音程序则进行编码操作)。当执行该函数时, 如果当前不需要解码 (例如并没有开始播放语音, 或者前次解码得到的数据还没有被播放完), 则直接从函数体中跳出; 如果需要执行解码, 则会自动调用 SACM\_xxxx\_User\_C.c 文件中定义的 USER\_xxxx\_GetResource 函数, 获取解码所需的语音资源。而对于 DVR1600 录音程序, 当不需要压缩编码时 (例如并没有开始录音, 或者 AD 还没有采集够一帧语音样本), 则直接跳出; 如果需要进行压缩编码, 则执行编码过程, 并在完成编码之后自动调用 SACM\_DVR1600\_User\_C.c 中的 USER\_DVR1600\_SaveResource 函数保存编码后的数据 (即语音资源)。

#### 【SACM\_xxxx\_Stop】

[C 格式]: void SACM\_xxxx\_Stop(void);

[汇编格式]: F\_SACM\_xxxx\_Stop

[实现功能]: 停止语音播放; 对于 DVR1600 录音程序, 则停止语音录制。

[参数]: 无

[返回值]: 无

[说明]: 当某段语音播放到末尾后, 将自动调用 SACM\_xxxx\_Stop 函数结束语音播放。也可在播放过程中利用该函数中止当前语音。对于 DVR1600 录音程序, 则使用 SACM\_DVR1600\_Stop 函数来

停止录音。SACM\_xxxx\_Stop 函数执行时将发生以下动作：

1. 如果 SACM\_xxxx\_Play 函数的 RampUpDn 参数值为 2 或 3，则将根据参数 DAC\_Channel 自动调用 SACM\_xxxx.asm 文件中定义的 F\_SACM\_xxxx\_RampDnDAC1、F\_SACM\_xxxx\_RampDnDAC2 函数。RampDown 的作用与 RampUp 类似，用于防止播放结束时 DAC 输出值突然变为 0 导致的爆音。
2. 自动调用 SACM\_xxxx.asm 文件中定义的 F\_SACM\_xxxx\_EndPlay 用户接口函数。用户可以在 F\_SACM\_xxxx\_EndPlay 函数中自定义播放结束时执行的动作。而对于 DVR1600 录音程序，则自动调用 F\_SACM\_DVR1600\_EndRecord 函数，用户可以在该函数中自定义录音结束时执行的动作。
3. （仅针对 DVR1600 录音程序）自动调用 SACM\_DVR1600\_User\_C.c 文件中的 USER\_DVR1600\_SaveResource\_End 函数，用来保存录音编码资源的长度信息。

#### 【SACM\_xxxx\_Pause】

[C 格式]: void SACM\_xxxx\_Pause(void);

[汇编格式]: F\_SACM\_xxxx\_Pause

[实现功能]: 暂停语音播放；对于 DVR1600 录音程序，则暂停语音录制。

[参数]: 无

[返回值]: 无

[说明]: 在语音播放或语音录制过程中，可以执行 SACM\_xxxx\_Pause 函数暂停播放或录制。当需要继续放音或录音时，可执行 SACM\_xxxx\_Resume 函数来恢复。

#### 【SACM\_xxxx\_Resume】

[C 格式]: void SACM\_xxxx\_Resume(void);

[汇编格式]: F\_SACM\_xxxx\_Resume

[实现功能]: 恢复被 SACM\_xxxx\_Pause 函数暂停的语音播放或录制过程。

[参数]: 无

[返回值]: 无

[说明]: 在语音播放或语音录制过程中，可以执行 SACM\_xxxx\_Pause 函数暂停播放或录制。当需要继续放音或录音时，可执行 SACM\_xxxx\_Resume 函数来恢复。

#### 【SACM\_xxxx\_Volume】

[C 格式]: void SACM\_xxxx\_Volume(unsigned int Volume);

[汇编格式]: F\_SACM\_xxxx\_Volume

[实现功能]: 设定语音播放的音量。

[参数]: Volume (R1): 音量等级，范围 0~15，默认为 8。

[返回值]: 无

[说明]: 该函数用于设定语音播放的音量, 当参数 Volume 为 0 则无声音输出, 为 15 则以最大音量播放语音。

#### 【SACM\_xxxx\_DAC\_FIRType】

[C 格式]: void SACM\_xxxx\_DAC\_FIRType(unsigned int FIRType);

[汇编格式]: F\_SACM\_xxxx\_DAC\_FIRType

[实现功能]: 设定语音播放的 DAC 输出滤波方式。

[参数]: FIRType (R1): 输出滤波方式代码, 范围 0~3。

0 - 不进行滤波, 解码后以 8KHz 采样率直接输出

1 - 不进行滤波, 以二倍采样率(16KHz)插值输出

2 - 以二倍采样率(16KHz)进行简易滤波输出

3 - 以二倍采样率(16KHz)进行复杂滤波输出。

[返回值]: 无

[说明]: 在语音播放时, 语音资源解码后得到的数据应以 8000Hz 的速率由 DAC 输出。但为了得到更好的播放效果, SACMv41dx 库提供了数字滤波输出功能。默认状态下, DAC 输出滤波方式代码为 2, 即以二倍采样率(16KHz)进行简易滤波输出。可利用 SACM\_xxxx\_DAC\_FIRType 函数对 DAC 滤波方式进行设定, FIRType 代码越大, 则播放音质越好, 但 CPU 占用率也会相应提高。

#### 【SACM\_xxxx\_ISR】

[C 格式]: void SACM\_xxxx\_ISR(void);

[汇编格式]: F\_SACM\_xxxx\_ISR

[实现功能]: 语音播放 (或录制) 中断服务, 应在 Timer 中断服务程序中执行该函数。

[参数]: 无

[返回值]: 无

[说明]: 该函数应放在 Timer 中断服务程序中执行, 将解码得到的待输出数据由 DAC 输出。如果当前没有进行语音播放, 则执行该函数后将不进行任何动作直接退出; 而在语音播放时, 该函数会自动调用 SACM\_xxxx.asm 文件中定义的 F\_SACM\_xxxx\_SendDAC1、F\_SACM\_xxxx\_SendDAC2 函数进行语音输出。对于 DVR1600 录音程序, 在录音时, 该函数时将会自动调用 SACM\_DVR1600.asm 中的 F\_SACM\_xxxx\_GetADC 函数, 获得 A/D 采样结果。

#### 【SACM\_xxxx\_Status】

[C 格式]: unsigned int SACM\_xxxx\_Status(void);

[汇编格式]: F\_SACM\_xxxx\_Status

[实现功能]: 该函数用来查询当前语音播放 (或录制) 的状态。

[参数]: 无

[返回值]: (R1) 录放音状态编码。

bit0: 1-当前处于录/放音状态; 0-当前不处于录/放音状态

bit1: 1-录音状态; 0-放音状态

bit2: 1-录放音暂停状态; 0-正常状态

bit3: 1-播放时可通过 DAC1 输出; 0-播放时不通过 DAC1 输出

bit4: 1-播放时可通过 DAC2 输出; 0-播放时不通过 DAC2 输出

bit5: 1-播放起始时采取渐入方式; 0-播放起始时不采取渐入方式

bit6: 1-播放结束时采取渐出方式; 0-播放结束时不采取渐出方式

bit7~bit14: 保留

bit15: 放音时, 如果当前处于输出缓冲欠载状态(不能及时进行解码)则该位为 1;  
录音时, 如果当前处于输入缓冲溢出状态(不能及时进行压缩编码)则该位为 1。

[说明]: 无

### 3.1.2 SACM\_xxxx.asm中的播放相关函数

#### 【F\_SACM\_xxxx\_Init】

[汇编格式]: F\_SACM\_xxxx\_Init

[实现功能]: 被库函数 SACM\_xxxx\_Initial 自动调用, 用于初始化时钟、中断、ADC、DAC 等。

[参数]: 无

[返回值]: 无

[说明]: 无

#### 【F\_SACM\_xxxx\_DAC\_Timer\_X1】

[汇编格式]: F\_SACM\_xxxx\_DAC\_Timer\_X1

[实现功能]: 当 DAC\_FIRType 选定为 0 时, 该函数被库函数 SACM\_xxxx\_Play 自动调用, 用于初始化 Timer。

[参数]: 无

[返回值]: 无

[说明]: 在一般应用中, 对于 A1600、DVR1600、S720 语音播放, 该函数应将 Timer 溢出频率设置为 8000Hz; 而对于 MS01, 应将 Timer 溢出频率设置为 16000Hz。

#### 【F\_SACM\_xxxx\_DAC\_Timer\_X2】

[汇编格式]: F\_SACM\_xxxx\_DAC\_Timer\_X2

[实现功能]: 当 DAC\_FIRType 选定为 1~3 时, 该函数被库函数 SACM\_xxxx\_Play 自动调用, 用于初始化 Timer。

[参数]: 无

[返回值]: 无

[说明]: 在一般应用中, 对于 A1600、DVR1600、S720 语音播放, 该函数应将 Timer 溢出频率设置为 16000Hz; 而对于 MS01, 应将 Timer 溢出频率设置为 32000Hz。

#### 【F\_SACM\_xxxx\_SendDAC1】

[汇编格式]: F\_SACM\_xxxx\_SendDAC1

[实现功能]: 在语音播放时, 如果 SACM\_xxxx\_Play 的 DAC\_Channel 参数为 1 或 3, 则该函数被 SACM\_xxxx\_ISR 函数自动调用, 将解码后得到的数字量由 DAC1 输出。

[参数]: R1: 待通过 DAC1 输出的数字量

[返回值]: 无

[说明]: 无

#### 【F\_SACM\_xxxx\_SendDAC2】

[汇编格式]: F\_SACM\_xxxx\_SendDAC2

[实现功能]: 在语音播放时, 如果 SACM\_xxxx\_Play 的 DAC\_Channel 参数为 2 或 3, 则该函数被 SACM\_xxxx\_ISR 函数自动调用, 将解码后得到的数字量由 DAC2 输出。

[参数]: R1: 待通过 DAC2 输出的数字量

[返回值]: 无

[说明]: 无

#### 【F\_SACM\_xxxx\_EndPlay】

[汇编格式]: F\_SACM\_xxxx\_EndPlay

[实现功能]: 当一段语音播放完毕时, 该函数将被 SACM\_xxxx\_Stop 函数自动调用, 用来执行自定义程序。

[参数]: 无

[返回值]: 无

[说明]: 无

#### 【F\_SACM\_xxxx\_RampUpDAC1】

[汇编格式]: F\_SACM\_xxxx\_RampUpDAC1

[实现功能]: 在语音播放开始时, 如果 SACM\_xxxx\_Play 的 DAC\_Channel 参数为 1 或 3, 并且 RampUpDn 参数为 1 或 3, 则该函数将被 SACM\_xxxx\_Play 函数自动调用, 使 DAC1 产生渐入过程, 以防止播放开始时出现爆音。

[参数]: 无

[返回值]: 无

[说明]: 关于RampUp的作用参见3.1.1节关于SACM\_xxxx\_Play函数的说明。

#### 【F\_SACM\_xxxx\_RampUpDAC2】

[汇编格式]: F\_SACM\_xxxx\_RampUpDAC2



**[实现功能]:** 在语音播放开始时, 如果 SACM\_xxxx\_Play 的 DAC\_Channel 参数为 2 或 3, 并且 RampUpDn 参数为 1 或 3, 则该函数将被 SACM\_xxxx\_Play 函数自动调用, 使 DAC2 产生渐入过程, 以防止播放开始时出现爆音。

**[参数]:** 无

**[返回值]:** 无

**[说明]:** 关于RampUp的作用参见3.1.1节关于SACM\_xxxx\_Play函数的说明。

#### 【F\_SACM\_xxxx\_RampDnDAC1】

**[汇编格式]:** F\_SACM\_xxxx\_RampDnDAC1

**[实现功能]:** 在语音播放开始时, 如果 SACM\_xxxx\_Play 的 DAC\_Channel 参数为 1 或 3, 并且 RampUpDn 参数为 2 或 3, 则该函数将被 SACM\_xxxx\_Stop 函数自动调用, 使 DAC1 产生渐出过程, 以防止播放结束时出现爆音。

**[参数]:** 无

**[返回值]:** 无

**[说明]:** 关于RampDown的作用参见3.1.1节关于SACM\_xxxx\_Stop函数的说明。

#### 【F\_SACM\_xxxx\_RampDnDAC2】

**[汇编格式]:** F\_SACM\_xxxx\_RampDnDAC2

**[实现功能]:** 在语音播放开始时, 如果 SACM\_xxxx\_Play 的 DAC\_Channel 参数为 2 或 3, 并且 RampUpDn 参数为 2 或 3, 则该函数将被 SACM\_xxxx\_Stop 函数自动调用, 使 DAC2 产生渐出过程, 以防止播放结束时出现爆音。

**[参数]:** 无

**[返回值]:** 无

**[说明]:** 关于RampDown的作用参见3.1.1节关于SACM\_xxxx\_Stop函数的说明。

### 3.1.3 SACM\_xxxx\_User\_C.c与SACM\_xxxx\_User\_ASM.asm中的播放相关函数

#### 【USER\_xxxx\_GetResource\_Init】

**[C 格式]:** void USER\_xxxx\_GetResource\_Init(unsigned int SoundIndex);

**[汇编格式]:** \_USER\_xxxx\_GetResource\_Init

**[实现功能]:** 在语音播放开始时, 该函数将被 SACM\_xxxx\_Play 函数自动调用, 可利用该函数初始化语音资源的起始地址。

**[参数]:** SoundIndex (R1): 由 SACM\_xxxx\_Play 函数传递过来的待播放语音序号。

**[返回值]:** 无

**[说明]:** 用户可以自行编写该函数体中的内容, 根据 SoundIndex 来确定待播放语音资源的起始地址。

#### 【USER\_xxxx\_GetResource】

**[C 格式]:** void USER\_xxxx\_GetResource(unsigned int \*p\_Buf, unsigned int Words);



[汇编格式]: \_USER\_xxxx\_GetResource

[实现功能]: 在 SACM\_xxxx\_ServiceLoop 执行解码时, 将会自动调用该函数, 获取解码所需的语音资源数据。

[参数]: p\_Buf (R1): 语音资源的目标存放地址, 它是由 SACM\_xxxx\_ServiceLoop 函数自动传递过来的参数。用户需要将语音资源数据填入以 p\_Buf 为起始地址的缓冲区中。

Words (R2): 需要填入 p\_Buf 缓冲区中的数据长度, 单位为 Word。该参数也是由 SACM\_xxxx\_ServiceLoop 函数自动传递过来的。

[返回值]: 无

[说明]: 用户可以自行编写该函数体中的内容, 从存储语音资源的介质中提取数据, 填入以 p\_Buf 为起始地址的 RAM 中。应注意的是, 每次执行该函数时, 向 p\_Buf 中填入数据的数量必须与 Words 参数所要求的数量一致, 否则将导致播放出错。

## 3.2 语音录制相关函数

仅 SACM\_DVR1600 具有语音录制功能。在录音程序中, 使用到的大多数函数是与放音程序相同的, 这里将不作详细说明。

### 3.2.1 SACMv41dx\_061A.lib中的录音相关函数

#### 【SACM\_DVR1600\_Initial】

参见3.1.1节关于该函数的说明。

#### 【SACM\_DVR1600\_Rec】

[C 格式]: void SACM\_DVR1600\_Rec(unsigned int UserParam, unsigned int BitRate);

[汇编格式]: F\_SACM\_DVR1600\_Rec

[实现功能]: 开始 DVR1600 录音。

[参数]: UserParam (R1): 用户自定义参数。

BitRate (R2): 语音压缩编码的码率

0 - 10Kbit/s

1 - 12Kbit/s

2 - 14Kbit/s

3 - 16Kbit/s

4 - 20Kbit/s

5 - 24Kbit/s

[返回值]: 无

[说明]: 执行 SACM\_DVR1600\_Rec 函数将开始录制一段语音。该函数将引发以下动作:

1. 自动调用 SACM\_DVR1600\_User\_C.c 文件中定义的 USER\_DVR1600\_SaveResource\_Init

函数，并把 UserParam 参数传递过去。UserParam 的内容由用户自行定义，例如在进行多段录音时，可以利用该参数传递当前录音的段序号。如果不需要这个参数，可以赋给它任意整数值。

2. 根据 ADC\_FIRType（参见本节中关于 SACM\_DVR1600\_ADC\_FIRType 函数的说明）自动调用 SACM\_DVR1600.asm 文件中定义的 SACM\_DVR1600\_ADC\_Timer\_X1、SACM\_DVR1600\_ADC\_Timer\_X2 或 SACM\_DVR1600\_ADC\_Timer\_X4 函数，以初始化 Timer。

#### 【SACM\_DVR1600\_ServiceLoop】

参见3.1.1节关于该函数的说明。

#### 【SACM\_DVR1600\_Stop】

参见3.1.1节关于该函数的说明。

#### 【SACM\_DVR1600\_Pause】

参见3.1.1节关于该函数的说明。

#### 【SACM\_DVR1600\_Resume】

参见3.1.1节关于该函数的说明。

#### 【SACM\_DVR1600\_ADC\_FIRType】

[C 格式]: void SACM\_DVR1600\_ADC\_FIRType(unsigned int FIRType);

[汇编格式]: F\_SACM\_DVR1600\_ADC\_FIRType

[实现功能]: 设定 A/D 采样时的滤波方式。

[参数]: FIRType (R1): 采样滤波方式代码，范围 0~2。

0 - 不进行滤波，以 8KHz 采样后直接编码

1 - 以二倍采样率(16KHz)采样滤波后再进行编码

2 - 以四倍采样率(32KHz)采样滤波后再进行编码

[返回值]: 无

[说明]: 在语音录制时，一般应以 8000Hz 的速率对语音进行 A/D 采样。但为了得到更好的录音效果，SACMv41dx 库提供了数字滤波功能。默认状态下，采样滤波方式代码为 1，即以二倍采样率(16KHz)采样滤波后再进行编码。可利用 SACM\_DVR1600\_ADC\_FIRType 函数对采样滤波方式进行设定，FIRType 代码越大，则录音质量越好，但 CPU 占用率也会相应提高。

#### 【SACM\_DVR1600\_ISR】

参见3.1.1节关于该函数的说明。

#### 【SACM\_DVR1600\_Status】

参见3.1.1节关于该函数的说明。

### 3.2.2 SACM\_DVR1600.asm中的录音相关函数

#### 【F\_SACM\_DVR1600\_Init】

参见3.1.2节关于该函数的说明。

#### 【F\_SACM\_DVR1600\_ADC\_Timer\_X1】

[汇编格式]: F\_SACM\_DVR1600\_ADC\_Timer\_X1

[实现功能]: 当 ADC\_FIRType 选定为 0 时, 该函数被 SACM\_DVR1600\_Rec 函数自动调用, 用于初始化 Timer。

[参数]: 无

[返回值]: 无

[说明]: 在一般应用中, 该函数应将 Timer 溢出频率设置为 8000Hz。

#### 【F\_SACM\_DVR1600\_ADC\_Timer\_X2】

[汇编格式]: F\_SACM\_DVR1600\_ADC\_Timer\_X2

[实现功能]: 当 ADC\_FIRType 选定为 1 时, 该函数被 SACM\_DVR1600\_Rec 函数自动调用, 用于初始化 Timer。

[参数]: 无

[返回值]: 无

[说明]: 在一般应用中, 该函数应将 Timer 溢出频率设置为 16000Hz。

#### 【F\_SACM\_DVR1600\_ADC\_Timer\_X4】

[汇编格式]: F\_SACM\_DVR1600\_ADC\_Timer\_X4

[实现功能]: 当 ADC\_FIRType 选定为 2 时, 该函数被 SACM\_DVR1600\_Rec 函数自动调用, 用于初始化 Timer。

[参数]: 无

[返回值]: 无

[说明]: 在一般应用中, 该函数应将 Timer 溢出频率设置为 32000Hz。

#### 【F\_SACM\_DVR1600\_GetADC】

[汇编格式]: F\_SACM\_DVR1600\_GetADC

[实现功能]: 在录音过程中, 该函数被 SACM\_DVR1600\_ISR 函数自动调用, 用于获取 A/D 转换值。

[参数]: 无

[返回值]: (R1): ADC 采集到的数字量。

[说明]: ADC 采集到的数字量通过 R1 传递给 SACM\_DVR1600\_ISR 函数, 以完成一次采样。

#### 【F\_SACM\_DVR1600\_EndRecord】

[汇编格式]: F\_SACM\_DVR1600\_EndRecord

**[实现功能]:** 当一段语音录制完毕时, 该函数将被 SACM\_DVR1600\_Stop 函数自动调用, 用来执行自定义程序。

**[参数]:** 无

**[返回值]:** 无

**[说明]:** 无

### 3.2.3 SACM\_DVR1600\_User\_C.c与SACM\_DVR1600\_User\_ASM.asm中的录音相关函数

#### 【USER\_DVR1600\_SaveResource\_Init】

**[C 格式]:** void USER\_DVR1600\_SaveResource\_Init(unsigned int UserParam);

**[汇编格式]:** \_USER\_DVR1600\_SaveResource\_Init

**[实现功能]:** 在语音录制开始时, 该函数将被 SACM\_DVR1600\_Rec 函数自动调用, 可利用该函数来初始化语音资源存储区的起始地址。

**[参数]:** UserParam (R1): 由 SACM\_DVR1600\_Rec 函数传递过来的自定义参数。

**[返回值]:** 无

**[说明]:** 用户可以自行编写该函数体中的内容, 在录音之前对语音存储介质进行初始化操作。UserParam 是由 SACM\_DVR1600\_Rec 函数传递过来的参数, 可利用该参数对初始化过程进行控制。例如在进行多段录音时, 可由 UserParam 来确定保存当前这段语音的存储器起始地址。

应注意的问题是, 在初始化目标存储地址时, 要在存储区的起始地址处保留 2 Words (即 4 Bytes) 的空间, 待录音结束后要将录制的语音资源长度信息保存在这 2 Words 空间中。

#### 【USER\_DVR1600\_SaveResource】

**[C 格式]:** void USER\_DVR1600\_SaveResource(unsigned int \*p\_Buf, unsigned int Words);

**[汇编格式]:** \_USER\_DVR1600\_SaveResource

**[实现功能]:** 在 SACM\_DVR1600\_ServiceLoop 执行压缩编码时, 将会自动调用该函数, 保存编码后的语音资源。

**[参数]:** p\_Buf (R1): 压缩编码后得到的语音资源的源存储地址, 它是由库函数 SACM\_DVR1600\_ServiceLoop 自动传递过来的参数。用户要从以 p\_Buf 为起始地址的 RAM 缓冲区中提取编码后的语音资源数据, 并保存到目标存储介质中。

Words (R2): 需要从 p\_Buf 缓冲区中提取的数据长度, 单位为 Word。该参数也是由 SACM\_DVR1600\_ServiceLoop 函数自动传递过来的。

**[返回值]:** 无

**[说明]:** 用户可以自行编写该函数体中的内容, 从以 p\_Buf 为起始地址的 RAM 中提取编码数据并保存。应注意的, 每次执行该函数时, 从 p\_Buf 中提取数据的数量必须与 Words 参数所要求的数量一致。如果目标存储介质已满, 则可通过执行 SACM\_DVR1600\_Stop 函数来停止录音。在编写 USER\_DVR1600\_SaveResource 函数时, 应注意记录已获得编码数据的累计字节数, 供 USER\_DVR1600\_SaveResource\_End 函数使用。

**【USER\_DVR1600\_SaveResource\_End】**

**[C 格式]:** void USER\_DVR1600\_SaveResource\_End(void);

**[汇编格式]:** \_USER\_DVR1600\_SaveResource\_End

**[实现功能]:** 录音结束时，被 SACM\_DVR1600\_Stop 函数自动调用，将录制的语音资源长度信息（单位为 Byte）写入到语音资源存储区的起始 2 Words 处。

**[参数]:** 无

**[返回值]:** 无

**[说明]:** 用户可以自行编写该函数体中的内容，将已获得录音编码数据的字节数（用来记录数据长度的 4 Bytes 也包括在内）写入录音资源存储区的起始 2 Words（即 4 Bytes）。因此，用户在编写 USER\_DVR1600\_SaveResource 函数时应注意记录已获取的编码数据长度。

## 4 语音压缩工具SunAudior

### 4.1 SunAudior 简介

SACMv41dx 语音库附带了专门的语音压缩编码工具——SunAudior，该工具可对.wav 扩展名的语音文件进行压缩编码，生成 A1600 或 S720 格式的语音资源。SunAudior 工具操作简便，功能比较丰富，除可进行语音压缩外，还具有语音文件组织管理、编码文件与源文件对比试听、查看语音波形等功能。关于 SunAudior 的使用方法详见该软件的帮助文档，这里仅介绍基本的语音压缩方法。

运行SunAudior软件安装包，界面如图 4.1所示，连续点击Next按钮即可完成软件的安装（也可在安装过程手工选择安装路径，方法略）。

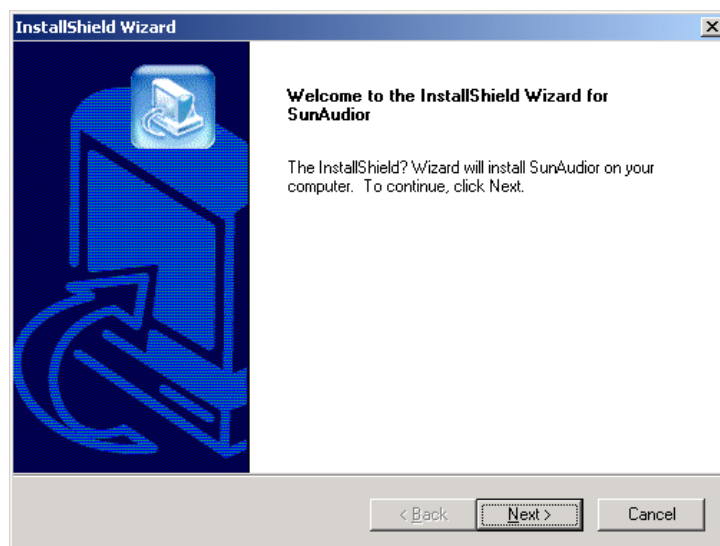


图 4.1 SunAudior 安装界面

安装成功后，将在桌面生成SunAudior快捷图标，在开始菜单的“程序→Sunplus→SunAudior”中也可找到该软件的快捷方式。打开SunAudior软件，其界面如图 4.2所示。

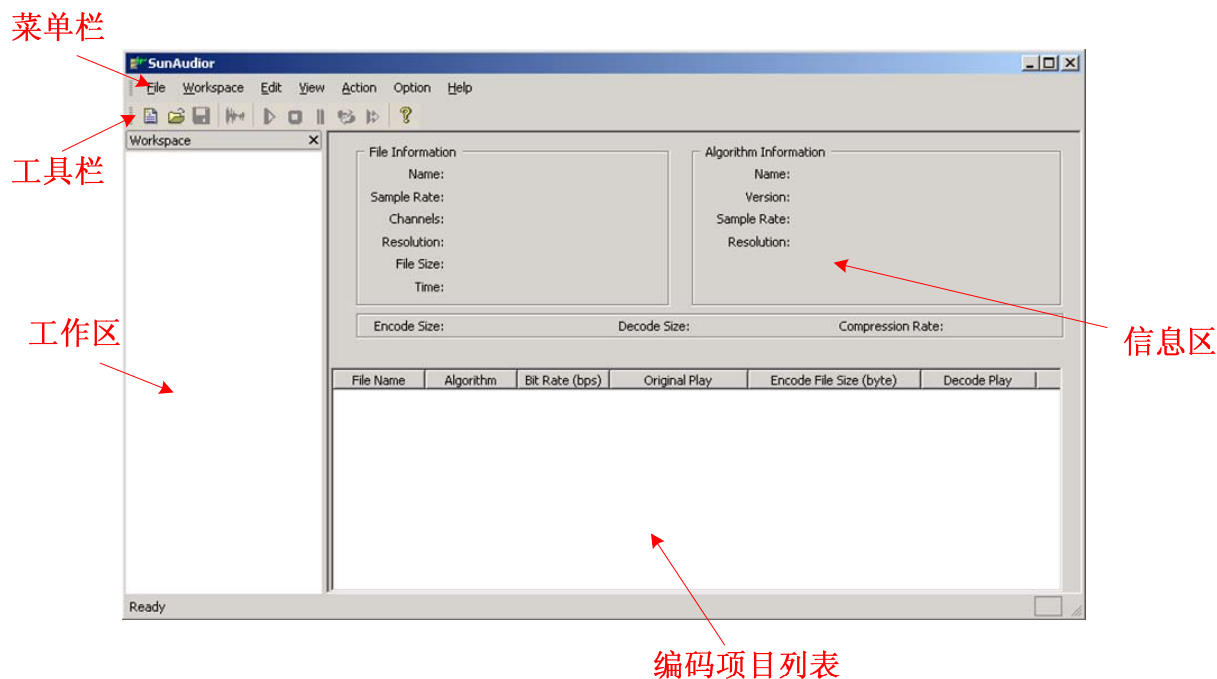


图 4.2 SunAudior 操作界面

菜单栏：可完成对 SunAudior 软件的全部操作。

工具栏：部分常用操作的快捷图标。

工作区：在该区域显示 wav 文件列表及其组织结构。

信息区：显示 wav 文件、编码算法以及编码项目的信息。

编码项目列表：显示选定的 wav 文件对应的压缩编码后的语音资源信息。

## 4.2 使用SunAudior软件进行语音压缩

### 【第 1 步】 建立工程。

SunAudior是以“工程”为单位来组织语音资源的。在菜单栏中选择“File→NewProject”，弹出新建工程对话框，如图 4.3所示。在Project Name栏中填入新工程的名称，在Project Path栏中选定工程所在的路径，点击“OK”，即可建立一个SunAudior工程。

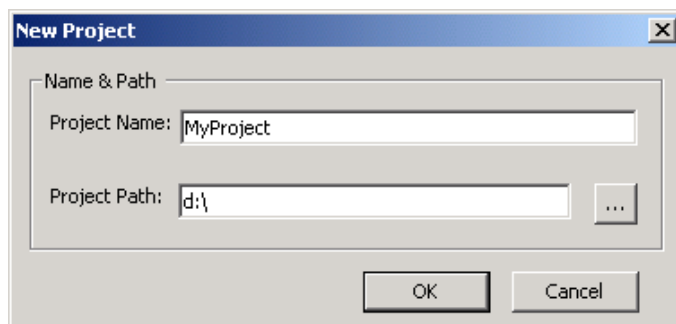


图 4.3 新建 SunAudior 工程



**【第2步】** 添加 wav 语音文件。

向刚刚建立的工程中加入欲压缩的wav文件。在Workspace菜单项中选择“Add Files”，弹出添加文件的对话框，如图 4.4、图 4.5所示。

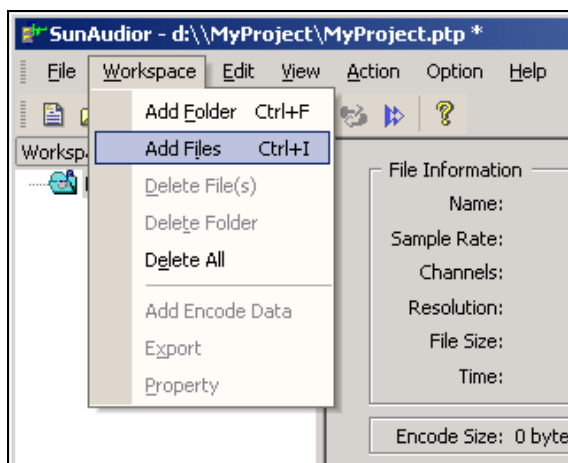


图 4.4 添加文件

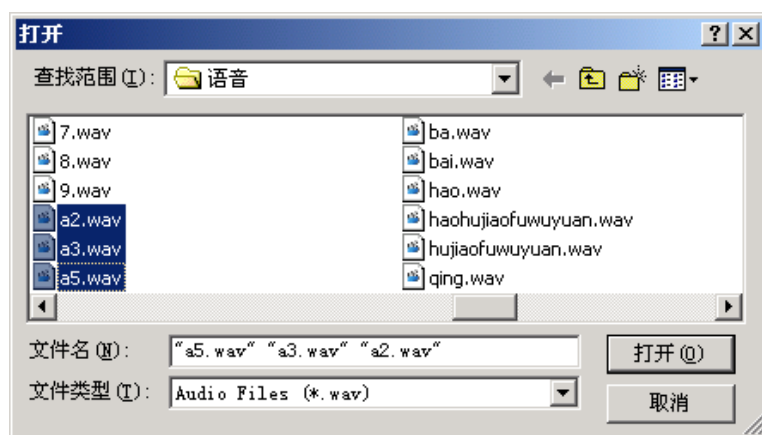


图 4.5 选择欲压缩的 wav 文件

**【注意】** wav 文件的格式。

应注意的是，SunAudior 软件只支持“8KHz、16 位、单声道、PCM 编码”格式的 wav 文件。对于不符合上述格式的 wav 文件，SunAudior 软件有可能无法压缩，或压缩后不能正常播放。

使用Windows自带的“录音机”软件即可查看wav文件格式是否符合SunAudior的要求，并且可以把不符合格式要求的wav文件转换为符合要求的文件。在Windows开始菜单下选择“程序→附件→娱乐→录音机”，启动录音机软件，如图 4.6所示。





图 4.6 Windows 自带的录音机软件

在录音机软件中选择“文件→打开”，打开希望查看或转换格式的wav文件。然后在文件菜单中选择属性，即可查看该wav文件的格式，如图 4.7、图 4.8所示。



图 4.7 查看 wav 文件属性

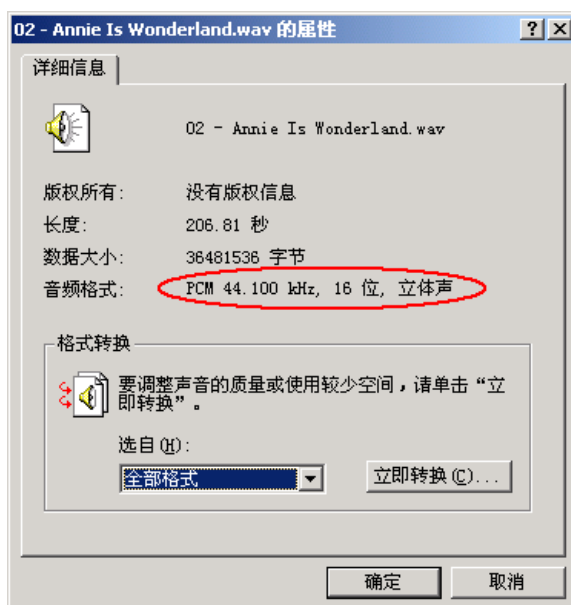


图 4.8 wav 文件格式

如果该wav文件的格式不是“PCM，8.000 kHz，16 位，单声道”，则可以点击图 4.8界面中的“立即转换”按钮，对wav文件进行格式转换。在“立即转换”弹出的对话框中，将格式选定为“PCM”，属性选定为“8.000 kHz，16 位，单声道”，如图 4.9所示。



图 4.9 格式转换

点击确定后，就完成了格式转换。保存转换后的文件，即可利用 SunAudior 工具进行压缩了。

### 【第 3 步】 执行压缩编码。

在 SunAudior 左半部分的工作区中选中待压缩的文件（可以按住 Ctrl 键选中多个文件，或按 Ctrl+A 全部选择），点击右键，在弹出的菜单中选“Add Encode Data”，对所选文件执行压缩编码，如图 4.10 所示。

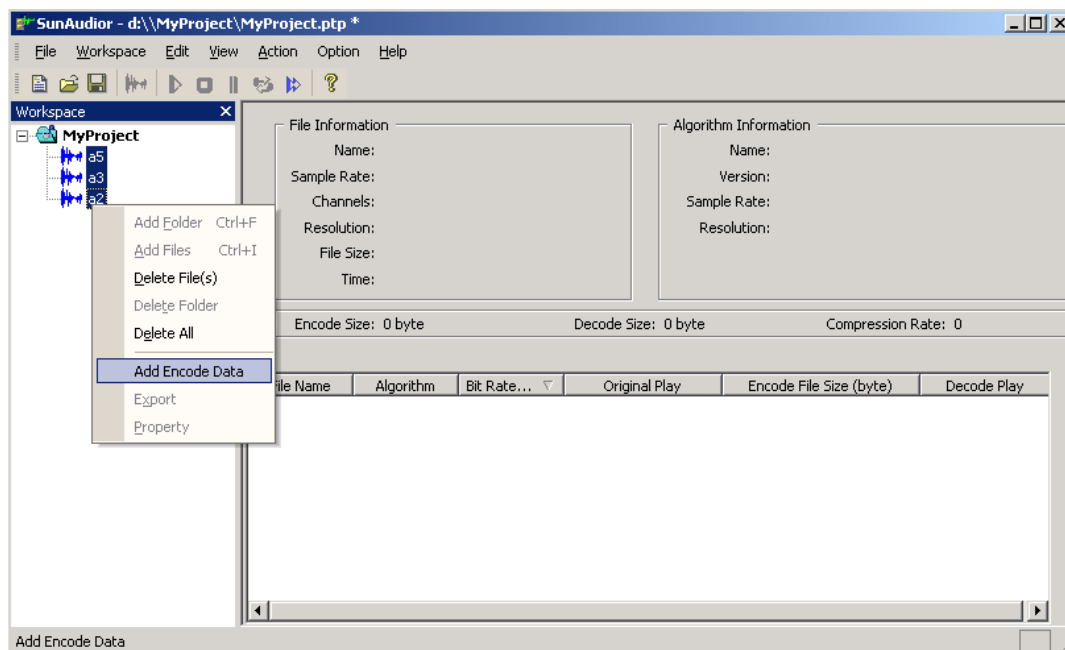


图 4.10 执行压缩编码

在弹出的压缩编码对话框中，“Name”下拉列表用来选择压缩编码算法（A1600 或 S720），“Bit Rate”下拉列表用来选择压缩后的码率（A1600 可选 10K~24Kbps 码率，S720 可选 4.8K 或 7.2Kbps 码率），如图 4.11 所示。选择好编码算法和码率后，点击“OK”即开始压缩编码。

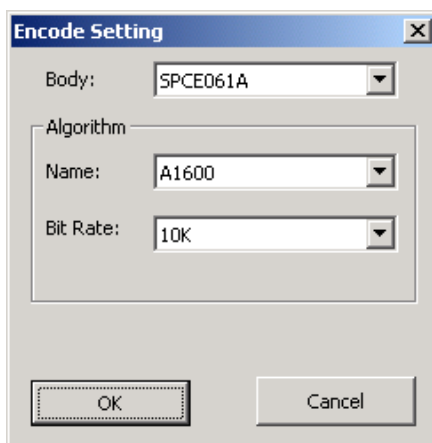


图 4.11 压缩编码选项

压缩编码完成后，编码项目将在列表中显示，如图 4.12所示，可分别点击“Original Play”和“Decode Play”栏中的按钮，对比试听原始wav文件和编码后语音资源。

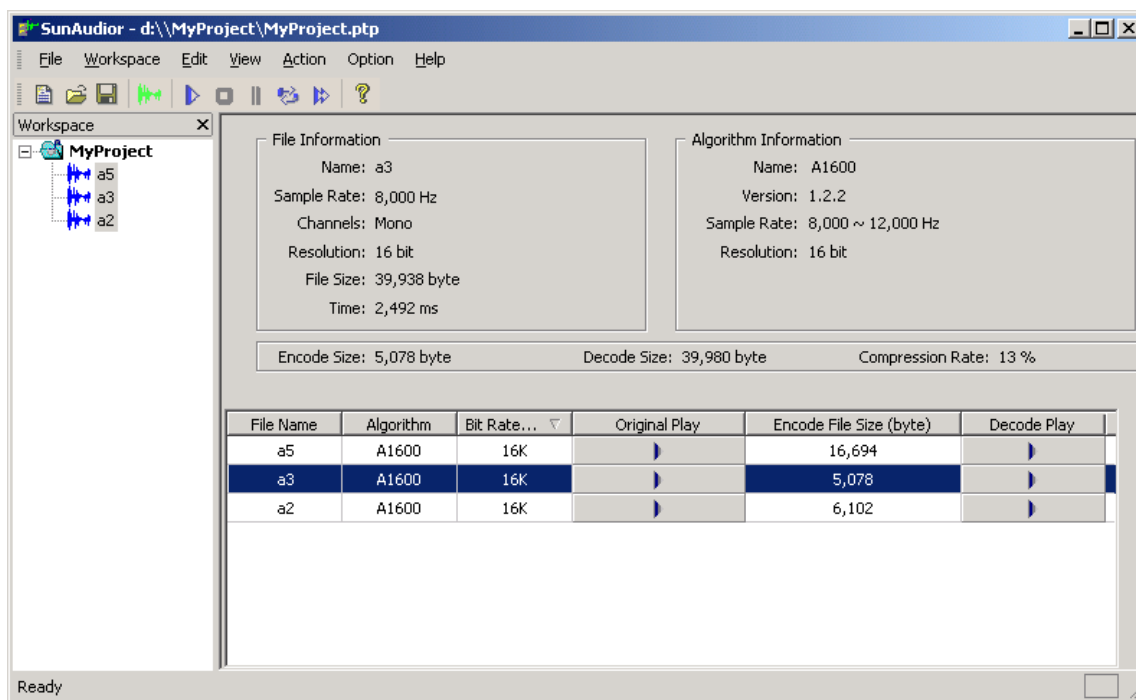


图 4.12 编码项目列表

#### 【第 4 步】 导出编码项目。

SunAudior压缩编码之后的项目是以临时文件的形式存放的，需要将其导出为语音资源，才可以应用于SACMv41dx语音播放。导出编码项目的方法是，在SunAudior左半部分的工作区选择需要导出为语音资源的文件（可按住Ctrl键选择多个文件，也可Ctrl+A选定全部），点击右键，执行“Export”选项，如图 4.13所示。

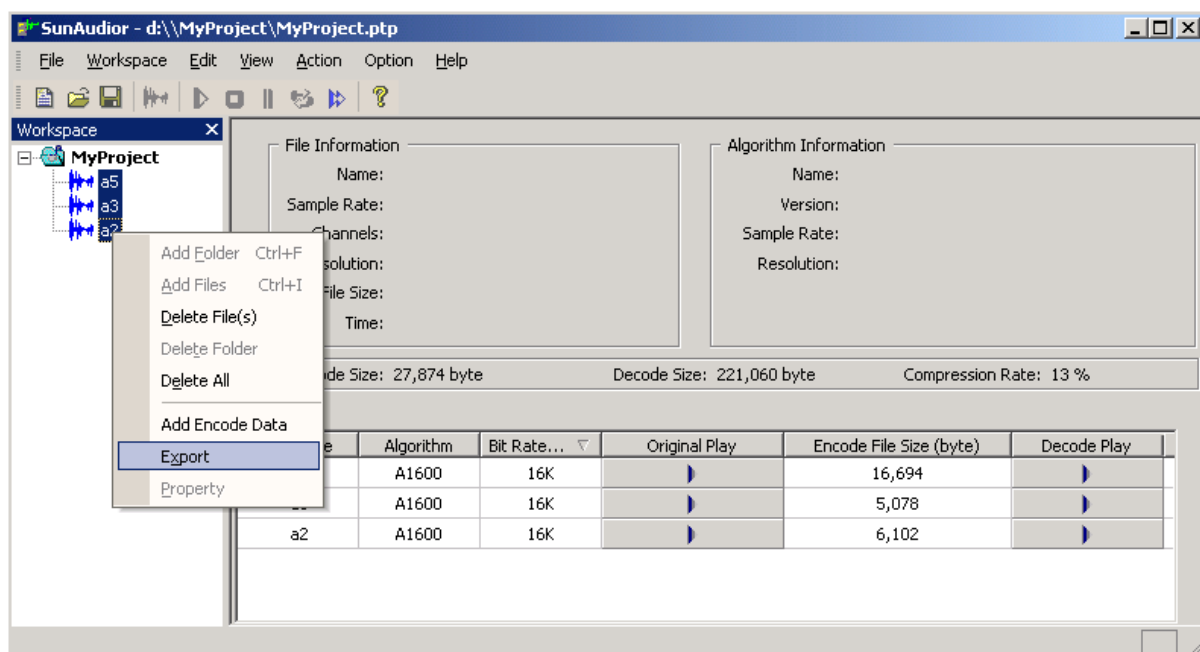


图 4.13 导出语音资源

在弹出的导出对话框中选定导出的语音资源的保存路径，点“OK”即可，如图 4.14所示。

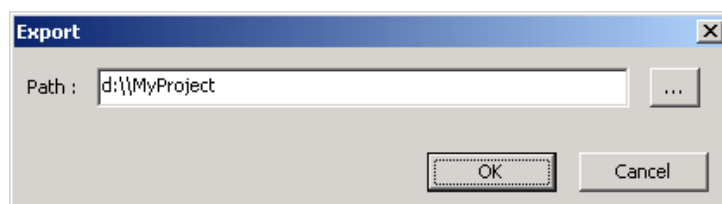


图 4.14 选择导出路径

导出的语音资源与原始 wav 文件的名称相同，而其扩展名具有以下规律（假设原始 wav 文件为“xxx.wav”）：

表 4.1 输出语音资源的命名规律

编码算法	码率	输出文件名
A1600	10K	xxx.A10
	12K	xxx.A12
	14K	xxx.A14
	16K	xxx.A16
	20K	xxx.A20
	24K	xxx.A24
S720	4.8K	xxx.S48
	7.2K	xxx.S72

应注意的是，如果将导出的语音资源保存在了当前工程所在的文件夹下（SunAudior 默认的就是这种情况），那么在退出 SunAudior 软件之前应保存工程（File 菜单下的 Save Project），否则在退出 SunAudior 后，未保存的工程文件夹中的文件将被自动清空。

## 5 应用举例

### 5.1 语音播放

对于 A1600、DVR1600、S720 和 MS01 语音播放，其程序结构是完全相同的，下面以 A1600 为例，介绍编写语音播放程序的基本步骤。

#### 【第 1 步】建立工程。

打开 unSP IDE 集成开发环境，选择 File→New Project 菜单项，建立新工程（这里在 D 盘下建立一个名为 SoundPlay 的工程）。

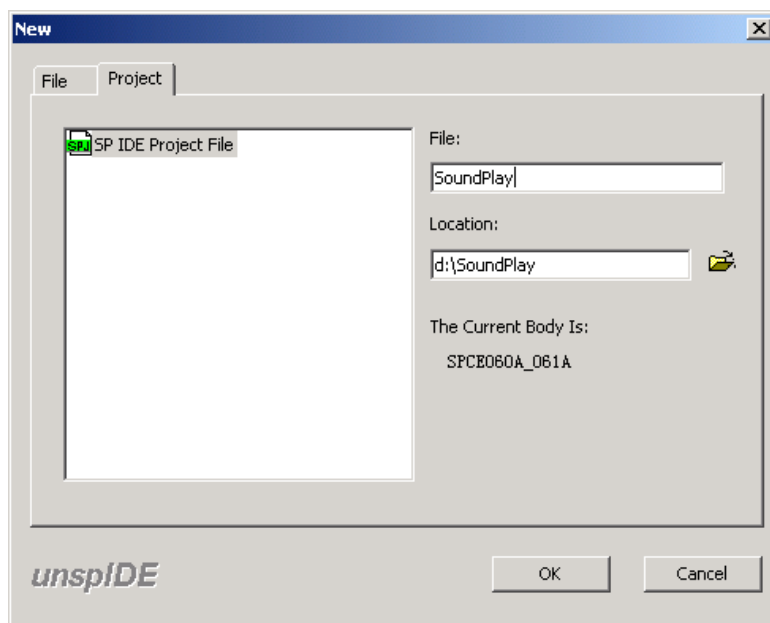


图 5.1 新建工程

#### 【第 2 步】复制语音资源和 SACMv41dx 相关文件到工程文件夹中。

将待播放的语音资源（采用 SunAudior 压缩并导出的资源文件，扩展名为 A10、A12、... 、A24）复制到工程所在的文件夹，这里假设有三个待播放资源：Sound0.A24、Sound1.A16 和 Sound2.A10。

在 SACMv41dx Files 文件夹中找到 A1600 相关文件，包括 SACMv41dx\_061A.lib 、SACM\_A1600.asm、SACM\_A1600\_User\_C.c（或 SACM\_A1600\_User\_ASM.asm）、A1600.h 和 A1600.inc。将这些文件复制到工程所在的文件夹。

复制文件之后的工程文件夹如图 5.2所示。

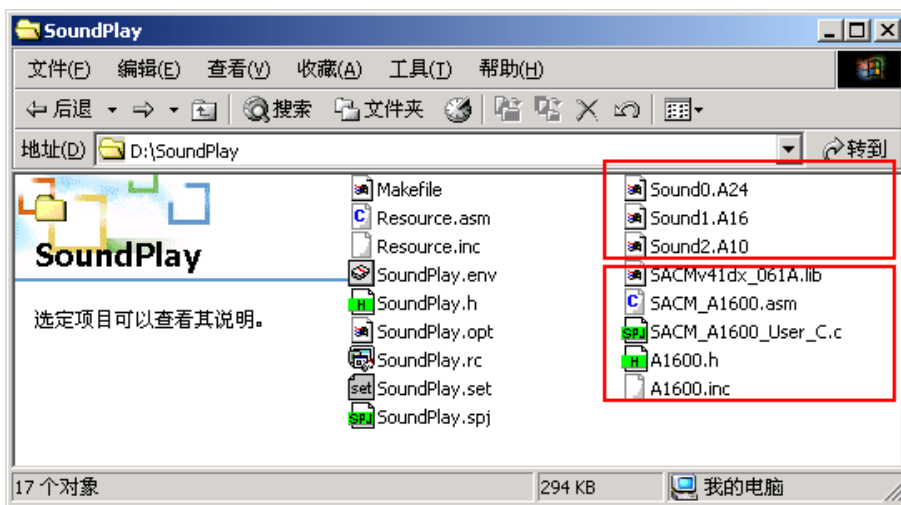


图 5.2 复制语音资源和 SACMv41dx 相关文件到工程文件夹

【第 3 步】将语音资源添加到工程。

向工程中添加语音资源的方法是，在IDE的Project菜单中选择“Add To Project→Resource”，在弹出的对话框中选择待播放的语音资源（按住Ctrl键选择多个文件），如图 5.3、图 5.4所示。

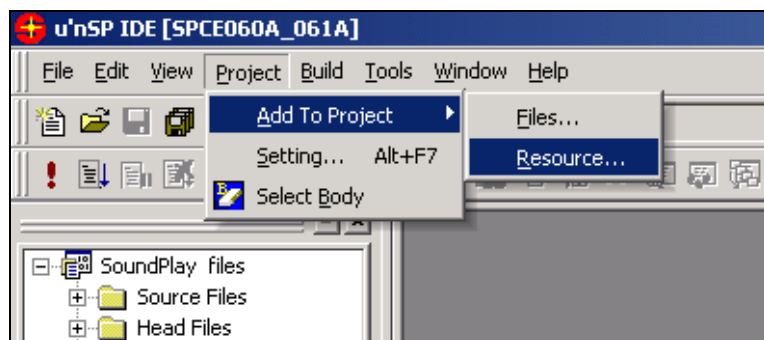


图 5.3 向工程添加语音资源

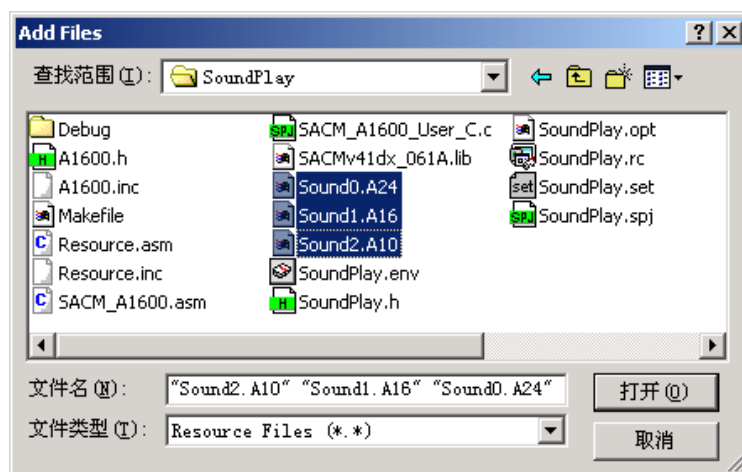


图 5.4 选择语音资源文件

添加语音资源后，可以在IDE工作区的“Resource View”栏中看到加入工程的语音资源，如图 5.5所示。

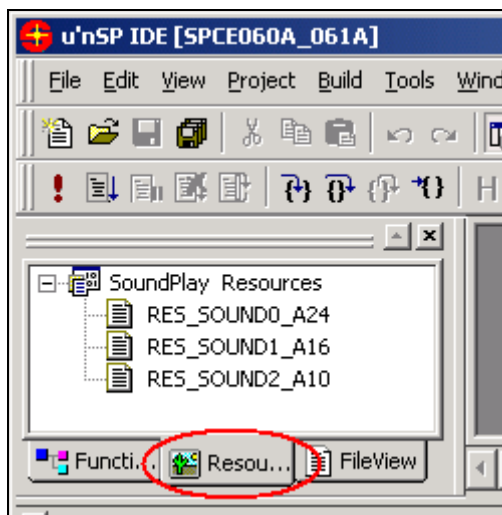



图 5.5 查看加入工程中的语音资源

【第 4 步】将 SACMv41dx\_061A.lib 添加到工程。

向工程中添加SACMv41dx\_061A.lib函数库的方法是：选择Project菜单的Setting项，在弹出的对话框中选择“Link”标签（需选中弹出对话框左面列表的根项目，才会出现Link标签），点击Library Modules项目中的“”图标，如图 5.6、图 5.7所示。

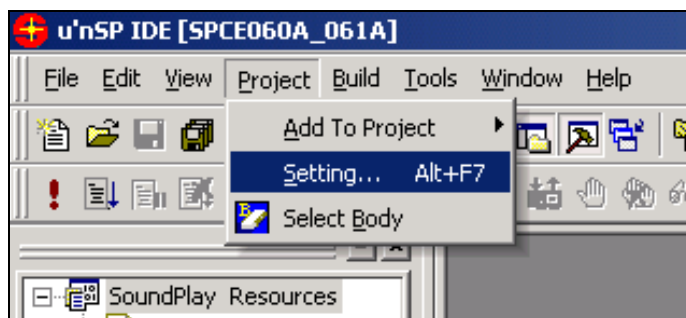


图 5.6 选择 Setting 项



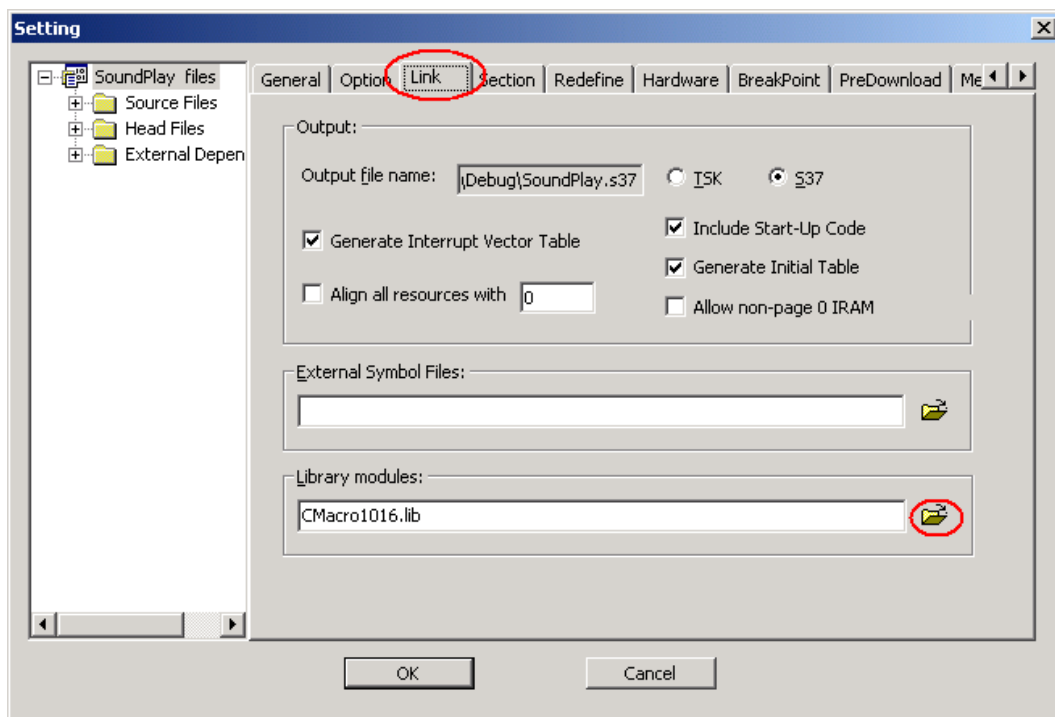


图 5.7 选择 Link 标签项

在弹出的文件选择对话框中选SACMv41dx\_061A.lib文件，点“打开”后即把SACMv41dx库添加到工程中了。如图 5.8所示。

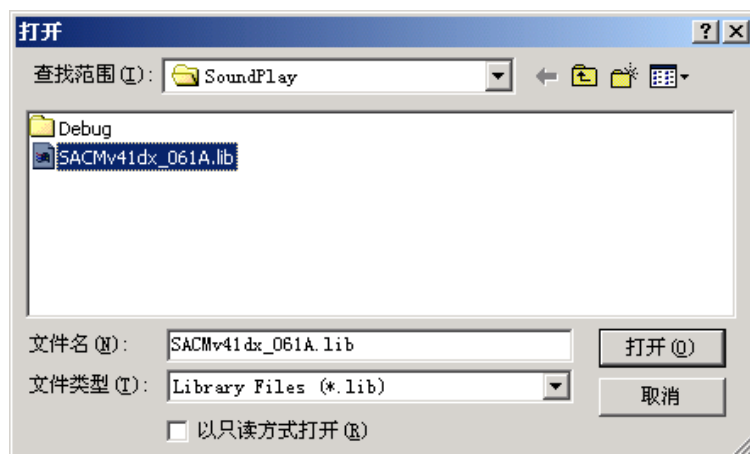


图 5.8 选择 SACMv41dx\_061A.lib 文件

【第 5 步】将 A1600 相关程序文件添加到工程。

相关程序文件包括SACM\_A1600.asm和SACM\_A1600\_User\_C.c。在IDE的Project菜单下选择“Add To Project→Files”，在弹出的对话框中选中两个程序文件并打开，如图 5.9、图 5.10所示。

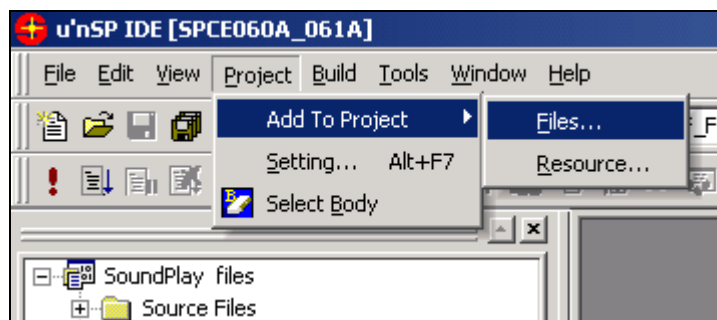


图 5.9 添加文件

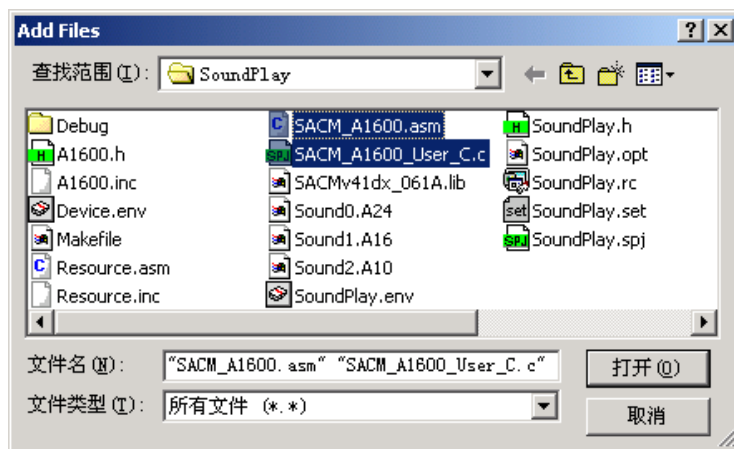


图 5.10 选中两个程序文件

#### 【第 6 步】编写语音播放程序

在工程中新建 C 程序文件（这里命名为 main.c），编写一个语音播放的函数，

```
#include "SPCE061A.h"
#include "A1600.h"

void PlaySnd(unsigned int SoundIndex)
{
    SACM_A1600_Initial();           // A1600 初始化
    SACM_A1600_Play(SoundIndex, 3, 3); // 播放第 SoundIndex 段语音，由 DAC1 和 DAC2 输出
    while(SACM_A1600_Status() & 0x01) // 检测播放是否完成
    {
        *P_Watchdog_Clear = 0x01; // 清看门狗
        SACM_A1600_ServiceLoop(); // 执行语音解码
    }
}
```

上面的函数完全是按照图 2.2 的流程来编写的。函数带有一个参数 SoundIndex，即待播放语音的序

号，这样可以通过调用该函数来播放指定的语音。

可以在 PlaySnd 函数下面编写主函数，调用 PlaySnd 函数，分别播放 Sound0.A24、Sound1.A16 和 Sound2.A10 三段语音资源：

```
int main()
{
    while(1)
    {
        PlaySnd(0);           // 播放第一段语音
        PlaySnd(1);           // 播放第二段语音
        PlaySnd(2);           // 播放第三段语音
    }
}
```

默认情况下，在 A1600 初始化时，将自动开启 FIQ\_TimerA 中断（在 SACM\_A1600.asm 的 F\_SACM\_A1600\_Init 函数中设置），因此要编写 FIQ\_TimerA 中断服务程序。可在 main 函数下面编写 C 语言版的 FIQ 中断服务函数：

```
void FIQ(void) __attribute__((ISR));           // 将 FIQ 声明为中断服务函数
void FIQ()
{
    *P_INT_Clear = 0x2000;                     // 清 FIQ_TimerA 中断请求标志
    SACM_A1600_ISR();                          // 调用 A1600 中断服务子程序
}
```

#### 【第 7 步】修改资源访问程序 SACM\_A1600\_User\_C.c。

A1600 访问语音资源数据的函数在 SACM\_A1600\_User\_C.c（或 SACM\_A1600\_User\_ASM.asm）中定义，包括 USER\_A1600\_GetResource\_Init 和 USER\_A1600\_GetResource 两个函数。由于当前例子中的语音资源是加载到 SPCE061A 内部 Flash 中的，因此，USER\_A1600\_GetResource\_Init 函数的任务是获取待播放语音资源在 Flash 中的起始存储地址，而 USER\_A1600\_GetResource 函数的任务是从 Flash 中读取语音资源数据。

下面将讨论如何获取待播放语音资源的起始存储地址。

对于利用 IDE 的“Project→Add To Project→Resource”添加到工程中的资源，IDE 会为它们自动分配一段 Flash 存储空间。点击 IDE 的“Build”（快捷键为 F7，由于当前这个程序还没有编写完成，所以可能会在 Build 过程中提示出错，暂时可以不必理会这些错误），查看工程中的 Resource.asm 文件，可以发现 IDE 自动生成了一些代码：

```
// Resource Table
// Created by IDE, Do not modify this table
.....
```

```
_RES_Table:

_RES_SOUND0_A24_SA:
    .DW offset __RES_SOUND0_A24_sa,seg __RES_SOUND0_A24_sa;
_RES_SOUND0_A24_EA:
    .DW offset __RES_SOUND0_A24_ea,seg __RES_SOUND0_A24_ea;
_RES_SOUND1_A16_SA:
    .DW offset __RES_SOUND1_A16_sa,seg __RES_SOUND1_A16_sa;
_RES_SOUND1_A16_EA:
    .DW offset __RES_SOUND1_A16_ea,seg __RES_SOUND1_A16_ea;
_RES_SOUND2_A10_SA:
    .DW offset __RES_SOUND2_A10_sa,seg __RES_SOUND2_A10_sa;
_RES_SOUND2_A10_EA:
    .DW offset __RES_SOUND2_A10_ea,seg __RES_SOUND2_A10_ea;

// End Table
```

上面的自动生成的代码中，`_RES_SOUND0_A24_SA`、`_RES_SOUND1_A16_SA` 和 `_RES_SOUND2_A10_SA` 三个标号下分别存储了 Sound0.A24、Sound1.A16 和 Sound2.A10 三个语音资源的起始地址。该起始地址由两个 Word 组成，“offset \_\_RES\_....\_sa”表示段内偏移量，“seg \_\_RES\_...\_sa”表示段基址（unSP 内核具有 22 bit 地址总线，以 6 bit 段基址+16 bit 偏移量的形式寻址。对于 SPCE061A 来说，仅 0x0000~0xFFFF 地址有效，因此段基址始终为 0，只需关心段内偏移量即可）。

`_RES_SOUND0_A24_EA`、`_RES_SOUND1_A16_EA` 和 `_RES_SOUND2_A10_EA` 三个标号下分别存储了三个语音资源的结束地址。事实上，语音资源何时播放结束是由语音播放程序自动判断的，不必关注资源的结束地址，只要能够获取起始地址就可以了。

可以在 Resource.asm 文件的尾部添加一个语音资源起始地址的索引表（必须添加在“// End Table”行的下面，否则会被 IDE 自动生成的代码覆盖掉），供 USER\_A1600\_GetResource\_Init 函数访问：

```
..... // IDE 自动生成的内容

// End Table

.PUBLIC _SACM_A1600_SpeechTable // 创建语音资源起始地址的索引，供外部程序调用
_SACM_A1600_SpeechTable:
    .DW _RES_SOUND0_A24_SA // 存储“第 1 段语音的起始地址”的地址
    .DW _RES_SOUND1_A16_SA // 存储“第 2 段语音的起始地址”的地址
    .DW _RES_SOUND2_A10_SA // 存储“第 3 段语音的起始地址”的地址
```

这样，SACM\_A1600\_User\_C.c 中的 USER\_A1600\_GetResource\_Init 函数就可以根据

SACM\_A1600\_Play 函数传递过来的语音序号 SoundIndex 来确定待播放语音资源的起始地址。  
USER\_A1600\_GetResource\_Init 函数可以写成：

```
extern unsigned int *SACM_A1600_SpeechTable; // 语音资源起始地址列表，在 resource.asm 中定义
unsigned int ResAddr;                        // 全局变量，用于记录资源数据地址

void USER_A1600_GetResource_Init(unsigned int SoundIndex)
{
    unsigned int *p_Addr;
    p_Addr = (&SACM_A1600_SpeechTable + SoundIndex); // 得到_RES_xxxx_SA，赋给 p_Addr
    ResAddr = *p_Addr;                               // 得到_RES_xxxx_SA 中存储的内容，即该资源的起始地址
}
```

对于 USER\_A1600\_GetResource 函数，其两个参数 p\_Buf 和 Words 都是由库函数自动传递过来的，可以利用全局变量 ResAddr 将解码所需的语音资源数据填入起始地址为 p\_Buf 空间中。USER\_A1600\_GetResource 函数的代码如下：

```
void USER_A1600_GetResource(unsigned int *p_Buf, unsigned int Words)
{
    while(Words>0)
    {
        *p_Buf = *(unsigned int*)ResAddr; // 获取语音资源数据，填入解码队列
        p_Buf++;                          // 更新队列指针
        ResAddr++;                        // 更新语音资源地址
        Words--;                          // 更新剩余的待填充数据长度
    }
}
```

#### 【第 8 步】Build 工程并下载运行。

执行Build菜单中的“Rebuild All”，如果没有语法错误就可以将程序下载到SPCE061A单片机中了。在下载之前请确定IDE的“Use ICE”图标被选中，如图 5.11所示。

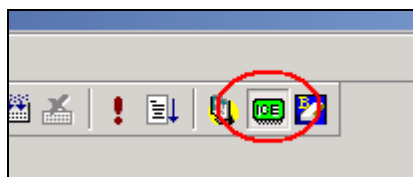


图 5.11 选择“Use ICE”

按 F5 键，下载程序并全速运行，单片机将循环播放三段语音。

## 5.2 语音录制 (DVR1600)

本例实现的功能是，利用SPCE061A内部Flash录制一段语音并播放。关于DVR1600 语音录制，其程序流程请参考图 2.4。

### 【第 1 步】建立工程。

参考5.1节的第 1 步，新建一个工程（这里命名为SoundRecord）。

### 【第 2 步】SACMv41dx 相关文件到工程文件夹中。

在 SACMv41dx Files 文件夹中找到 DVR1600 相关文件，包括 SACMv41dx\_061A.lib 、 SACM\_DVR1600.asm 、 SACM\_DVR1600\_User\_C.c （或 SACM\_DVR1600\_User\_ASM.asm ） 、 DVR1600.h 和 DVR1600.inc。将这些文件复制到工程所在的文件夹。

复制文件之后的工程文件夹如图 5.12所示。

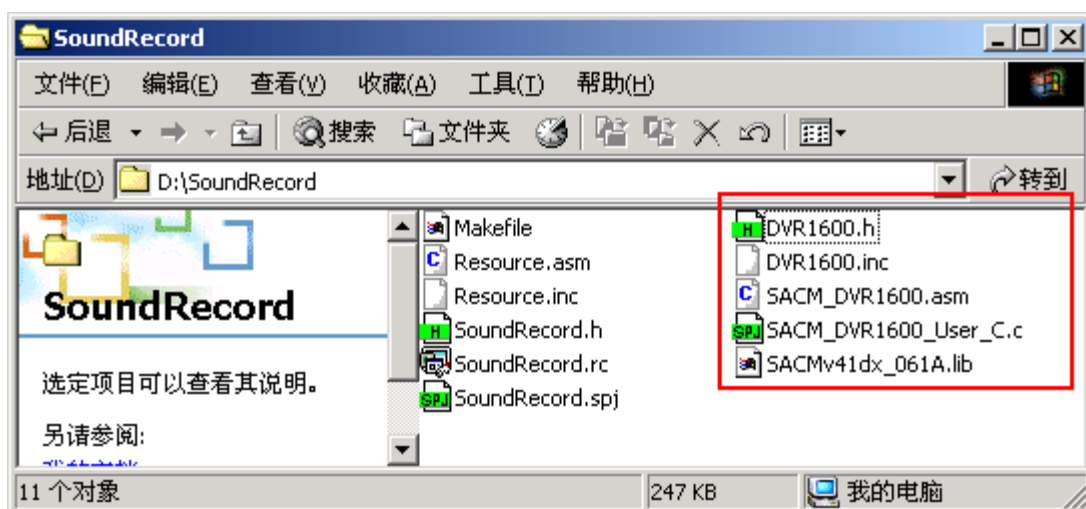


图 5.12 复制 SACMv41dx 相关文件到工程文件夹

### 【第 3 步】将 SACMv41dx\_061A.lib 和 DVR1600 程序文件添加到工程。

参考5.1节的第 4、5 步，选择IDE的“Project→Setting→Link”，将SACMv41dx\_061A.lib加入工程；再选择“Project→Add To Project→Files”，将SACM\_DVR1600\_User\_C.c和SACM\_DVR1600.asm两个程序文件添加到工程中。

### 【第 4 步】编写语音录制和播放程序

在工程中新建 C 程序文件（这里命名为 main.c），编写一个录音函数和一个放音函数。

```
#include "SPCE061A.h"
#include "DVR1600.h"

void Record()
{
```

```
SACM_DVR1600_Initial();           // DVR1600 初始化

SACM_DVR1600_Rec(0,3);           // 开始录音, 选择 16Kbps 码率。第一个参数无意义。

while(SACM_DVR1600_Status()&0x01) // 检测录音是否完成(当录音数据充满 Flash 后录音完成)
{
    *P_Watchdog_Clear = 0x01;     // 清看门狗
    SACM_DVR1600_ServiceLoop();    // 进行压缩编码
}

}

void PlayRecord(void)
{
    SACM_DVR1600_Initial();         // DVR1600 初始化
    SACM_DVR1600_Play(0, 3, 3);    // 开始放音, 由 DAC1 和 DAC2 输出。第一个参数无意义。
    while(SACM_DVR1600_Status()&0x01) // 检测放音是否完成
    {
        *P_Watchdog_Clear = 0x01; // 清看门狗
        SACM_DVR1600_ServiceLoop(); // 进行解码
    }
}
```

可在录音和放音函数之下继续编写 main 函数以及中断服务程序:

```
int main()
{
    Record();           // 录音
    PlayRecord();       // 放音
    while(1)           // 录放音结束后进入死循环
    {
        *P_Watchdog_Clear = 0x01;
    }
}

void FIQ(void)__attribute__((ISR)); // 将 FIQ 声明为中断服务函数
void FIQ()
{
    *P_INT_Clear = 0x2000; // 清 FIQ_TimerA 中断请求标志
    SACM_DVR1600_ISR();    // 调用 DVR1600 中断服务子程序
}
```

**【第 5 步】** 修改资源访问程序 SACM\_DVR1600\_User\_C.c。

SACM\_DVR1600\_User\_C.c 文件中包括与录音相关的 USER\_DVR1600\_SaveResource\_Init、USER\_DVR1600\_SaveResource 和 USER\_DVR1600\_SaveResource\_End 函数，以及与放音相关的 USER\_DVR1600\_GetResource\_Init 和 USER\_DVR1600\_GetResource 函数。

对于录音过程，资源存储初始化函数 USER\_DVR1600\_SaveResource\_Init 需要完成的工作是，擦除录音所需的 Flash 空间，并初始化录音资源的存储地址。Flash 擦写程序可参考 SPCE061A 的相关资料，这里编写了实现 Flash 擦除和改写功能的函数：

```
#include "SPCE061A.h"
#include "DVR1600.h"

/*****

// Flash 擦除程序，擦除 FlashAddr 所在的页
*****/

void Flash_Erase(unsigned int FlashAddr)
{
    *P_Flash_Ctrl = 0xAAAA;
    *P_Flash_Ctrl = 0x5511;
    *(unsigned int *)FlashAddr = 0;
}

/*****

// Flash 写入程序，向 FlashAddr 地址写入数据 Data
*****/

void Flash_WriteWord(unsigned int FlashAddr, unsigned Data)
{
    *P_Flash_Ctrl = 0xAAAA;
    *P_Flash_Ctrl = 0x5533;
    *(unsigned int *)FlashAddr = Data;
}

/*****

// 以下是 DVR1600 的资源访问相关程序
*****/

unsigned int ResAddr;                // 全局变量，用于记录资源数据地址

#define RECORD_SA 0xC000             // 录音资源起始地址
```



```
#define RECORD_EA 0xEFFF // 录音资源结束地址

void USER_DVR1600_SaveResource_Init(unsigned int UserParam)
{
    unsigned int Addr;
    for(Addr=RECORD_SA; Addr<=RECORD_EA; Addr+=0x0100)
    {
        Flash_Erase(Addr); // 擦除录音所需的 Flash
        *P_Watchdog_Clear = 0x01;
    }
    ResAddr = RECORD_SA + 2; // 跳过前两个 Word（用于录音结束时保存资源长度）
}
```

资源存储函数 USER\_DVR1600\_SaveResource 则根据库函数传递过来的参数 p\_Buf 和 Words，将编码后的数据保存到 Flash 中。

```
void USER_DVR1600_SaveResource(unsigned int *p_Buf, unsigned int Words)
{
    while(Words--)
    {
        Flash_WriteWord(ResAddr++, *p_Buf++); // 将编码后的数据写入 Flash
        if(ResAddr>RECORD_EA) // 达到结束地址则停止录音
        {
            SACM_DVR1600_Stop();
            break;
        }
    }
}
```

录音结束时，将自动调用 USER\_DVR1600\_SaveResource\_End 函数，该函数的任务是将录音编码的长度信息写入到该资源的起始存储地址处。

```
void USER_DVR1600_SaveResource_End(void)
{
    unsigned int ResSize;
    ResSize = (ResAddr - RECORD_SA)<<1; // 计算录音资源的长度，转换为 Byte 单位
    Flash_WriteWord(RECORD_SA, ResSize); // 资源长度的低 16 位写入 RECORD_SA 中
    Flash_WriteWord(RECORD_SA+1, 0x0000); // 资源长度的高 16 位填充 0
}
```

对于播放语音相关的两个资源访问函数，则与 5.1 节的例子相仿，所不同的是 USER\_DVR1600\_GetResource\_Init 函数初始化语音资源地址时，直接将全局变量 ResAddr 指向 RECORD\_SA 即可。

```
void USER_DVR1600_GetResource_Init(unsigned int SoundIndex)
{
    ResAddr = RECORD_SA;           // 初始化语音资源的起始地址
}

void USER_DVR1600_GetResource(unsigned int *p_Buf, unsigned int Words)
{
    while(Words>0)
    {
        *p_Buf = *(unsigned int*)ResAddr; // 获取语音资源数据，填入解码队列
        p_Buf++;                          // 更新队列指针
        ResAddr++;                         // 更新语音资源地址
        Words--;                           // 更新剩余的待填充数据长度
    }
}
```

**【第 6 步】** Build 工程并下载运行。

参考 5.1 节的第 7 步，Build 工程，下载并全速运行。SPCE061A 将录音约 14 秒，然后将录制的声音播放出来。

## 6 常见问题

### ◆ A1600 语音资源的数据格式

以 A1600 算法编码的语音资源是以“帧”为单位组织起来的。文件格式如下：

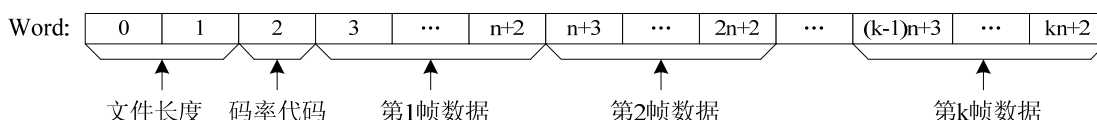


图 6.1 A1600 语音资源的数据格式

由 Word 0 和 Word 1 组成的长整型数据（Word 0 存储低 16 位，Word 1 存储高 16 位）用来表示整个语音资源文件的长度，单位为 byte。对于一个帧数为 k 的语音资源，其文件长度为  $(k \cdot n + 3) \cdot 2$  字节，其中 n 代表每帧数据的长度（由 Word 2 决定，见表 6.1）。

Word 2 用来表示该语音的码率。Word 2 中的数值与码率、帧长度的关系如下：

表 6.1 Word 2 与码率、帧长度的对应关系

Word 2 中的数值	码率 (bps)	帧长度 n (Words)
0x8004	10K	10
0x8005	12K	12
0x8006	14K	14
0x8007	16K	16
0x8008	20K	20
0x8009	24K	24

对于各种码率的 A1600 数据帧，解码后的帧长度都是 128 Words。

### ◆ S720 语音资源的数据格式

以 S720 算法编码的语音资源格式如下：

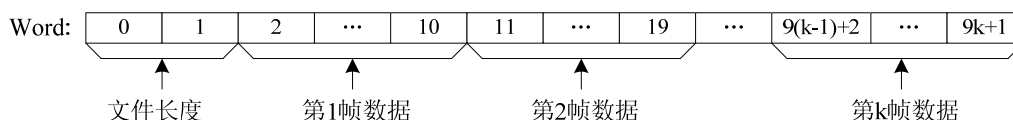


图 6.2 S720 语音资源的数据格式

由 Word 0 和 Word 1 组成的长整型数据（Word 0 存储低 16 位，Word 1 存储高 16 位）用来表示整个语音资源文件的长度，单位为 byte。对于一个帧数为 k 的语音资源，其文件长度为  $(k \cdot 9 + 2) \cdot 2$  字节，每帧数据长度为 9 Words。

对于以 4.8Kbps 码率编码的资源，每帧数据解码后的长度为 240 Words；对于以 7.2Kbps 码率编码的资源，每帧数据解码后的长度为 160 Words。

### ◆ 错误提示：The external symbol "\_SACM\_xxxx\_ISR" has not a public definition.

这个错误提示一般与下面的错误提示同时出现：

The external symbol "\_SACM\_xxxx\_Initial" has not a public definition.

The external symbol "\_SACM\_xxxx\_Play" has not a public definition.

The external symbol "\_SACM\_xxxx\_ServiceLoop" has not a public definition.

.....

出现这些错误的原因是没有把SACMv41dx\_061A.lib添加到工程中。请参考5.1节的第4步。

◆ **错误提示： The external symbol "F\_SACM\_xxxx\_Init" has not a public definition.**

这个错误提示一般与下面的错误提示同时出现：

The external symbol "F\_SACM\_xxxx\_RampUpDAC1" has not a public definition.

The external symbol "F\_SACM\_xxxx\_RampUpDAC2" has not a public definition.

The external symbol "F\_SACM\_xxxx\_EndPlay" has not a public definition.

.....

出现这些错误提示的原因是没有把SACM\_xxxx.asm文件添加到工程中。请参考5.1节的第5步。

◆ **错误提示： The external symbol "\_USER\_xxxx\_GetResource\_Init" has not a public definition.**

这个错误提示一般与下面的错误提示同时出现：

The external symbol "\_USER\_xxxx\_GetResource" has not a public definition.

出现这些错误提示的原因是没有把SACM\_xxxx\_User\_C.c或者SACM\_xxxx\_User\_ASM.asm添加到工程中。请参考5.1节的第5步。

◆ **错误提示： The external symbol "\_SACM\_xxxx\_SpeechTable" has not a public definition.**

用于访问语音资源的文件SACM\_xxxx\_User\_C.c或SACM\_xxxx\_User\_ASM.asm中，默认的程序是通过语音资源列表\_SACM\_xxxx\_SpeechTable来定位语音资源起始地址的。该列表应在Resource.asm中手工定义，参见5.1节的第7步。

◆ **错误提示： Can't locate DATA section automatically, please check it.**

SPCE061A的内部Flash-ROM空间为32K Word。当程序代码和语音资源占用的总空间超出32K Word的容量限制时，将提示上述错误。解决的办法是精简语音资源，或者使用外置存储器来保存语音资源。

◆ **错误提示： SPCE061A.h: No such file or directory**

SPCE061A.h 以及 SPCE061A.inc 是针对 SPCE061A 单片机的硬件定义头文件，unSP IDE v2.0.0 及以上的版本会在创建工程的时候自动地在工程文件夹下生成这两个头文件，供用户程序引用。但较低版本的 IDE 不具备这个功能。如果出现上述错误提示，请手工拷贝这两个文件到工程文件夹下，或者升级 IDE 到 2.0.0 以上版本。