A76xx Series OpenSDK_ Audio_调试指导

LTE 模组

芯讯通无线科技(上海)有限公司
上海市长宁区临虹路289号3号楼芯讯通总部大楼
电话：86-21-31575100
技术支持邮箱：support@simcom.com
官网：www.simcom.com

名称：	A76xx Series Open SDK_Audio_调试指导
版本：	V1.00
类别：	应用文档
状态：	已发布

版权声明

芯讯通无线科技(上海)有限公司
上海市长宁区临虹路289号3号楼芯讯通总部大楼
电话：86-21-31575100
邮箱：simcom@simcom.com
官网：www.simcom.com

了解更多资料，请点击以下链接：
http://cn.simcom.com/download/list-230-cn.html

技术支持，请点击以下链接：
http://cn.simcom.com/ask/index-cn.html 或发送邮件至 support@simcom.com

Version History

Version	Date	Owner	What is new
V1.00	2022-11-22		第一版

About this Document

本文档适用于 A1803S open 系列、A1603 open 系列、A1606 open 系列。

目录

缩略语

PCM	Pulse Code Modulation
MP3	Moving Picture Experts Group Audio Layer III
WAV	Waveform Audio File Format
AMR	Adaptive Multi-Rate

1 音频系统概述

Crane 参考设计的音频系统包括如下功能：

1) 通过控制模拟外设，进行打开和关闭外设，从而进行设置音量、静音等功能；
2) 设置通讯语音功能（下文称为 voice 功能），包括 2G/3G CS call、4G VoLTE call、以及通话中的 PCM streaming 功能
3) 多媒体音频功能（下文称为 audio 功能），包括对 MP3/AMR 等文件的编解码功能。

下面将通过对硬件和软件框架结构进行描述：

1.1 音频系统硬件框架

在 Crane 参考设计上，音频系统的硬件组成包括以下几个部分：

1) 模拟音频外设，如麦克风和耳机；
2) Audio Codec： 音频数字通路（DFE），audio DAC/ADC、PGA，Headphone driver，RCV driver 等；
3) CP（ARM Cortex R5）、DSP（CEVA X1）用于处理基带通讯和控制，包括音频编解码和语音增强。

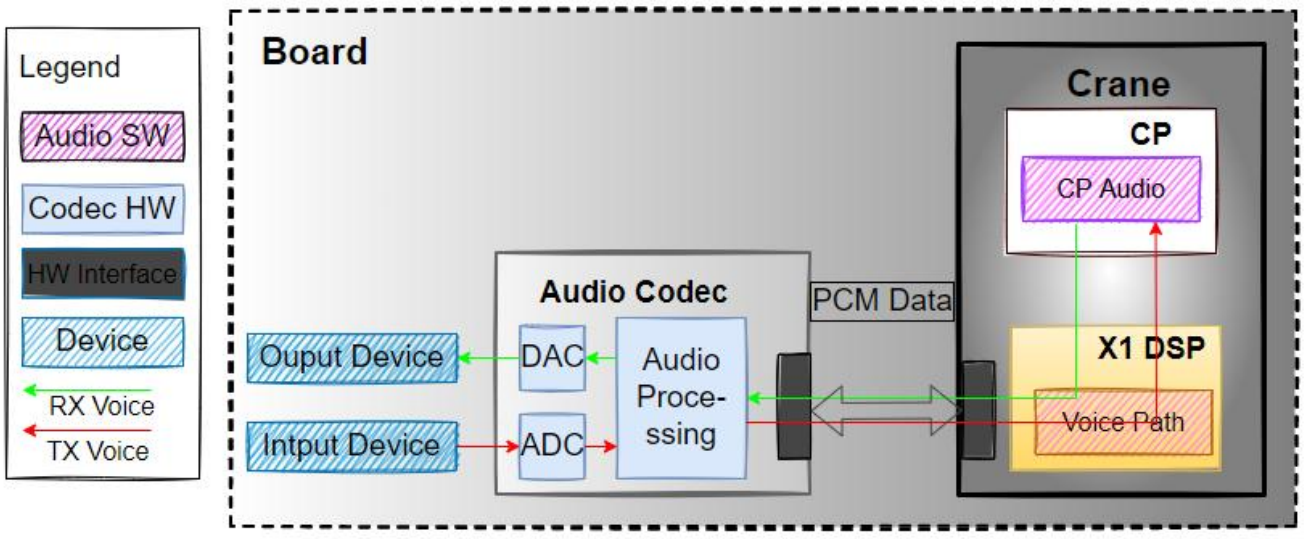

图 2-1 音频系统的硬件组成部件及其连接。

1.2 音频系统软件框架

下图描述了 Crane 的音频系统软件框架：

1) Platform：提供 DIAG、NVM、FAT 等服务；
2) Mini GUI：提供多媒体播放、录音等功能，以及多媒体编解码库；
3) CP Audio：提供与 GUI 接口的 Audio HAL，与 PS/DSP 接口的 AuC 以及音频校准模块；
4) X1 Audio：提供电话相关的编解码，完成语音增强、操作 audio codec。

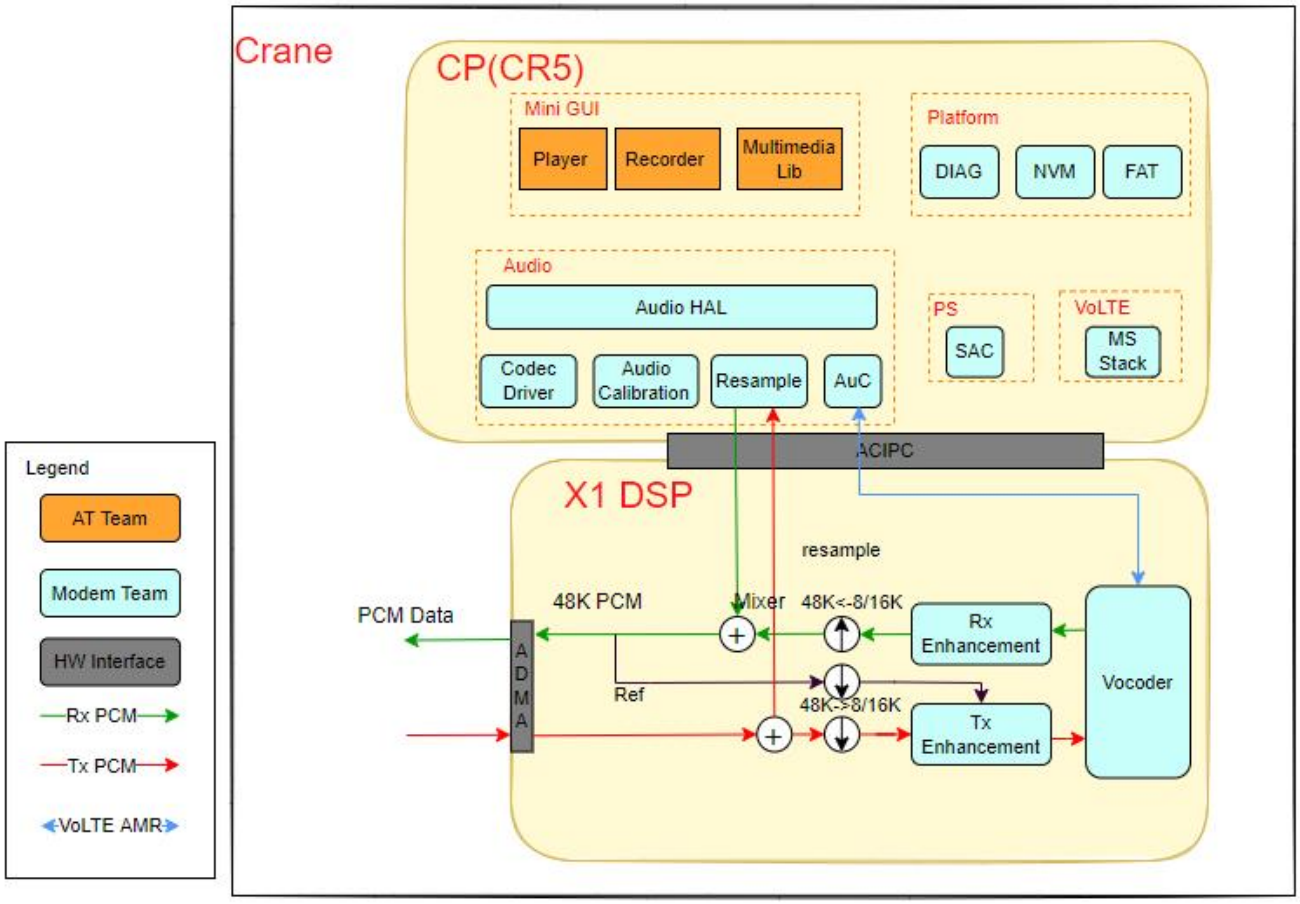

图 2.2 音频系统软件框架

1.3 音频系统采样率配置

由于音频系统涉及多个硬件和软件模块，根据应用场景不同，采样率可能不一样，如图 2-3 所示， Crane 平台的采样率配置如下：

1) Crane codec 的 DFE（Digital Front End，数值前端）采用 48KHz，不论是 voice 还是 audio 场景；
2) DSP 的 voice enhancement 采用 8/16KHz，根据 vocoder 配置确定是 8KHz 还是 16KHz；
3) DSP 和 CP 的录音 PCM 数据采用 16KHz，不论是 voice 还是 audio 场景；
4) CP 和 DSP 的播放 PCM 数据采用 48KHz

如果 MMI 配置 Audio HAL 采用不同的录音采样率，Audio HAL 完成重采样后再写入 DSP 提供的地址。

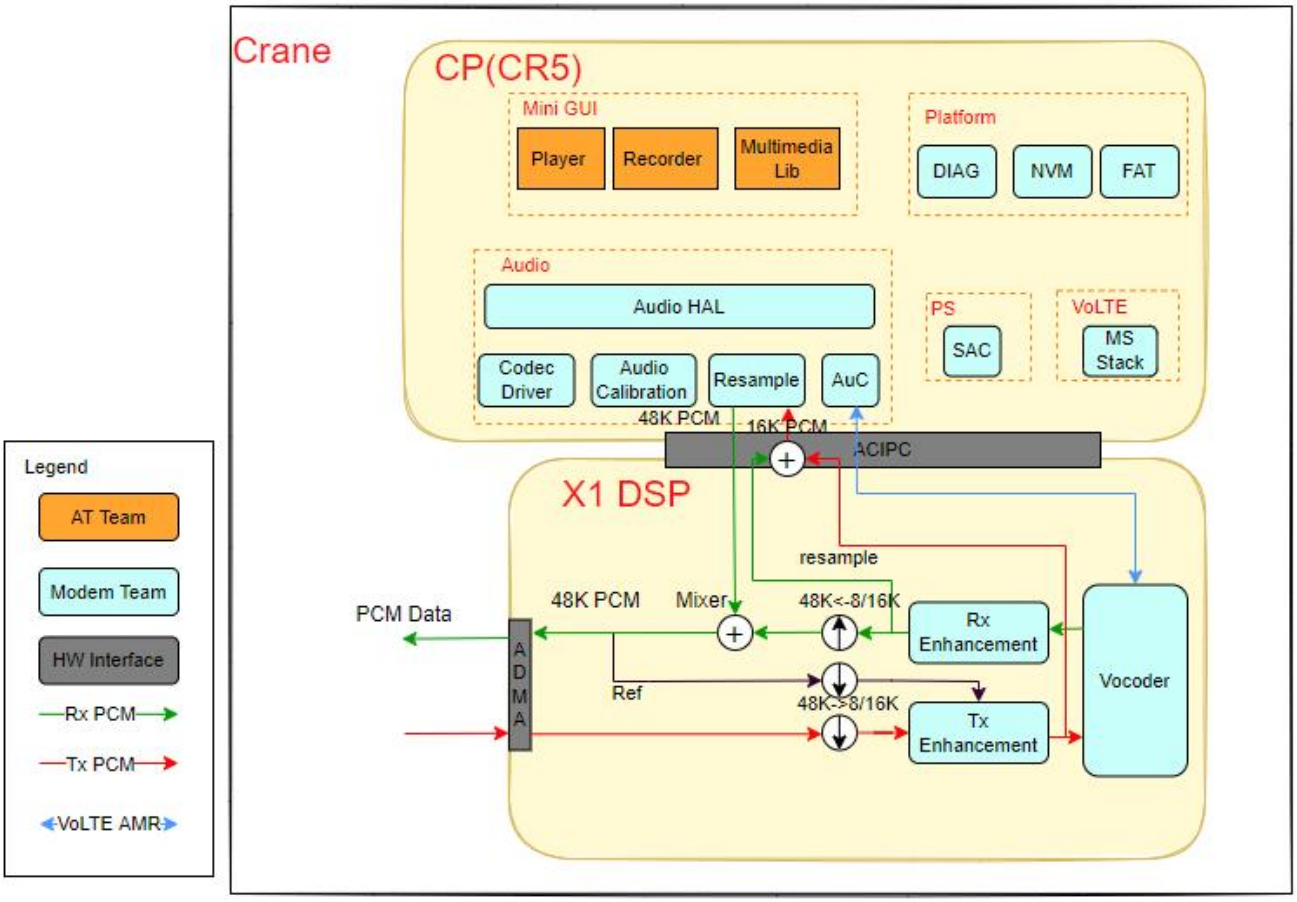
Crane
CP(CR5)
Mini GUI
Player
Recorder
Multimedia Lib
Platform
DIAG
NVM
FAT
Audio
Audio HAL
Codec Driver
Audio Calibration
Resample
AuC
PS
SAC
VoLTE
MS Stack
48K PCM
16K PCM
ACIPC
X1 DSP
resample
PCM Data
48K PCM
Mixer
48K<-8/16K
Rx Enhancement
Vocoder
ADMA
Ref
48K->8/16K
Tx Enhancement
Legend
AT Team
Modem Team
HW Interface
Rx PCM
Tx PCM
VoLTE AMR

图 2.3 采样率配置

2Audio HAL

如图 2-2 所示音频系统软件框架，Audio HAL（Hardware Adaption Layer）为上层多媒体接口（MMI，Multi-Media Interface）提供操作底层硬件的接口，诸如打开/关闭外设，设置音量，静音，DTMF 按键音，多媒体录音/播放等功能。

控制:

Audio HAL 将高层的命令分解成具体的控制，并且下发给底层模块（codec 和 DSP）。

数据:

Audio HAL 接收来自 DSP 的录音数据（可能需要重采样），写入 MMI 提供的地址；

Audio HAL 把 MMI 提供的播放数据进行重采样，写入 DSP 提供的地址。

3Audio API 介绍

头文件：#include “simcom_audio.h”

3.1 Audio 通用 API 接口

3.1.1 sAPI_AudioPlaySampleRate

此应用程序接口是设置音频播放的采样率。

接口:	void **sAPI_AudioPlaySampleRate**(AUD_SampleRate rate);
输入:	**rate：** 采样率 **SAMPLE_RATE_8K:** 采样速率设置为 8K **SAMPLE_RATE_16K:** 采样速率设置为 16K
输出:	无
返回值:	None
NOTE:	无

3.1.2 sAPI_AudioPlay

此应用程序接口是用来播放音频文件的。

接口:	BOOL **sAPI_AudioPlay**(char *file, BOOL direct, BOOL isSingle);
输入:	**file：** 文件路径和文件名，支持 PCM, WAV, AMR, MP3 **drect：** 是否直接播放。(如果设置为“True”，则停止当前播放，播放新的 PCM 数据) **isSingle:** 是否播放单个文件("FALSE"表示在一行中播放多个文件。)
输出:	无
返回值:	True: 播放成功。

	False: 播放失败。
NOTE:	无

3.1.3 sAPI_AudioPlayMp3Cont

此应用程序接口是读取 Mp3 进行连续播放。

接口:	BOOL **sAPI_AudioPlayMp3Cont** (char *file, BOOL startplay, BOOL frame);
输入:	**file：** 文件路径和文件名，支持 MP3; **startplay：** 是否开始播放。(如果设置为“True”，则继续播放 mp3 文件); **frame：** 删除 mp3 文件的前几帧(删除的帧数从 0 到 2);
输出:	无
返回值:	True: 播放或加载成功。 False: 播放或加载失败。
NOTE:	无

3.1.4 sAPI_AudioStop

此应用程序接口是停止音频文件的播放。

接口:	BOOL **sAPI_AudioStop**(void);
输入:	无
输出:	无
返回值:	True: 停止播放成功。 False: 停止播放失败。
NOTE:	无

3.1.5 sAPI_AudioRecord

此应用程序接口是用来录制音频文件的。

接口:	BOOL **sAPI_AudioRecord**(BOOL enable, AUD_RecordSrcPath path, char *file);
输入:	**enable:** 是否录制; 0 停止录制 1 开始录制 **path:** 录制路径，REC_PATH_LOCAL 为录制本地音频 **file:** 文件的路径和名称;只能保存到“C:/”，只支持 WAV 和 AMR 格式
输出:	无
返回值:	True: 开始录音 False: 录音失败
NOTE:	无

3.1.6 sAPI_AudioPcmPlay

此应用程序接口用于直接播放 PCM 流。

接口:	BOOL **sAPI_AudioPcmPlay**(char *buffer, UINT32 len, BOOL direct);
输入:	**buffer:** PCM 流缓冲区 **len:** PCM 流缓冲区的长度 **direct:** 是否直接播放（如果设置为“True”，则停止当前播放，播放新的 PCM 数据)
输出:	无
返回值:	True: 播放成功 False: 播放失败
NOTE:	无

3.1.7 sAPI_AudioPcmStop

此应用程序接口用于停止 PCM 流。

接口:	BOOL **sAPI_AudioPcmStop**(void);
输入:	无

输出:	无
返回值:	True: 停止播放成功 False: 停止播放失败
NOTE:	无

3.1.8 sAPI_AudioSetVolume

此应用程序接口用于设置系统音量。

接口:	void **sAPI_AudioSetVolume**(AUD_Volume volume);
输入:	**volume:** 系统音量，取值范围为 **0-11**。
输出:	无
返回值:	无
NOTE:	无

3.1.9 sAPI_AudioGetVolume

此应用程序接口用于获取音量大小。

接口:	AUD_Volume **sAPI_AudioGetVolume**(void);
输入:	无
输出:	无
返回值:	音量数值等级；
NOTE:	无

3.1.10 sAPI_AudioStatus

此应用程序接口用于获取音频状态。

接口:	UINT8 **sAPI_AudioStatus**(void);
输入:	无
输出:	无
返回值:	0: 音频状态空闲 1: 音频状态繁忙
NOTE:	无

3.1.11 sAPI_AudRec

此应用程序接口用于设置录音时长。

接口:	BOOL **sAPI_AudRec**(UINT8 oper, char *file, UINT8 duration, sAPI_AudRecCallback callback);
输入:	**oper:** 录音操作，取值为: 1 开始 0 停止 **file:** 文件名和路径，如 C:/test.wav **duration:** 设置录音时长；wav 音频文件的录音范围为 1 ~ 15s，amr 音频文件的录音时长为 1 ~ 180s **callback:** 获取录音状态的回调函数
输出:	无
返回值:	0: 开始录音 1: 录音失败
NOTE:	无

3.1.12 sAPI_AudioSetAmrEncodeRate

此应用程序接口是设置使用 AMR 文件进行录音的编码率(这个 api 只适用 **sAPI_AudioRecord**)。

接口:	void **sAPI_AudioSetAmrEncodeRate**(AudioAmrEncodeRate rate);
输入:	**rate:** 设置 AMR 编码速率；默认为 REC_AMR_MR122 typedef enum {

	REC_AMR_MR475, REC_AMR_MR515, REC_AMR_MR59, REC_AMR_MR67, REC_AMR_MR74, REC_AMR_MR795, REC_AMR_MR102, REC_AMR_MR122, } AudioAmrEncodeRate; //编码率数据
输出:	无
返回值:	None
NOTE:	无

3.1.13 sAPI_AudioSetMicGain

此应用程序接口是设置麦克风增益。

接口:	void **sAPI_AudioSetMicGain**(UINT32 micgain);
输入:	**micgain**： 麦克风增益范围 0-7。
输出:	无
返回值:	None
NOTE:	无

3.1.14 sAPI_AudioGetMicGain

此应用程序接口是用来获取麦克风增益。

接口:	int **sAPI_AudioGetMicGain**(void);
输入:	无
输出:	无
返回值:	麦克风增益等级

NOTE:	无

3.1.15 sAPI_AudioMp3StreamPlay

此应用程序接口用于播放 mp3 缓冲区数据，仅适用于 1603 OpenSDK。

接口:	BOOL **sAPI_AudioMp3StreamPlay**(char *buffer, UINT32 len, BOOL direct);
输入:	**buffer**： MP3 流缓冲区 **len**： MP3 流缓冲区的长度 **direct**： 是否直接播放 (如果设置为“True”，则停止当前播放，播放新的 MP3 数据)
输出:	无
返回值:	True: 播放成功 False: 播放失败
NOTE:	无

3.1.16 sAPI_AudioMp3StreamStop

此应用程序接口用于停止 mp3 缓冲区数据播放，仅适用于 1603 OpenSDK。

接口:	BOOL **sAPI_AudioMp3StreamStop**(void);
输入:	无
输出:	无
返回值:	True: 停止播放成功 False: 停止播放失败
NOTE:	无

3.1.17 sAPI_AudioAmrStreamPlay

此应用程序接口用于播放 AMR 缓冲区数据，仅适用于 1603 OpenSDK。

接口:	BOOL **sAPI_AudioAmrStreamPlay**(char *buffer, UINT32 len, BOOL direct);
输入:	**buffer** ： AMR 流缓冲区。 **len**： AMR 流缓冲区长度。 **direct**： 是否直接播放。(如果设置为“True”，则停止当前播放，播放新的 AMR 数据)
输出:	无
返回值:	True: 播放 AMR 数据成功 False: 播放 AMR 数据失败
NOTE:	无

3.1.18 sAPI_AudioAmrStreamStop

此应用程序接口用于停止播放 AMR 缓冲区数据，仅适用于 1603 OpenSDK。

接口:	BOOL **sAPI_AudioAmrStreamStop**(void);
输入:	无
输出:	无
返回值:	True: 停止播放 AMR 数据成功 False: 停止播放 AMR 数据失败
NOTE:	无

3.1.19 sAPI_AudioPlayAmrCont

此应用程序接口是读取 AMR 进行连续播放。

接口:	BOOL **sAPI_AudioPlayAmrCont** (char *file, BOOL startplay);
输入:	**file** ： AMR 文件的路径和名称 **startplay**：是否开始播放(如果设置为“True”，开始播放 AMR 文件列表);
输出:	无
返回值:	True: 播放或加载成功 False: 播放或加载失败

NOTE:	无

3.1.20 sAPI_AudioWavFilePlay

此应用程序接口用于播放除 8k 和 16k 采样率之外的其他 wav 文件。

接口:	BOOL **sAPI_AudioWavFilePlay** (char *file, BOOL direct);
输入:	**file** ： wav 文件的路径和名称 **direct**：是否直接播放 (如果设置为“True”，停止当前播放，播放新的文件);
输出:	无
返回值:	True: 播放 wav 文件成功 False: 播放 wav 文件失败
NOTE:	无

3.1.21 sAPI_AudioSetPlayPath

此应用程序接口用于设置播放 path 为 本地 或 远程。

接口:	BOOL **sAPI_AudioSetPlayPath**(UINT8 path);
输入:	path：播放路径； 0 本地 1 远程
输出:	无
返回值:	true : 设置成功 false: 设置失败
NOTE:	无

3.1.22 sAPI_AudioGetPlayPath

此应用程序接口用于获取播放 path 的 value。

接口:	UINT8 **sAPI_AudioGetPlayPath** (void);
输入:	无
输出:	无
返回值:	0 本地 1 远程
NOTE:	无

3.2 Audio Poc 函数接口

3.2.1 sAPI_PocInitLib

此应用程序接口是用于初始化 Poc 函数。

接口:	int **sAPI_PocInitLib**(sAPI_NotifyBufStateCb callback);
输入:	**callback:** 注册回调函数来报告剩余数据
输出:	无
返回值:	0 : 初始化成功 -1: 初始化失败
NOTE:	无

3.2.2 sAPI_PocPlaySound

此应用程序接口是用来播放 poc PCM 缓冲区数据。

接口:	int **sAPI_PocPlaySound**(const char *data, int length);
输入:	**data ： PCM 数据缓冲区** **length： PCM 数据长度**
输出:	无
返回值:	**0 :** 播放成功. **-1:** 播放失败
NOTE:	无

3.2.3 sAPI_PocStopSound

此应用程序接口用于停止播放 poc PCM 缓冲区数据。

接口:	int **sAPI_PocStopSound**(void);
输入:	无
输出:	无
返回值:	0： 停止播放成功 -1: 停止播放失败
NOTE:	无

3.2.4 sAPI_PocGetPcmAvail

此应用程序接口用于获取可用的 Poc 数据空闲大小。

接口:	int **sAPI_PocGetPcmAvail**(void);
输入:	无
输出:	无
返回值:	剩余缓存的大小
NOTE:	无

3.2.5 sAPI_PocCleanBufferData

此应用程序接口用于清理缓冲区数据。

接口:	int **sAPI_PocCleanBufferData**(void);
输入:	无
输出:	无
返回值:	**0**： 清除成功 **-1:** 清除失败

NOTE:	无

3.2.6 sAPI_PocStartRecord

此应用程序接口用于开始录音。

接口:	int **sAPI_PocStartRecord**(sAPI_ProcessRecordCb callback,int mode);
输入:	**callback**：注册回调函数来报告记录数据 **mode** ： 主动或被动报告数据 1 主动 2 被动
输出:	无
返回值:	0： 录音开始 -1: 录音失败
NOTE:	无

3.2.7 sAPI_PocStopRecord

此应用程序接口用于停止录音。

接口:	int **sAPI_PocStopRecord**(void);
输入:	无
输出:	无
返回值:	0： 停止录音设置成功 -1: 停止录音设置失败
NOTE:	无

3.2.8 sAPI_PocPcmRead

此应用程序接口用于读取录音数据。

接口:	int **sAPI_PocPcmRead**(char *buffer,int dataLen);
输入:	**dataLen**: 只能是 320(8k16 比特)

输出:	**buffer** : 录音数据缓冲区
返回值:	0 : 读取录音数据成功 -1: 读取录音数据失败
NOTE:	无

4Audio 数据定义

4.1 AUD_SampleRate

```
typedef enum
{
    SAMPLE_RATE_8K = 0,
    SAMPLE_RATE_16K,
    SAMPLE_RATE_MAX
}AUD_SampleRate;    // Audio 采样率设置
```

4.2 AUD_Volume

```
typedef enum
{
    AUDIO_VOLUME_MUTE = 0,
    AUDIO_VOLUME_1,
    AUDIO_VOLUME_2,
    AUDIO_VOLUME_3,
    AUDIO_VOLUME_4,
    AUDIO_VOLUME_5,
    AUDIO_VOLUME_6,
    AUDIO_VOLUME_7,
    AUDIO_VOLUME_8,
    AUDIO_VOLUME_9,
    AUDIO_VOLUME_10,
    AUDIO_VOLUME_11,
} AUD_Volume;          // Audio 音量调节设置
```

4.3 AUD_RecordSrcPath

```
typedef enum
{
    REC_PATH_LOCAL = 0,    //本地录制
    REC_PATH_MAX
}AUD_RecordSrcPath;   //录制路径信息
```

5Audio Demo

```
#include “simcom_audio.h”

void AudioDemo(void)
{
    SIM_MSG_T optionMsg ={0,0,0,NULL};
    char audioFile[26] = {0};
    char fileNameInfo[MAX_VALID_USER_NAME_LENGTH] = {0};
    SCFILE* fh = NULL;
    static char *buffer = NULL;
    char tempBuffer[64];
    char recording_file[26] = {0};
    int len, opt = 0;
    AUD_SampleRate sampleRate = 0;
    AUD_Volume volume;
    BOOL directPlay = 0;
    BOOL isSingle = 0;
    BOOL startplay = 0;
    BOOL frame = 0;
    BOOL ret = 0;
    int RateLevel;
    unsigned int gpioNum;
    UINT8 activeLevel;
    char *note = "\r\nPlease select an option to test from the items listed below.\r\n";
    char *optionsList[] = {
        "1. SampleRate",
        "2. Play file",
        "3. Stop play file",
        "4. Record",
        "5. Stop record",
        "6. Play PCM",
        "7. Stop PCM",
        "8. Set volume",
        "9. Get volume",
        "10. slxk record",
        "11, Play MP3 cont",
        "12. Set micgain",
        "13. Get micgain",
        "14. play mp3 buffer",
        "15, stop MP3 buffer",
```

```
        "16. PA control config",
        "17. play amr buffer",
        "18, stop amr buffer",
        "19, Play amr cont",
        "20, Play WAV with high sampling rate",
        "21. get amr record frame",
        "22. stop amr frame",
        "23. set amr encoder rate",
        "24. Play Wav cont",
        "25. EchoSuppressionParameter",
        "26. EchoParaModeSet",
        "99. back",
    };

    buffer = sAPI_Malloc(100*1024);

    while(1)
    {
        PrintfResp(note);
        PrintfOptionMenu(optionsList,sizeof(optionsList)/sizeof(optionsList[0]));
        sAPI_MsgQRecv(simcomUI_msgq,&optionMsg,SC_SUSPEND);
        if(SRV_UART != optionMsg.msg_id)
        {
            printf("%s,msg_id is error!!",__func__);
            break;
        }

        printf("arg3 = [%s]",(char *)optionMsg.arg3);
        opt = atoi(optionMsg.arg3);
        sAPI_Free(optionMsg.arg3);

        switch(opt)
        {
            case SC_AUDIO_DEMO_SET_SAMPLE_RATE:
            {
                PrintfResp("\r\nPlease input sample rate, 0(8K) or 1(16K)\r\n");
                optionMsg = GetParamFromUart();
                sampleRate = atoi(optionMsg.arg3);
                sAPI_Free(optionMsg.arg3);

                ret = sAPI_AudioPlaySampleRate(sampleRate);
                if(ret)
                   printf("sAPI_AudioPlaySampleRate: %d success !", sampleRate);
                else
                   printf("sAPI_AudioPlaySampleRate: %d failed !", sampleRate);
```

```
        break;
    }

    case SC_AUDIO_DEMO_PLAY_FILE:
    {
        UINT8 path;
        PrintfResp("\r\nPlease input File Name, like c:/test.wav\r\n");
        optionMsg = GetParamFromUart();
        memset(audioFile, 0, sizeof(audioFile));
        strcpy(audioFile, optionMsg.arg3);
        sAPI_Free(optionMsg.arg3);

        PrintfResp("\r\nPlease input direct or not, 1 or 0\r\n");
        optionMsg = GetParamFromUart();
        directPlay = atoi(optionMsg.arg3);
        sAPI_Free(optionMsg.arg3);

        PrintfResp("\r\nPlease input is single file palyback, 1 or 0\r\n");
        optionMsg = GetParamFromUart();
        isSingle = atoi(optionMsg.arg3);
        sAPI_Free(optionMsg.arg3);

        PrintfResp("\r\nPlease input playback path, 1(remote) or 0(local)\r\n");
        optionMsg = GetParamFromUart();
        path = atoi(optionMsg.arg3);
        sAPI_Free(optionMsg.arg3);
        ret = sAPI_AudioSetPlayPath(path);
        printf("sAPI_AudioSetPlayPath: play path set result %d ",ret);

        ret = sAPI_AudioPlay(audioFile, directPlay,isSingle);
        if(ret)
          printf("sAPI_AudioPlay: playing %s success ", audioFile);
        else
          printf("sAPI_AudioPlay: playing %s failed", audioFile);
        sAPI_TaskSleep(1);
        printf("sAPI_AudioStatus: %d", sAPI_AudioStatus());

        break;
    }

    case SC_AUDIO_DEMO_STOP_PLAY_FILE:
    {
        ret = sAPI_AudioStop();
        if(ret)
          printf("sAPI_AudioStop: stop %s success ", audioFile);
        else
```

```
            printf("sAPI_AudioStop: stop %s failed", audioFile);
        sAPI_TaskSleep(1);
        printf("sAPI_AudioStatus: %d", sAPI_AudioStatus());

        break;
    }

    case SC_AUDIO_DEMO_RECORD:
    {
        PrintfResp("\r\nPlease input File Name.\r\n");
        optionMsg = GetParamFromUart();
        strcpy(recording_file,optionMsg.arg3);
        sAPI_Free(optionMsg.arg3);

        ret = sAPI_AudioRecord(1, REC_PATH_LOCAL, recording_file);
        if(ret)
            printf("sAPI_AudioRecord: record to %s success ",recording_file);
        else
            printf("sAPI_AudioRecord: record to %s failed",recording_file);
        sAPI_TaskSleep(1);
        printf("sAPI_AudioStatus: %d", sAPI_AudioStatus());

        break;
    }

    case SC_AUDIO_DEMO_RECORD_STOP:
    {
        PrintfResp("\r\nPlease input File Name.\r\n");
        optionMsg = GetParamFromUart();
        strcpy(recording_file,optionMsg.arg3);
        sAPI_Free(optionMsg.arg3);

        ret = sAPI_AudioRecord(0, REC_PATH_LOCAL, recording_file);
        if(ret)
            printf("sAPI_AudioRecord: stop success ");
        else
            printf("sAPI_AudioRecord: stop failed");
        sAPI_TaskSleep(1);
        printf("sAPI_AudioStatus: %d", sAPI_AudioStatus());
        break;
    }

    case SC_AUDIO_DEMO_PLAY_PCM:
    {
        UINT8 path;
        PrintfResp("\r\nPlease input File Name to test, like test.wav\r\n");
```

```
        optionMsg = GetParamFromUart();
        memset(audioFile, 0, sizeof(audioFile));
        strcpy(audioFile,optionMsg.arg3);
        sAPI_Free(optionMsg.arg3);

        PrintfResp("\r\nPlease input direct or not, 1 or 0\r\n");
        optionMsg = GetParamFromUart();
        directPlay = atoi(optionMsg.arg3);
        sAPI_Free(optionMsg.arg3);
        memset(fileNameInfo,0,sizeof(fileNameInfo));
        memcpy(fileNameInfo, "c:/", strlen("c:/"));
        strcat(fileNameInfo, audioFile);

        printf("fileNameInfo[%s]",fileNameInfo);

        PrintfResp("\r\nPlease input playback path, 1(remote) or 0(local)\r\n");
        optionMsg = GetParamFromUart();
        path = atoi(optionMsg.arg3);
        sAPI_Free(optionMsg.arg3);
        ret = sAPI_AudioSetPlayPath(path);
        printf("sAPI_AudioSetPlayPath: play path set result %d ",ret);

        fh = sAPI_fopen(fileNameInfo, "rb");
        //buffer = sAPI_Malloc(100*1024);
        if(buffer == NULL && fh != NULL)
        {
            printf("sAPI_Malloc fail !");
        }
        else
        {
            len = sAPI_fread(buffer, 1,100*1024, fh);
            printf("sAPI_FsFileRead len:%d !", len);
            if(len > 44)
            {
                ret = sAPI_AudioPcmPlay(&buffer[44], (len - 44), directPlay);
                printf("sAPI_AudioPcmPlay ret:%d !", ret);
            }
        }
        sAPI_fclose(fh);
        sAPI_TaskSleep(1);
        printf("sAPI_AudioStatus: %d", sAPI_AudioStatus());
        break;
    }

    case SC_AUDIO_DEMO_STOP_PLAY_PCM:
    {
```

```
        PrintfResp("\r\nPlease input any key.\r\n");
        optionMsg = GetParamFromUart();
        sAPI_Free(optionMsg.arg3);

        ret = sAPI_AudioPcmStop();
        if(ret)
          printf("sAPI_AudioPcmStop: stop success !");
        else
          printf("sAPI_AudioPcmStop: stop failed !");
        sAPI_TaskSleep(1);
        printf("sAPI_AudioStatus: %d", sAPI_AudioStatus());
        break;
    }

    case SC_AUDIO_DEMO_SET_VOLUME:
    {
        PrintfResp("\r\nPlease input volume: 0~11.\r\n");
        optionMsg = GetParamFromUart();
        volume = (AUD_Volume)atoi(optionMsg.arg3);
        sAPI_Free(optionMsg.arg3);

        sAPI_AudioSetVolume(volume);
        printf("sAPI_AudioSetVolume !");
        break;
    }

    case SC_AUDIO_DEMO_GET_VOLUME:
    {
        volume = sAPI_AudioGetVolume();
        snprintf(tempBuffer, sizeof(tempBuffer), "\r\nvolume = %d\r\n", volume);

        PrintfResp(tempBuffer);
        printf("sAPI_AudioGetVolume: %d !", volume);
        break;
    }

    case SC_AUDIO_DEMO_CUS_AUDREC:
    {
        int duration,oper;
        PrintfResp("\r\nPlease input oper, 1(start record)/0(stop record)\r\n");
        optionMsg = GetParamFromUart();
        oper = atoi(optionMsg.arg3);
        sAPI_Free(optionMsg.arg3);

        PrintfResp("\r\nPlease input file name, like C:/test.amr\r\n");
        optionMsg = GetParamFromUart();
```

```
        memset(audioFile, 0, sizeof(audioFile));
        strcpy(audioFile, optionMsg.arg3);
        sAPI_Free(optionMsg.arg3);

        PrintfResp("\r\nPlease input record time amr(1-180s),wav/pcm(1-15s)\r\n");
        optionMsg = GetParamFromUart();
        duration = atoi(optionMsg.arg3);
        sAPI_Free(optionMsg.arg3);

        ret = sAPI_AudRec(oper, audioFile,duration,recordCallBac_Test);
        if(ret)
          printf("sAPI_AudRec: %d operate success", ret);
        else
          printf("sAPI_AudRec: %d operate failed", ret);
        sAPI_TaskSleep(1);
        printf("sAPI_AudioStatus: %d", sAPI_AudioStatus());
        break;
    }

    case SC_AUDIO_DEMO_PLAY_MP3_CONT:
    {
        do
        {
            PrintfResp("\r\nPlease input File Name, like c:/test.mp3\r\n");
            optionMsg = GetParamFromUart();
            memset(audioFile, 0, sizeof(audioFile));
            strcpy(audioFile, optionMsg.arg3);
            sAPI_Free(optionMsg.arg3);

            PrintfResp("\r\nPlease input startplay or not, 1 or 0\r\n");
            optionMsg = GetParamFromUart();
            startplay = atoi(optionMsg.arg3);
            sAPI_Free(optionMsg.arg3);

            PrintfResp("\r\nPlease input is frame, 0 ~ 2\r\n");
            optionMsg = GetParamFromUart();
            frame = atoi(optionMsg.arg3);
            sAPI_Free(optionMsg.arg3);

            ret = sAPI_AudioPlayMp3Cont(audioFile, startplay,frame);
            if(ret)
              printf("sAPI_AudioPlayMp3Cont: load %s success ", audioFile);
            else
              printf("sAPI_AudioPlayMp3Cont: load %s failed", audioFile);

        }while(!startplay);
```

```
        printf("sAPI_AudioPlayMp3Cont%s: start play!", audioFile);
        sAPI_TaskSleep(1);
        printf("sAPI_AudioStatus: %d", sAPI_AudioStatus());
        break;
    }

    case SC_AUDIO_DEMO_SET_MICGAIN:
    {
        int micgain;
        PrintfResp("\r\nPlease input micgain(0-7)\r\n");
        optionMsg = GetParamFromUart();
        micgain = atoi(optionMsg.arg3);
        sAPI_Free(optionMsg.arg3);
        if(micgain < 0 || micgain > 7)
        {
            PrintfResp("\r\nparameter error\r\n");
            break;
        }
        sAPI_AudioSetMicGain(micgain);
        printf("sAPI_AudioSetMicGain !");
        PrintfResp("\r\nset parameter OK\r\n");
        break;
    }

    case SC_AUDIO_DEMO_GET_MICGAIN:
    {
        int ret = sAPI_AudioGetMicGain();
        printf("sAPI_AudioGetMicGain ret = %d !", ret);
        break;
    }
    case SC_AUDIO_DEMO_PLAY_MP3_BUFFER:
    {
        PrintfResp("\r\nPlease input File Name to test, like c:/test.mp3\r\n");
        optionMsg = GetParamFromUart();
        memset(audioFile, 0, sizeof(audioFile));
        strcpy(audioFile,optionMsg.arg3);
        sAPI_Free(optionMsg.arg3);

        PrintfResp("\r\nPlease input direct or not, 1 or 0\r\n");
        optionMsg = GetParamFromUart();
        directPlay = atoi(optionMsg.arg3);
        sAPI_Free(optionMsg.arg3);

        memcpy(fileNameInfo, audioFile, strlen(audioFile));
        printf("fileNameInfo[%s]",fileNameInfo);
```

```
        fh = sAPI_fopen(fileNameInfo, "rb");
        //buffer = sAPI_Malloc(100*1024);
        if(buffer == NULL || fh == NULL)
        {
            printf("sAPI_Malloc fail !");
        }
        else
        {
            len = sAPI_fread(buffer, 1,100*1024, fh);
            printf("sAPI_FsFileRead len:%d !", len);
            if(len > 0)
            {
                ret = sAPI_AudioMp3StreamPlay(buffer, len, directPlay);
                printf("sAPI_AudioMp3StreamPlay ret:%d !", ret);
            }
        }
        sAPI_fclose(fh);
        sAPI_TaskSleep(1);
        printf("sAPI_AudioStatus: %d", sAPI_AudioStatus());
        break;
    }

    case SC_AUDIO_DEMO_STOP_MP3_BUFFER:
    {
        ret = sAPI_AudioMp3StreamStop();
        printf("sAPI_AudioMp3StreamStop ret:%d !", ret);
        sAPI_TaskSleep(1);
        printf("sAPI_AudioStatus: %d", sAPI_AudioStatus());
        break;
    }

    case SC_AUDIO_DEMO_PA_CONFIG:
    {
        PrintfResp("\r\nPlease input gpio number.\r\n");
        optionMsg = GetParamFromUart();
        gpioNum = atoi(optionMsg.arg3);
        sAPI_Free(optionMsg.arg3);

        PrintfResp("\r\nPlease input active level.      0:low    1:high.\r\n");
        optionMsg = GetParamFromUart();
        activeLevel = atoi(optionMsg.arg3);
        sAPI_Free(optionMsg.arg3);

        ret = sAPI_AudioSetPaCtrlConfig(gpioNum, activeLevel);
        if(ret != SC_GPIORC_OK)
```

```
            printf("sAPI_AudioSetPaCtrlConfig failed.");
        else
            printf("sAPI_AudioSetPaCtrlConfig success.");

        PrintfResp("\r\noperation successful!\r\n");

        break;
    }
    case SC_AUDIO_DEMO_PLAY_AMR_BUFFER:
    {
        PrintfResp("\r\nPlease input File Name to test, like c:/test.amr\r\n");
        optionMsg = GetParamFromUart();
        memset(audioFile, 0, sizeof(audioFile));
        strcpy(audioFile,optionMsg.arg3);
        sAPI_Free(optionMsg.arg3);

        PrintfResp("\r\nPlease input direct or not, 1 or 0\r\n");
        optionMsg = GetParamFromUart();
        directPlay = atoi(optionMsg.arg3);
        sAPI_Free(optionMsg.arg3);

        memcpy(fileNameInfo, audioFile, strlen(audioFile));
        printf("fileNameInfo[%s]",fileNameInfo);

        fh = sAPI_fopen(fileNameInfo, "rb");
        //buffer = sAPI_Malloc(100*1024);
        if(buffer == NULL || fh == NULL)
        {
            printf("sAPI_Malloc fail !");
        }
        else
        {
            len = sAPI_fread(buffer, 1,100*1024, fh);
            printf("sAPI_FsFileRead len:%d !", len);
            if(len > 0)
            {
                ret = sAPI_AudioAmrStreamPlay(buffer, len, directPlay);
                printf("sAPI_AudioAmrStreamPlay ret:%d !", ret);
            }
        }
        sAPI_fclose(fh);
        sAPI_TaskSleep(1);
        printf("sAPI_AudioStatus: %d", sAPI_AudioStatus());
        break;
    }
```

```
case SC_AUDIO_DEMO_STOP_AMR_BUFFER:
{
    ret = sAPI_AudioAmrStreamStop();
    printf("sAPI_AudioAmrStreamStop ret:%d !", ret);
    sAPI_TaskSleep(1);
    printf("sAPI_AudioStatus: %d", sAPI_AudioStatus());
    break;
}

case SC_AUDIO_DEMO_PLAY_AMR_CONT:
{
    do
    {
        PrintfResp("\r\nPlease input File Name, like c:/test.amr\r\n");
        optionMsg = GetParamFromUart();
        memset(audioFile, 0, sizeof(audioFile));
        strcpy(audioFile, optionMsg.arg3);
        sAPI_Free(optionMsg.arg3);

        PrintfResp("\r\nPlease input startplay or not, 1 or 0\r\n");
        optionMsg = GetParamFromUart();
        startplay = atoi(optionMsg.arg3);
        sAPI_Free(optionMsg.arg3);

        ret = sAPI_AudioPlayAmrCont(audioFile, startplay);
        if(ret)
          printf("sAPI_AudioPlayAmrCont: load %s success ", audioFile);
        else
          printf("sAPI_AudioPlayAmrCont: load %s failed", audioFile);
    }while(!startplay);

    printf("sAPI_AudioPlayAmrCont %s: start play!", audioFile);
    sAPI_TaskSleep(1);
    printf("sAPI_AudioStatus: %d", sAPI_AudioStatus());
    break;
}

case SC_AUDIO_DEMO_PLAY_WAV_FILE:
{
    PrintfResp("\r\nPlease input File Name, like c:/test.wav\r\n");
    optionMsg = GetParamFromUart();
    memset(audioFile, 0, sizeof(audioFile));
    strcpy(audioFile, optionMsg.arg3);
    sAPI_Free(optionMsg.arg3);

    PrintfResp("\r\nPlease input direct or not, 1 or 0\r\n");
```

```
        optionMsg = GetParamFromUart();
        directPlay = atoi(optionMsg.arg3);
        sAPI_Free(optionMsg.arg3);

        ret = sAPI_AudioWavFilePlay(audioFile, directPlay);
        if(ret)
          printf("sAPI_AudioWavFilePlay: playing %s success ", audioFile);
        else
          printf("sAPI_AudioWavFilePlay: playing %s failed", audioFile);
        sAPI_TaskSleep(1);
        printf("sAPI_AudioStatus: %d", sAPI_AudioStatus());
        break;
    }

    case SC_AUDIO_DEMO_GET_AMR_FRAME:
    {
        ret = sAPI_AmrStreamRecord(GetAmrFrame);
        if(ret)
            printf("sAPI_AmrStreamRecord: start record!");
        else
            printf("sAPI_AmrStreamRecord: fail record!");
        break;
    }

    case SC_AUDIO_DEMO_STOP_AMR_FRAME:
    {
        ret = sAPI_AmrStopStreamRecord();
        if(ret)
            printf("sAPI_AmrStreamRecord: stop record!");
        else
            printf("sAPI_AmrStreamRecord: stop failed!");
        break;
    }

    case SC_AUDIO_DEMO_AMR_ENCODE_RATE:
    {
        PrintfResp("\r\nPlease input is amr encoder rate, 0 ~ 7\r\n");
        optionMsg = GetParamFromUart();
        RateLevel = atoi(optionMsg.arg3);
        sAPI_Free(optionMsg.arg3);

        sAPI_AudioSetAmrRateLevel(RateLevel);
        printf("RateLevel = %d", RateLevel);
        break;
    }
```

```
            case SC_AUDIO_DEMO_PLAY_WAV_CONT:
            {
                do
                {
                    PrintfResp("\r\nPlease input File Name, like c:/test.WAV\r\n");
                    optionMsg = GetParamFromUart();
                    memset(audioFile, 0, sizeof(audioFile));
                    strcpy(audioFile, optionMsg.arg3);
                    sAPI_Free(optionMsg.arg3);

                    PrintfResp("\r\nPlease input startplay or not, 1 or 0\r\n");
                    optionMsg = GetParamFromUart();
                    startplay = atoi(optionMsg.arg3);
                    sAPI_Free(optionMsg.arg3);

                    ret = sAPI_AudioPlayWavCont(audioFile, startplay);
                    if(ret)
                      printf("sAPI_AudioPlayWavCont: load %s success ", audioFile);
                    else
                      printf("sAPI_AudioPlayWavCont: load %s failed", audioFile);
                }while(!startplay);

                printf("sAPI_AudioPlayWavCont%s: start play!", audioFile);
                sAPI_TaskSleep(1);
                printf("sAPI_AudioStatus: %d", sAPI_AudioStatus());
                break;
            }
            case SC_AUDIO_DEMO_MAX:
            {
                return;
            }
        }
    }
}
```