Android 15 音频
开发指南

版本号: 1.0
发布日期: 2025.03.04

版本历史

版本号	日期	制/修订人	内容描述
1.0	2025.03.04	AW2183	初始版本

目　录

插　图

1 前言

1.1 文档简介

本文档目的是让开发者了解全志安卓音频系统框架，能够在全志平台上开发新的音频方案。

1.2 目标读者

音频系统开发人员。

1.3 适用范围

本模块说明适用于 A537/A333 平台。

1.4 相关术语介绍

术语	说明
ALSA	高级 Linux 声音体系（Advanced Linux Sound Architecture），是 Linux 内核中，为声卡提供的驱动组件，以替代原先的 OSS（开放声音系统）
DMA	直接内存存取（Direct Memory Access），指数据不经 cpu,，直接在设备和内存，内存和内存，设备和设备之间传输
OSS	开放声音系统（Open Sound System）
sample	样本长度，样本是记录音频数据最基本的单位，常见的有 8 位和 16 位
channel	通道数，该参数为 1 表示单声道，2 则是立体声
frame	帧，帧记录了一个声音单元，其长度为样本长度与通道数的乘积
rate	采样率，每秒钟采样次数，该次数是针对帧而言
period	周期，音频设备一次处理所需要的帧数，对于音频设备的数据访问以及音频数据的存储，都是以此为单位

术语	说明
interleave	交错模式，是一种音频数据的记录模式，在交错模式下，数据以连续帧的形式存放，即首先记录完帧 1 的左声道样本和右声道样本（假设为立体声格式），再开始帧 2 的记录，而在非交错模式下，首先记录的是一个周期内所有帧的左声道样本，再记录右声道样本，数据是以连续通道的方式存储。不过多数情况下，我们只需要使用交错模式就可以了
daudio	数字音频接口，可配置成 i2s/pcm 格式标准音频接口
cedarSE	全志音频算法管理框架
AEC	回声消除算法

2 音频系统框架概述

2.1 原型机音频硬件框架

支持的音频接口如下。

> AudioCodec x1
> I2S/PCM x4
> OWA x1
> DMIC x1

接口对应驱动生成的音频声卡设备节点可通过以下指令查看。

```
cat /proc/asound/cards
 0 [audiocodec    ]: audiocodec - audiocodec
                     audiocodec
 1 [sndi2s0       ]: sndi2s0 - sndi2s0
                     sndi2s0
 2 [sndi2s1       ]: sndi2s1 - sndi2s1
                     sndi2s1
```

其中。

audiocodec 声卡连接 mic 作为输入，speaker 作为输出，headset 即作为输入又作为输出。

sndi2s1 和 VoHCILoopback 声卡用于蓝牙通话。

2.2 软件框架图

音频软件框架如图所示。

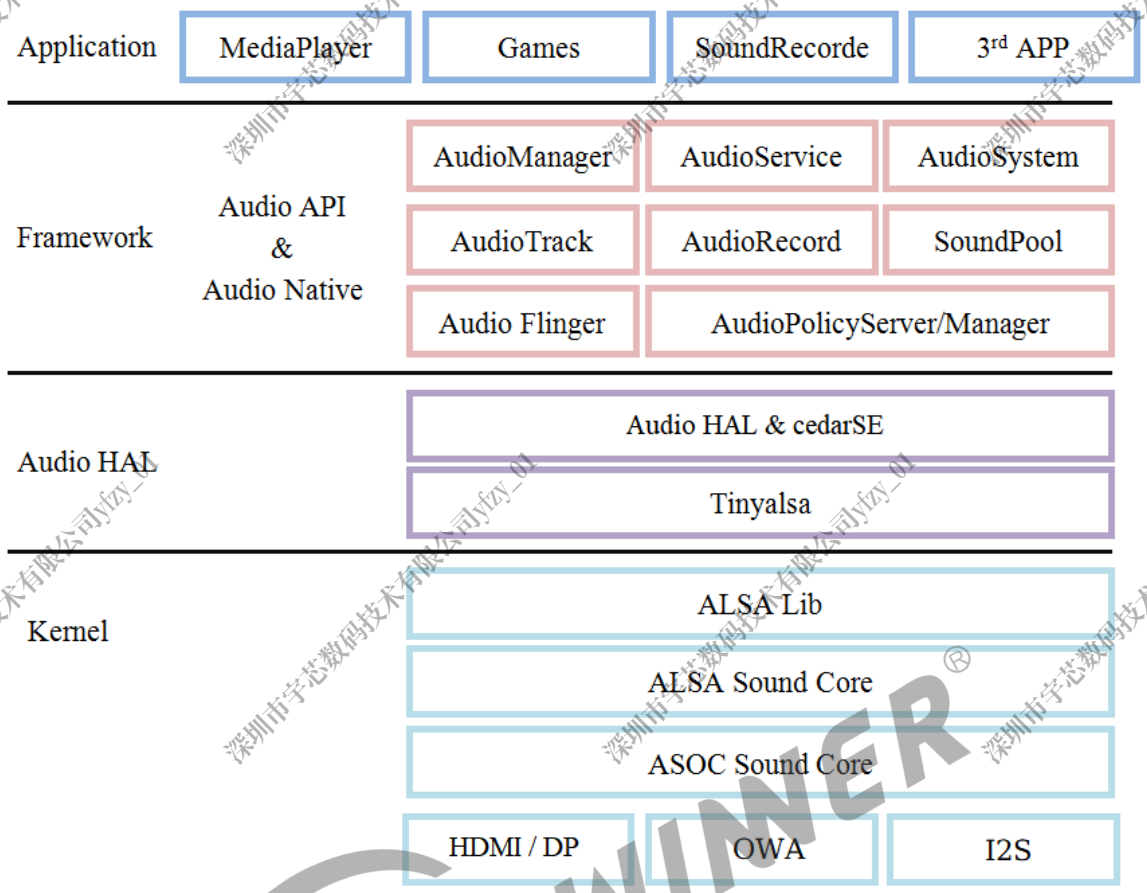

图 2-1: 音频软件框架图

3 音频驱动介绍

音频驱动相关的配置，请参考相应的《Linux_Audio_ 开发指南》文档。

4 音频 hal 介绍

全志音频 primary hal 框架由三部分组成，audio_device 模块实现安卓音频 hal 接口，platform 模块解析平台配置信息，cedarSE 模块为音频算法框架。

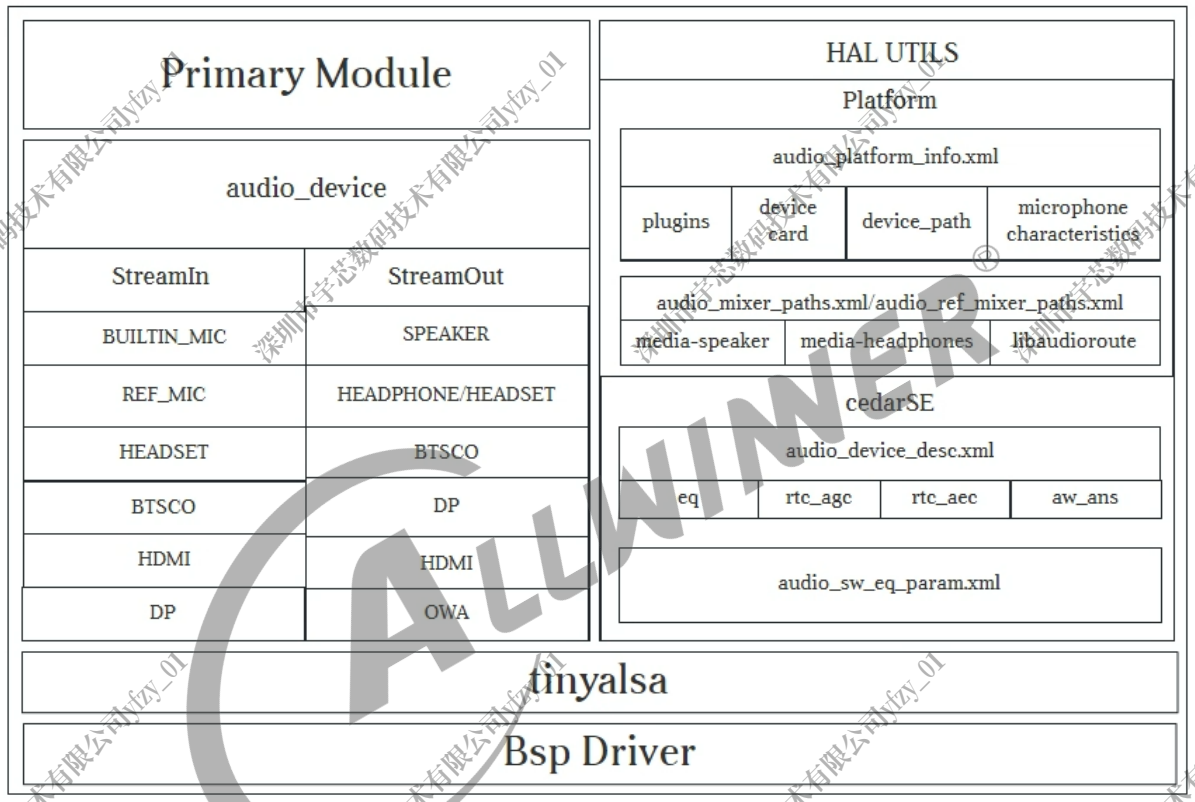

图 4-1: 音频 hal 框架图

全志音频 hal 支持以下音频设备类型，分别是。

音频 hal 音频设备类型	说明
OUT_SPK	单喇叭输出
OUT_DULSPK	双喇叭输出
OUT_HP	耳机输出
OUT_SPK_HP	单喇叭和耳机同时输出
OUT_DULSPK_HP	双喇叭和耳机同时输出
OUT_BTSCO	vohci 蓝牙通话输出
OUT_BTSCO_PCM	i2s 蓝牙通话输出
OUT_HDMI	HDMI/DP 音频输出

音频 hal 音频设备类型	说明
OUT_HDMI_RAW	HDMI 音频透传输出
OUT_OWA	OWA 音频输出
OUT_OWA_RAW	OWA 音频透传输出
OUT_SPK_HDMI	单喇叭和 HDMI 同时输出
OUT_DULSPK_HDMI	双喇叭和 HDMI 同时输出
OUT_HP_HDMI	耳机和 HDMI 同时输出
OUT_SPK_OWA	单喇叭和 OWA 同时输出
OUT_DULSPK_OWA	双喇叭和 OWA 同时输出
OUT_HP_OWA	耳机和 OWA 同时输出
OUT_HDMI_OWA	HDMI 和 OWA 同时输出
OUT_DULSPK_HDMI_OWA	双喇叭和 HDMI 和 OWA 同时输出
OUT_HP_HDMI_OWA	耳机和 HDMI 和 OWA 同时输出
OUT_REF	AEC 算法 i2s 回采参考音频输出
IN_AMIC	机器 MIC 输入
IN_DMIC	机器 DMIC 输入
IN_HPMIC	耳机 MIC 输入
IN_BTSCO	vohci 蓝牙通话输入
IN_BTSCO_PCM	i2s 蓝牙通话输入
IN_REF	AEC 算法 AC107 回采参考音频输入
IN_HDMI	HDMI/DP 音频输入

4.1 音频 hal 通路流程

1. 选择音频设备。

（1）根据安卓音频框架传下来的音频设备，选择对应的音频设备。如AUDIO_DEVICE_OUT_SPEAKER对应OUT_DULSPK。

（2）根据Settings应用设置下来的音频设备，选择对应的音频设备，优先级比安卓音频框架的高。

2. 根据传入的音频设备类型解析获取其对应音频路径配置并设置。

audio_platform_info.xml文件配置的device_path_map，如OUT_DULSPK双喇叭对应media-speaker音频路径。

audio_mixer_paths.xml文件配置的ctl，media-speaker音频路径，就会打开"LINEOUTL Switch"和"LINEOUTR Switch"和"SPK Switch"三个mixer。

3. 根据传入的音频设备类型解析获取其对应的实际声卡设备。

audio_platform_info.xml文件配置的platform_devices_profile，如OUT_DULSPK双喇叭对应audiocodec声卡。

4. 根据传入的音频输入/输出解析 CedarSE 音效配置文件，确认音效算法的使用并组件初始化。

5. 配置对应声卡设备的播放/录音参数并打开声卡。
6. CedarSE 音频数据算法处理。
7. 处理后音频数据写入/读取。

4.2 音频 hal 配置文件解析

4.2.1 audio_platform_info.xml

```
默认的audio_platform_info.xml文件路径。
device/softwinner/{CHIP}/common/media/audio/audio_platform_info.xml
各个不同方案的audio_platform_info.xml文件，如a333-pro方案的配置文件，会覆盖默认的音频配置文件。
device/softwinner/earth/a333-pro/media/audio/audio_platform_info.xml(目前没有覆盖，共用common目录配置文件)

platform_devices_profile配置音频设备对应的声卡，如OUT_DULSPK双喇叭对应audiocodec声卡。
其中audiocodec是底层驱动声卡ID，可通过cat /proc/asound/cards节点获取查看。
device="0"，对应pcmC0D0p中的D0。
channels="2" rate="48000" period_size="960" period_count="6"对应pcm_open时的pcm_config参数。

<platform_devices_profile>
  <platform_devices devices="OUT_SPK|OUT_EAR|OUT_HP|OUT_DULSPK|OUT_DULSPK_HP|OUT_SPK_HP"/>
  <snd_card_config
    type="frontend" card_name="audiocodec" device="0"
    channels="2" rate="48000" period_size="960" period_count="6"
    />
</platform_devices_profile>

platform_device_path配置音频设备对应的音频路由或音频控件，即配置对应输入/输出的音频通路。如OUT_DULSPK双喇叭
      对应media-speaker音频路径。

<platform_device_path>
  <device_path_map device="OUT_DULSPK" path="media-speaker"/>
</platform_device_path>
```

4.2.2 audio_mixer_paths.xml

```
默认的audio_mixer_paths.xml文件路径。
device/softwinner/earth/common/media/audio/audio_mixer_paths.xml

该文件是audiocodec声卡的音频路径配置文件。
如audio_platform_info.xml文件中，配置了OUT_DULSPK双喇叭对应media-speaker音频路径。
就会打开"LINEOUTL Switch"和"LINEOUTR Switch"和"SPK Switch"三个mixer。

<path name="media-speaker">
  <ctl name="LINEOUTL Switch" value="1" />
  <ctl name="LINEOUTR Switch" value="1" />
  <ctl name="SPK Switch" value="1" />
</path>

audio_mixer_paths.xml文件由audio_route.c解析。
新增其他声卡的配置文件，可以调用audio_route_init接口读取解析并配置。
```

4.3 音频设备通路

4.3.1 喇叭输出

1. 单喇叭输出。

```
OUT_SPK --> audiocodec声卡 --> media-single-speaker

有些版型喇叭只接了LINEOUTL，播放音频只有左声道声音输出，需要做软件左右声道混音输出。

int mix_s16_stereo_linear(char *input, int frames)
{
   int value = 0 ;
   int i = 0;
   struct pcm_channel *pcm_data = (struct pcm_channel *)input;
   for(i = 0 ; i < frames ; i++) {
      value = pcm_data[i].left + pcm_data[i].right;
      pcm_data[i].left = (short)(value/2);
      pcm_data[i].right = (short)(value/2);
   }
   return 0;
}

将方案中的双喇叭属性关闭，修改如下。
PRODUCT_PROPERTY_OVERRIDES += \
   ro.vendor.spk_dul.used=false \
```

2. 双喇叭输出。

```
OUT_DULSPK --> audiocodec声卡 --> media-speaker

需要在安卓方案目录中添加双喇叭属性
#set speaker project(true: double speaker, false: single speaker)
PRODUCT_PROPERTY_OVERRIDES += \
   ro.vendor.spk_dul.used=true \
```

4.3.1.1 喇叭音频输出调试

1. 核对原理图以及板卡器件，确认喇叭器件有正常焊接。
2. 确认存在 audiocodec 声卡，如果不存在则需要参考音频驱动开发指南，配置声卡。

```
# cat /proc/asound/cards
 0 [audiocodec    ]: audiocodec - audiocodec
                     audiocodec
```

3. 使用 tinyplay 确认 audiocodec 声卡工作正常。如果 tinyplay 播放没有声音，则需要排查音频驱动是否正常。

```
参考media-single-speaker/media-speaker音频路径打开对应的mixer。

如A333 media-speaker音频路径。
tinymix -D 0 "LINEOUTL Switch" 1
tinymix -D 0 "LINEOUTR Switch" 1
tinymix -D 0 "SPK Switch" 1
tinyplay test_2ch.wav -D 0
```

4. logcat 确认 audio hal 选择喇叭音频设备。

```
# logcat -c && logcat --pid=`ps -A | grep android.hardware.audio.service | awk '{print $2}'`

播放音乐，对应log，选择OUT_DULSPK喇叭音频输出设备，音频路径为media-speaker。
AHAL_ModulePrimary: select_devices: select device(out) pdev: OUT_DULSPK, path: media-speaker

打开card(0)，对应audiocodec声卡。
AHAL_StreamOutPrimary: start_output_stream: pcm_open, card(0) port(0) pcm_flags(0x8) rate(48000) format(0)
        channels(2) period_size(960) period_count(6) pcm(0xb400007b1ae3bd00)

执行lsof命令，确认也是打开audiocodec声卡。
# lsof | grep pcm
binder:444_2  444 audioserve  12u     CHR         116,2     0t0     1149 /dev/snd/pcmC0D0p

执行tinymix命令，确认media-speaker音频路径上的mixer都已设置正确。
```

5. dump 数据确认数据正常。

```
通过audiohal dump data方式确认上层传下来的数据或上传的数据是正常的。
dump data工具使用请见FAQ中的音频播放/录制HAL层获取数据方法。
```

4.3.2 MIC 输入

1. 单 MIC 输入。

```
IN_AMIC --> audiocodec声卡 --> media-main-mic
```

4.3.2.1 MIC 音频输入调试

1. 核对原理图以及板卡器件，确认麦克风器件有正常焊接。
2. 确认存在 audiocodec 声卡，如果不存在则需要参考音频驱动开发指南，配置声卡。

```
# cat /proc/asound/cards
 0 [audiocodec    ]: audiocodec - audiocodec
                     audiocodec
```

3. 使用 tinycap 确认 audiocodec 声卡工作正常。如果 tinycap 录音没有声音，则需要先排查音频驱动是否正常。

```
参考media-main-mic音频路径打开对应的mixer。

如media-main-mic音频路径。
tinymix -D 0 "MIC1 Switch" 1
tinycap mic.wav -D 0 -c 1 -T 10
```

4. logcat 确认 audio hal 选择 MIC 音频设备。

```
# logcat -c && logcat --pid=`ps -A | grep android.hardware.audio.service | awk '{print $2}'`

开始录音，对应log，选择IN_AMIC MIC音频输入设备，音频路径为media-main-mic。
AHAL_ModulePrimary: select_devices: select device(in) pdev: IN_AMIC, path: media-main-mic

打开card(0)，对应audiocodec声卡。
AHAL_StreamInPrimary: start_input_stream: pcm_open, card(0) port(0) pcm_flags(0x10000008) rate(48000) format(0)
        channels(1) period_size(960) period_count(2) pcm(0xb400007b1ae3bf80)

执行lsof命令，确认也是打开audiocodec声卡。
# lsof | grep pcm
 binder:444_2   444 audioserve  17u     CHR         116,3      0t0      1150 /dev/snd/pcmC0D0c

执行tinymix命令，确认media-main-mic音频路径上的mixer都已设置正确。
```

5. dump 数据确认数据正常。

```
通过audiohal dump data方式确认上层传下来的数据或上传的数据是正常的。
dump data工具使用请见FAQ中的音频播放/录制HAL层获取数据方法。
```

4.3.3 耳机

1. 耳机输出。

```
OUT_HP —> audiocodec声卡 —> media-headphones
```

2. 耳机输入。

```
IN_HPMIC —> audiocodec声卡 —> media-headset-mic
```

耳机输入输出使用的也是 audiocodec 声卡，输入输出调试可参考上面的喇叭和 MIC。

4.3.3.1 耳机事件调试

对于耳机插拔，或者耳机音量控制，耳机播放控制等问题，应先查看耳机事件是否正常上报。如果耳机事件没有上报，则先排查音频驱动的问题。如果耳机事件正常上报了，则排查安卓音频框架的问题。

1. 执行 getevent 命令，获取 audiocodec HeadPhones 的 event 设备。可以看到 audiocodec HeadPhones 的 event 设备为/dev/input/event2。

```
# getevent -l
add device 1: /dev/input/event1
 name:    "sunxi-keyboard"
add device 2: /dev/input/event0
 name:    "axp2202-pek"
add device 3: /dev/input/event2
 name:    "audiocodec Headphones"
add device 4: /dev/input/event3
 name:    "msa"
add device 5: /dev/input/event4
 name:    "chipone-tddi"
add device 6: /dev/input/event5
 name:    "chipone-tddi,pen"
add device 7: /dev/input/event6
 name:    "MH248"
```

2. 获取耳机插拔事件

```
 # getevent -l /dev/input/event2
插入耳机的事件。
EV_SW      SW_HEADPHONE_INSERT  00000001
EV_SW      SW_MICROPHONE_INSERT 00000001
EV_SYN     SYN_REPORT           00000000
对应串口打印。
[sound] jack report -> HEADSET

拔出耳机的事件。
EV_SW      SW_HEADPHONE_INSERT  00000000
EV_SW      SW_MICROPHONE_INSERT 00000000
EV_SYN     SYN_REPORT           00000000
对应串口打印。
[sound] jack report -> OUT
```

3. 获取耳机音量控制或者播放暂停事件

```
# getevent -l /dev/input/event2

耳机音量加事件。
EV_KEY     KEY_VOLUMEUP         DOWN
EV_SYN     SYN_REPORT           00000000
EV_KEY     KEY_VOLUMEUP         UP
EV_SYN     SYN_REPORT           00000000
对应串口打印。
[sound] jack report -> Volume ++

耳机音量减事件。
EV_KEY     KEY_VOLUMEDOWN       DOWN
EV_SYN     SYN_REPORT           00000000
EV_KEY     KEY_VOLUMEDOWN       UP
EV_SYN     SYN_REPORT           00000000
对应串口打印。
[sound] jack report -> Volume --
```

```
# getevent -l /dev/input/event2
耳机播放暂停事件。
EV_KEY      KEY_MEDIA         DOWN
EV_SYN      SYN_REPORT        00000000
EV_KEY      KEY_MEDIA         UP
EV_SYN      SYN_REPORT        00000000
对应串口log。
[sound] jack report -> Hook
```

4.3.4 蓝牙通话

```
audio hal会根据蓝牙服务设置的vendor.audio.btsco.pcm_support属性去判断该板子是否支持pcm蓝牙通话。

如果支持PCM蓝牙通话，则使用OUT_BTSCO_PCM和IN_BTSCO_PCM音频设备类型，即sndi2s1声卡。

如果支持UART蓝牙通话，则使用OUT_BTSCO和IN_BTSCO音频设备类型，即VoHCILoopback声卡。

audio_platform_info.xml文件中的蓝牙通话配置。
<platform_devices_profile>
  <platform_devices devices="OUT_BTSCO|IN_BTSCO"/>

  <snd_card_config
    type="frontend" card_name="VoHCILoopback" device="1"
    channels="1" rate="8000" period_size="960" period_count="4"
    />
</platform_devices_profile>

<platform_devices_profile>
  <platform_devices devices="OUT_BTSCO_PCM"/>

  <snd_card_config
    type="frontend" card_name="sndi2s1" device="0"
    channels="2" rate="8000" period_size="480" period_count="2"
    />

</platform_devices_profile>

<platform_devices_profile>
  <platform_devices devices="IN_BTSCO_PCM"/>

  <snd_card_config
    type="frontend" card_name="sndi2s1" device="0"
    channels="1" rate="8000" period_size="480" period_count="2"
    />
</platform_devices_profile>
```

4.3.4.1 蓝牙通话调试

1. 确认存在 sndi2s1 或者 VoHCILoopback 声卡，如果不存在则需要参考音频驱动开发指南，配置声卡。

```
# cat /proc/asound/cards
 0 [audiocodec    ]: audiocodec - audiocodec
                     audiocodec
 1 [sndi2s0       ]: sndi2s0 - sndi2s0
                     sndi2s0
 2 [sndi2s1       ]: sndi2s1 - sndi2s1
                     sndi2s1
```

2. logcat 确认 audio hal 选择蓝牙音频设备。

```
下面以sndi2s1声卡为例，进行蓝牙通话。
# logcat -c && logcat --pid=` ps -A | grep android.hardware.audio.service | awk '{print $2}' `

连接蓝牙设备，打开通话软件如微信，进行蓝牙通话测试。

开始通话，对应log，选择OUT_BTSCO_PCM蓝牙音频输出设备。
AHAL_ModulePrimary: select_devices: select device(out) pdev: OUT_BTSCO_PCM, path: com-ap-bt

打开card(2)，对应sndi2s1声卡。
AHAL_StreamOutPrimary: start_output_stream: pcm_open, card(2) port(0) pcm_flags(0) rate(8000) format(0) channels
       (2) period_size(480) period_count(2) pcm(0xf3388d00)

接通通话，打开录音，对应log，选择IN_BTSCO_PCM蓝牙音频输入设备。
AHAL_ModulePrimary: select_devices: select device(in) pdev: IN_BTSCO_PCM, path: com-bt-ap

打开card(2)，对应sndi2s1声卡。
AHAL_StreamInPrimary: start_input_stream: pcm_open, card(2) port(0) pcm_flags(0x10000000) rate(8000) format(0)
       channels(1) period_size(480) period_count(2) pcm(0xf3388e00)

执行lsof命令，确认也是打开sndi2s1声卡。
# lsof | grep pcm
binder:478_2  478 audioserve  24u    CHR         116,7     0t0      725 /dev/snd/pcmC2D0p
binder:478_2  478 audioserve  29u    CHR         116,8     0t0      728 /dev/snd/pcmC2D0c
```

3. 确认 audio hal 层打开的是 sndi2s1 声卡进行通话后，仍然无声音，则需要排查音频驱动是否正常。

4.4 cedarSE 介绍

cedarSE 是全志自研的音频算法管理框架。

目前支持的算法有。

- 软件音频算法：EQ、RTC_AEC、RTC_AGC、RTC_ANS、AW_ANS、PASS_FILTER、ROUTE、DELAY、VOLUME、INVERTER 音频算法。

4.4.1 cedarSE 模块配置

cedarSE 模块配置文件

默认的audio_device_desc.xml文件路径
device/softwinner/earth/common/media/audio/cedarSE/audio_device_desc.xml

该文件用于配置不同场景使用的不同算法。

（1）目前音频输出只支持Speaker场景。
如Speaker打开rtc_agc算法。
```
<play_device type="Speaker">
  <sw_effect sw="RTC_AGC"/>
</play_device>
```

同时audio_sw_rtc_agc_param.xml配置文件中，Speaker的enable设置为1
```
<param path="Speaker"     param_id="0" enable="1"/>
```

Speaker打开eq算法。
```
<play_device type="Speaker">
  <sw_effect sw="EQ" sub_id="1"/>
</play_device>
```

同时audio_sw_eq_param.xml配置文件中，Speaker的sub_id为1的enable设置为1
```
<param path="Speaker" sub_id="1" param_id="1" enable="1"/>
```

（2）音频输入支持MIC和MIC_PHONE和MIC_PHONE_AEC场景。
1. MIC为普通录音。
2. MIC_PHONE为通话场景。
3. MIC_PHONE_AEC只用于双mic AEC算法的通话场景。

MIC默认打开RTC_AGC算法，MIC_PHONE打开RTC_ANS和RTC_AGC算法，MIC_PHONE_AEC打开RTC_AEC和RTC_ANS算法。
```
<cap_device type="MIC">
  <sw_effect sw="RTC_AGC"/>
</cap_device>
<cap_device type="MIC_PHONE">
  <sw_effect sw="RTC_ANS"/>
  <sw_effect sw="RTC_AGC"/>
</cap_device>
<cap_device type="MIC_PHONE_AEC">
  <sw_effect sw="RTC_AEC"/>
  <sw_effect sw="RTC_ANS"/>
</cap_device>
```

同时audio_sw_rtc_agc_param.xml配置文件中，MIC和MIC_PHONE的enable设置为1
```
<param path="MIC"       param_id="1" enable="1"/>
<param path="MIC_PHONE"     param_id="1" enable="1"/>
```

同时audio_sw_rtc_ans_param.xml配置文件中，MIC_PHONE和MIC_PHONE_AEC的enable设置为1
```
<param path="MIC_PHONE"    param_id="2" enable="1"/>
<param path="MIC_PHONE_AEC" param_id="3" enable="1"/>
```

同时audio_sw_rtc_aec_param.xml配置文件中，MIC_PHONE_AEC的enable设置为1
```
<param path="MIC_PHONE_AEC" param_id="0" enable="1"/>
```

（3）device/softwinner/earth/common/media/audio/cedarSE目录下的其他配置文件为各个算法对应的配置文件。

如eq算法的配置文件audio_sw_eq_param.xml
```
<?xml version="1.0" encoding="UTF-8"?>
<audio_sw_eq_param version="2.10.0-later">
 <param_tree>
  <param path="Speaker" sub_id="0" param_id="0" enable="0"/>
  <!-- classic -->
```

```
  <param path="Speaker" sub_id="1" param_id="1" enable="0"/>
  <!-- flat -->
  <param path="Speaker" sub_id="2" param_id="2" enable="0"/>
  <!-- jazz -->
  <param path="Speaker" sub_id="3" param_id="3" enable="0"/>
  <!-- pop -->
  <param path="Speaker" sub_id="4" param_id="4" enable="0"/>
  <!-- rock -->
  <param path="Speaker" sub_id="5" param_id="5" enable="0"/>
  <param path="MIC"             param_id="6" enable="0"/>
 </param_tree>
 <param_unit_pool>
  <param_unit param_id="0">
   <!-- set "1" then use diffrent param per channel -->
   <param name="diff_param" value="0"/>
   <param name="channels" value="2"/>
   <param name="bin_num" value="2"/>
   <param name="samplerate" value="48000"/>
   <!-- filtertype fc gain quailty enable -->
   <param name="BQ1" value="1,500,1,0.6,1"/>
   <param name="BQ2" value="1,1000,1,0.6,1"/>
  </param_unit>
  <param_unit param_id="1">
   <param name="diff_param" value="0"/>
   <param name="channels" value="2"/>
   <param name="bin_num" value="5"/>
   <param name="samplerate" value="48000"/>
   <param name="BQ1" value="1,125,3,0.5,1"/>
   <param name="BQ2" value="1,500,-1,2,1"/>
   <param name="BQ3" value="1,1000,-1,2,1"/>
   <param name="BQ4" value="1,4000,-1,1,1"/>
   <param name="BQ5" value="2,16000,3,1,1"/>
  </param_unit>
  <param_unit param_id="2">
   <param name="diff_param" value="0"/>
   <param name="channels" value="2"/>
   <param name="bin_num" value="5"/>
   <param name="samplerate" value="48000"/>
   <param name="BQ1" value="1,125,3,2,1"/>
   <param name="BQ2" value="1,500,-3,2,1"/>
   <param name="BQ3" value="1,1000,3,2,1"/>
   <param name="BQ4" value="1,4000,-3,2,1"/>
   <param name="BQ5" value="2,16000,2,1,1"/>
  </param_unit>
  <param_unit param_id="3">
   <param name="diff_param" value="0"/>
   <param name="channels" value="2"/>
   <param name="bin_num" value="5"/>
   <param name="samplerate" value="48000"/>
   <param name="BQ1" value="1,125,-1,1,1"/>
   <param name="BQ2" value="1,500,-3,1,1"/>
   <param name="BQ3" value="1,1000,-3,1,1"/>
   <param name="BQ4" value="1,4000,2,2,1"/>
   <param name="BQ5" value="2,16000,2,1,1"/>
  </param_unit>
  <param_unit param_id="4">
   <param name="diff_param" value="0"/>
   <param name="channels" value="2"/>
   <param name="bin_num" value="5"/>
   <param name="samplerate" value="48000"/>
```

```
    <param name="BQ1" value="1,125,4,0.5,1"/>
    <param name="BQ2" value="1,500,-1,2,1"/>
    <param name="BQ3" value="1,1000,3,1,1"/>
    <param name="BQ4" value="1,4000,-1,2,1"/>
    <param name="BQ5" value="2,16000,2,1,1"/>
   </param_unit>
   <param_unit param_id="5">
    <param name="diff_param" value="0"/>
    <param name="channels" value="2"/>
    <param name="bin_num" value="5"/>
    <param name="samplerate" value="48000"/>
    <param name="BQ1" value="1,125,3,2,1"/>
    <param name="BQ2" value="1,500,-3,2,1"/>
    <param name="BQ3" value="1,1000,-1,2,1"/>
    <param name="BQ4" value="1,4000,1,2,1"/>
    <param name="BQ5" value="2,16000,2,1,1"/>
   </param_unit>
   <param_unit param_id="6">
    <!-- set "1" then use diffrent param per channel -->
    <param name="diff_param" value="1"/>
    <param name="channels" value="2"/>
    <param name="samplerate" value="48000"/>
    <param name="0_bin_num" value="2"/>
    <param name="0_BQ1" value="1,500,1,0.6,1"/>
    <param name="0_BQ2" value="1,700,1,0.6,1"/>
    <param name="1_bin_num" value="3"/>
    <param name="1_BQ1" value="1,400,1,0.6,1"/>
    <param name="1_BQ2" value="1,500,1,0.6,1"/>
    <param name="1_BQ3" value="1,700,1,0.6,1"/>
   </param_unit>
  </param_unit_pool>
</audio_sw_eq_param>

详情请参考《音效算法_开发指南》
```

4.4.2 Settings 应用打开 cedarSE

1. 打开 Settings 中 cedarSE 选项的 overlay 配置。

```
device/softwinner/earth/common/overlay/overlay/packages/apps/Settings/res/values/config.xml
<bool name="has_sfxmgmt_sounds">true</bool>
```

2. 配置 cedarSE 选项相关的属性。

```
device/softwinner/earth/common/media/config.mk
PRODUCT_SYSTEM_DEFAULT_PROPERTIES += \
  persist.sys.audio.use_sfxmgmt=true

PRODUCT_PROPERTY_OVERRIDES += \
  persist.vendor.audio.use_sfxmgmt=true
```

注意

persist.sys.audio.use_sfxmgmt 是 system 分区的属性，只作用于 Settings 应用中的 AW audio effects 选项是否被选中，手动 setprop 该属性是没有效果的。需要手动选择 Settings 应用中的选项功能才能生效。

手动关闭 Settings 应用中的 AW audio effects 选项，会通知 audio hal 设置 persist.vendor.audio.use_sfxmgmt 属性为 false，然后去掉 cedarSE 音效算法。

persist.vendor.audio.use_sfxmgmt 是 vendor 分区的属性，作用于 audio hal 中。手动 setprop 该属性为 false，也可以去掉 cedarSE 音效算法。

由于 system 和 vendor 分区的属性不能共用，所以只能使用两个属性。

4.4.3 EQ 和 DRC 的调式方法

EQ 和 DRC 的调试可以使用 TigerAudio 工具，它是 Allwinnertech SoC 的音频调试工具。软件的界面如下。

图 4-2: TigerAudio 主界面

关于 TigerAudio 工具的详细使用可以参考《TigerAudio_ 使用指南》

5 开机音乐

由于开机动画启动比 audioserver 启动靠前，所以全志的开机音乐采用直接写声卡的方式来实现，不走安卓音频框架通路。

5.1 开机音乐配置

5.1.1 开机动画文件

```
device/softwinner/{CHIP}/common/media/bootanimation/bootanimation.zip

解压之后可以看到audio_conf.txt开机音乐配置文件，和audio.wav开机音乐文件。
.
├── audio_conf.txt
├── desc.txt
├── part0
│   ├── 1_00000.png
│   ├── 1_00006.png
│   ├── 1_00015.png
│   ├── 1_00024.png
│   ├── 1_00030.png
│   ├── 1_00036.png
│   ├── 1_00042.png
│   ├── 1_00048.png
│   ├── 1_00054.png
│   ├── 1_00060.png
│   ├── 1_00072.png
│   ├── 1_00084.png
│   ├── 1_00096.png
│   └── audio.wav
└── part1
    └── 1_00101.png
```

5.1.2 audio_conf.txt 开机音乐配置文件解析

```
# audiocodec声卡号
card=0
# audiocodec声卡device号
device=0
# 打开声卡的pcm_config配置
period_size=2048
period_count=4

# 读取该节点，判断耳机是否插入了。
```

```
switch_path=/sys/module/snd_soc_sunxi_common/parameters/jack_state

# 设置为1表示喇叭和耳机同时出声。设置为0表示不插耳机时从喇叭出声，插入耳机后只从耳机出声。
speaker_headphones_out=0
# audiocodec声卡控件，控制喇叭输出声音大小。如果喇叭输出的开机音乐声音太小，可以调大这个值。如果声音太大，可以
        调小这个值。
mixer "LINEOUT Gain"=25
# 下面四个audiocodec声卡控件，用于控制喇叭输出通路。
mixer "LINEOUT Gain"=25
mixer "LINEOUTL Switch"=1
mixer "LINEOUTR Switch"=1
mixer "SPK Switch"=1
# 下面三个audiocodec声卡控件，用于控制耳机输出通路。
headset mixer "HPOUT Gain"=5
headset mixer "SPK Switch"=0
headset mixer "HPOUT Switch"=1
```

注意

修改 audio_conf.txt 开机音乐配置文件或者替换 audio.wav 开机音乐文件后，重新打包开机动画文件的命令。

```
zip -0qry -i \*.txt \*.png \*.wav @ bootanimation.zip *.txt part*
```

6 FAQ

6.1 查看播放或录音参数

原型机使用 audiocodec 进行录音和喇叭播放，使用 sndi2s1 或者 VoHCILoopback 进行蓝牙语音通话，输入以下命令查看系统当前音频设备节点。

```
# cat /proc/asound/cards
 0 [audiocodec    ]: audiocodec - audiocodec
                     audiocodec
 1 [sndi2s0       ]: sndi2s0 - sndi2s0
                     sndi2s0
 2 [sndi2s1       ]: sndi2s1 - sndi2s1
                     sndi2s1
```

由于 Android 13 之后使用了 GKI 2.0，内核不允许打开 CONFIG_SND_VERBOSE_PROCFS 选项。

所以，要想使用下面的调试功能，需要在内核中打开 CONFIG_SND_VERBOSE_PROCFS 选项。

同时不使用 GKI，在 device/softwinner/ceres/BoardConfig.mk 文件中将 CONFIG_AW_ENABLE_GKI 设为 false。

执行以下命令可获取当前播放硬件参数，如下所示。

```
cat /proc/asound/card0/pcm0p/sub0/hw_params

access: RW_INTERLEAVED
format: S16_LE
subformat: STD
channels: 2
rate: 48000 (48000/1)
period_size: 2720
buffer_size: 5440
```

若当前正在播放，执行以下命令可获取当前播放状态，如下所示。

```
cat /proc/asound/card0/pcm0p/sub0/status
state: RUNNING
owner_pid   : 4820
trigger_time: 1578540242.793497380
tstamp      : 1578540410.366986967
delay       : 3040
avail       : 2400
avail_max   : 2656
-----
hw_ptr      : 7390016
appl_ptr    : 7393056
```

执行以下命令可获取当前录音硬件参数，如下所示。

```
cat /proc/asound/card0/pcm0c/sub0/hw_params
access: RW_INTERLEAVED
format: S16_LE
subformat: STD
channels: 2
rate: 48000 (48000/1)
period_size: 1024
buffer_size: 2048
```

若当前正在录音，执行以下命令可获取当前录音状态，如下所示。

```
cat proc/asound/card0/pcm0c/sub0/status
state: RUNNING
owner_pid   : 4187
trigger_time: 4291.710215239
tstamp      : 4314.001112062
delay       : 1216
avail       : 832
avail_max   : 1184
-----
hw_ptr      : 1069984
appl_ptr    : 1071200
```

在没有打开任何节点，既不播放也不录音，或获取的设备节点并未播放或录音时，状态为closed，如下所示。

```
cat proc/asound/card0/pcm0c/sub0/hw_params
closed
```

6.2 tinyalsa 工具的使用

在 android/external/tinyalsa 目录下使用 mm 编译，会生成 tinycap tinyplay tinymix tinypcminfo tinyhostless 这五个调试工具，tinypcminfo tinyhostless 在当前平台中未使用。

编译生成的调试工具均在/android/out/target/product/版本号/system/bin 下，以 tinymix 举例，安装命令方法：

1. adb root
2. adb remount
3. adb push tinymix /vendor/bin/
4. adb shell
5. cd /vendor/bin/
6. chmod 777 tinymix

调试工具的用途与用法。

1. **tinycap** 录音测试工具。用于操作 audiocedec，DMIC 的音频录音设备节点。

```
Usage: tinycap file.wav [-D card] [-d device] [-c channels] [-r rate] [-b bits] [-p period_size] [-n n_periods] [-T capture time
    ]
```

2. **tinyplay** 播放测试工具。用于操作 audiocedec 的音频播放设备节点。

```
Usage: tinyplay file.wav [-D card] [-d device] [-p period_size] [-n n_periods]
```

3. **tinymix** 查看音频通路相关的各项配置参数，并通过命令修改参数配置。

6.3 音频播放/录制 HAL 层获取数据方法

获取步骤，源码请参考 audio_data_dump.c。

1. 更改 selinux 权限，机子上执行命令：setenforce 0
2. 设置抓取输出或者输入数据属性。

- setprop persist.vendor.audio.dump_data.out true
- setprop persist.vendor.audio.dump_data.in true

3. 默认文件保存路径。

- /data/vendor/audio_d/out.pcm
- /data/vendor/audio_d/in.pcm

6.4 framework 层获取数据方法

打开 AudioPolicyManager 的 log。

```
setprop log.tag.APM_AudioPolicyManager V
```

用于确认是否是框架层处理音频数据导致的问题。

```
参考：https://source.android.com/docs/core/audio/debugging
1. frameworks/av/services/audioflinger/Configuration.h打开TEE_SINK宏后重编烧固件或编库、推库和更新。
2. 确认adb shell getprop | grep ro.debuggable为1。
3. mkdir /data/misc/audioserver
chown media:media /data/misc/audioserver
echo af.tee=# > /data/local.prop (#值：1=输入；2=FastMixer输出；4=各音轨的AudioRecord和AudioTrack)
4. chmod 644 /data/local.prop && reboot
5. 播放异常音频---> dumpsys media.audio_flinger ---> /data/misc/media/xxxx.wav（生成音频文件）
```

6.5 打开安卓音频框架的 log

1. 打开 AudioPolicyManager 的打印，设置属性后，立刻生效。

```
setprop log.tag.APM_AudioPolicyManager V
```

2. 查看 media 事件信息。

```
dumpsys media.metrics
```

3. 打开安卓音频框架服务的 log

```
安卓音频框架audioflinger服务和audiopolicy服务源码目录。
frameworks/av/services/audioflinger
frameworks/av/services/audiopolicy

安卓音频框架应用层接口源码目录。
frameworks/av/media/libaudioclient

安卓音频框架audio hal中间层源码目录。
frameworks/av/media/libaudiohal

如果想打开audiopolicy服务的log，则执行以下命令。其他目录类似。
cd frameworks/av/services/audiopolicy
find -type f | xargs sed "s|//#define LOG_NDEBUG 0|#define LOG_NDEBUG 0|g" -i
find -type f | xargs sed "s|// #define LOG_NDEBUG 0|#define LOG_NDEBUG 0|g" -i
find -type f | xargs sed "s|//#define VERY_VERY_VERBOSE_LOGGING|#define VERY_VERY_VERBOSE_LOGGING|g" -i
```

注意

打印太多，可能会造成音频卡顿现象。

6.6 adb 中关闭按键音

```
settings put system sound_effects_enabled 0
```

6.7 usb 和蓝牙音频 buffer 调节

```
vim device/softwinner/earth/common/media/config.mk

# 调节usb音频的buffer大小，默认为20ms的buffer，如果卡顿可以将ro.vendor.audio.usb.period_us调大，如果延迟太大，
      可以调小一点。
# 调节蓝牙音频的buffer大小，默认为10ms的buffer，如果卡顿可以将ro.vendor.audio.bluetooth.period_ms调大，如果玩游
    戏时延迟太大，可以调小一点。
PRODUCT_PROPERTY_OVERRIDES += \
```

```
ro.vendor.audio.usb.period_us=20000 \
ro.vendor.audio.bluetooth.period_ms=10 \
```

6.8 设置系统开机后的默认音量

```
// 各个属性对应不同的stream的，如music stream对应media。
ro.config.alarm_vol_default
ro.config.media_vol_default
ro.config.system_vol_default
ro.config.vc_call_vol_default
```

6.9 音频音量曲线修改方法

6.9.1 实现原理

Android 系统启动后，根据 audio_policy_configuration.xml 配置 HAL 和音量曲线。以 A333 为例，路径：device/softwinner/earth/common/media/audio/，音量曲线部分代码如下。

```
<!-- Volume section -->
<xi:include href="audio_policy_volumes_drc.xml"/>
<xi:include href="default_volume_tables.xml"/>
```

代码可以看出音量曲线由 audio_policy_volumes_drc.xml 和 default_volume_tables.xml 两个配置文件组成。

Android 系统的音量由流类型和设备类型共同控制。流类型主要体现在 UI，不同场景下音量 UI 不同且互不干扰独自管理。一般平台的流类型如下所示，有些平台（TV）只有一个流类型。常用的流类型是 MUSIC MEDIA，因此一般会调整这两个流对应的音量曲线。设备类型一般是喇叭或者耳机。

```
protected static int[] MAX_STREAM_VOLUME = new int[] {
    5,  // STREAM_VOICE_CALL
    7,  // STREAM_SYSTEM
    7,  // STREAM_RING
    15, // STREAM_MUSIC
    7,  // STREAM_ALARM
    7,  // STREAM_NOTIFICATION
    15, // STREAM_BLUETOOTH_SCO
    7,  // STREAM_SYSTEM_ENFORCED
    15, // STREAM_DTMF
    15, // STREAM_TTS
    15  // STREAM_ACCESSIBILITY
};
```

6.9.2 修改方法

根据流和设备找到对应的音量曲线，代码在 audio_policy_volumes_drc.xml 中，部分如下所示。

```
<volume stream="AUDIO_STREAM_MUSIC" deviceCategory="DEVICE_CATEGORY_HEADSET"
                    ref="DEFAULT_MEDIA_VOLUME_CURVE"/>
<volume stream="AUDIO_STREAM_MUSIC" deviceCategory="DEVICE_CATEGORY_SPEAKER">
  <point>1,-4800</point>
  <point>33,-3000</point>
  <point>66,-2000</point>
  <point>100,0</point>
</volume>
```

比如修改 music 流，喇叭设备，可修改上述代码中 point 对应的值。音量曲线一般分为三段，成对数增长。（100，0）对应最大值，（1，-4800）对应最小值，-4800 可调整，若某一段音量不明显，可适当增加这一段的 point 差值。

比如修改 music 流，耳机设备，可修改上述代码中 DEFAULT_MEDIA_VOLUME_CURVE 的值。DEFAULT_MEDIA_VOLUME_CURVE 在 default_volume_tables.xml 中。修改方法同喇叭设备。

6.10 无声音问题排查

具体排查思路请参考上面的通路调试步骤。

6.11 录音声音小问题排查

1. 排查硬件参数是否达标，具体参考硬件开发指南。
2. 排查声卡控件确保下面控件的值。

```
1. ADC1 Gain和ADC2 Gain为31。
2. ADC1 Volume和ADC2 Volume至少为160。

# tinymix  -a
Mixer name: 'audiocodec'
Number of controls: 25
ctl    type    num    name                    value

0      ENUM    1      tx hub mode             Off
1      ENUM    1      rx sync mode            Off
2      ENUM    1      DAC DRC Mode            Off
3      ENUM    1      DAC HPF Mode            Off
4      ENUM    1      ADC DRC Mode            Off
5      ENUM    1      ADC HPF Mode            Off
6      ENUM    1      AD2DA Loop Mode         Off
7      ENUM    1      DA2AD Loop Mode         Off
8      ENUM    1      DACL DACR Swap          Off
9      ENUM    1      ADC1 ADC2 Swap          Off
10     INT     1      DAC Volume              63
```

```
11  INT  1  DACL Volume          160
12  INT  1  DACR Volume          160
13  INT  1  ADC1 Volume          160
14  INT  1  ADC2 Volume          160
15  INT  1  LINEOUT Gain          31
16  INT  1  HPOUT Gain             7
17  INT  1  ADC1 Gain             31
18  INT  1  ADC2 Gain             31
19  BOOL 1    MIC1 Switch         Off
20  BOOL 1    MIC2 Switch         On
21  BOOL 1    LINEOUTL Switch       Off
22  BOOL 1    LINEOUTR Switch       Off
23  BOOL 1    HPOUT Switch        On
24  BOOL 1    SPK Switch         Off
```

3. 使用算法

可以使用 rtc_agc 算法来增大 MIC 录音声音大小。

没有添加 rtc_agc 算法时的录音波形在-12 到-6db 之间。

添加 rtc_agc 算法后的录音波形在 0db 左右，声音明显增大。

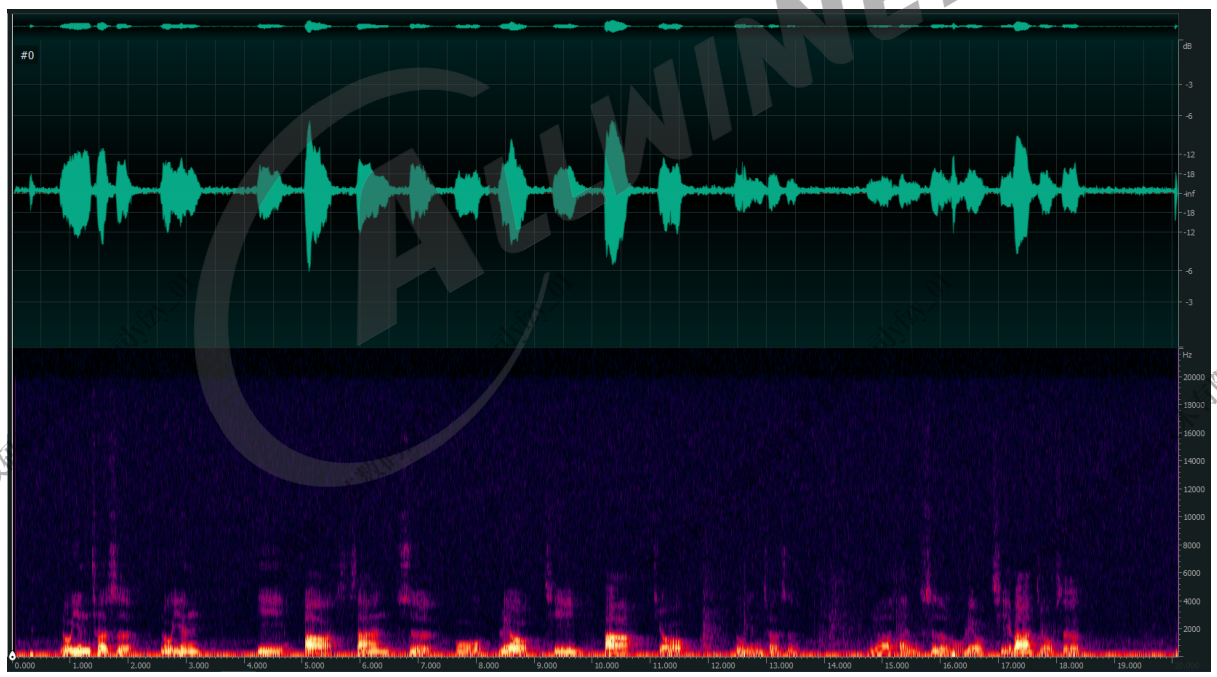

图 6-1: MIC 没有添加 rtc_agc 算法时的录音波形频谱图

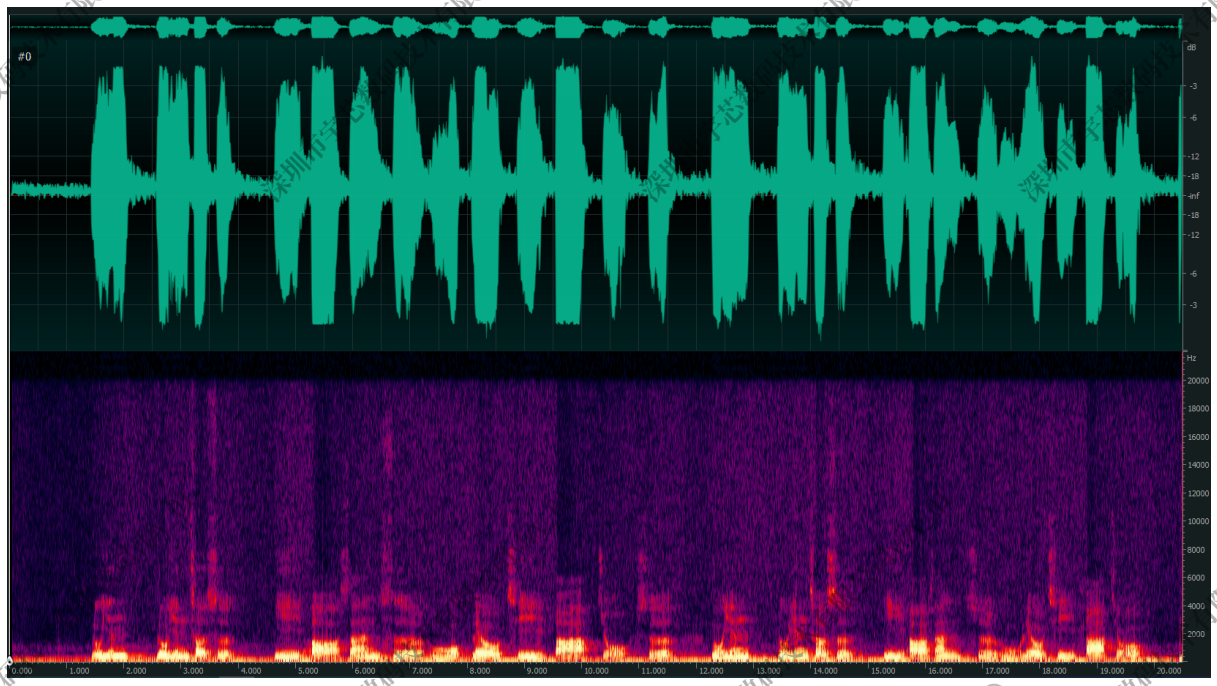

图 6-2: MIC 添加 rtc_agc 算法后的录音波形频谱图

6.12 喇叭声音小问题排查

1. 排查硬件参数是否达标，具体参考硬件开发指南。
2. 排查声卡控件。确保下面控件的值。

```
1. DAC Volume为63。
2. DACL Volume和DACR Volume至少为160。
3. LINEOUT Gain为31。
4. HPOUT Gain为7。

# tinymix -a
Mixer name: 'audiocodec'
Number of controls: 25
ctl   type   num   name                value

0     ENUM   1     tx hub mode         Off
1     ENUM   1     rx sync mode        Off
2     ENUM   1     DAC DRC Mode        Off
3     ENUM   1     DAC HPF Mode        Off
4     ENUM   1     ADC DRC Mode        Off
5     ENUM   1     ADC HPF Mode        Off
6     ENUM   1     AD2DA Loop Mode     Off
7     ENUM   1     DA2AD Loop Mode     Off
8     ENUM   1     DACL DACR Swap      Off
9     ENUM   1     ADC1 ADC2 Swap      Off
10    INT    1     DAC Volume          63
11    INT    1     DACL Volume         160
12    INT    1     DACR Volume         160
13    INT    1     ADC1 Volume         160
14    INT    1     ADC2 Volume         160
```

```
15   INT   1    LINEOUT Gain              31
16   INT   1    HPOUT Gain                7
17   INT   1    ADC1 Gain                 31
18   INT   1    ADC2 Gain                 31
19   BOOL  1      MIC1 Switch               Off
20   BOOL  1      MIC2 Switch               On
21   BOOL  1      LINEOUTL Switch             On
22   BOOL  1      LINEOUTR Switch             On
23   BOOL  1      HPOUT Switch              Off
24   BOOL  1      SPK Switch              Onn
```

3. 使用算法

（1）使用 rtc_agc 算法来增大喇叭播放声音大小。

添加 rtc_agc 算法的音频会比没添加 rtc_agc 算法的音频大 3db 左右。

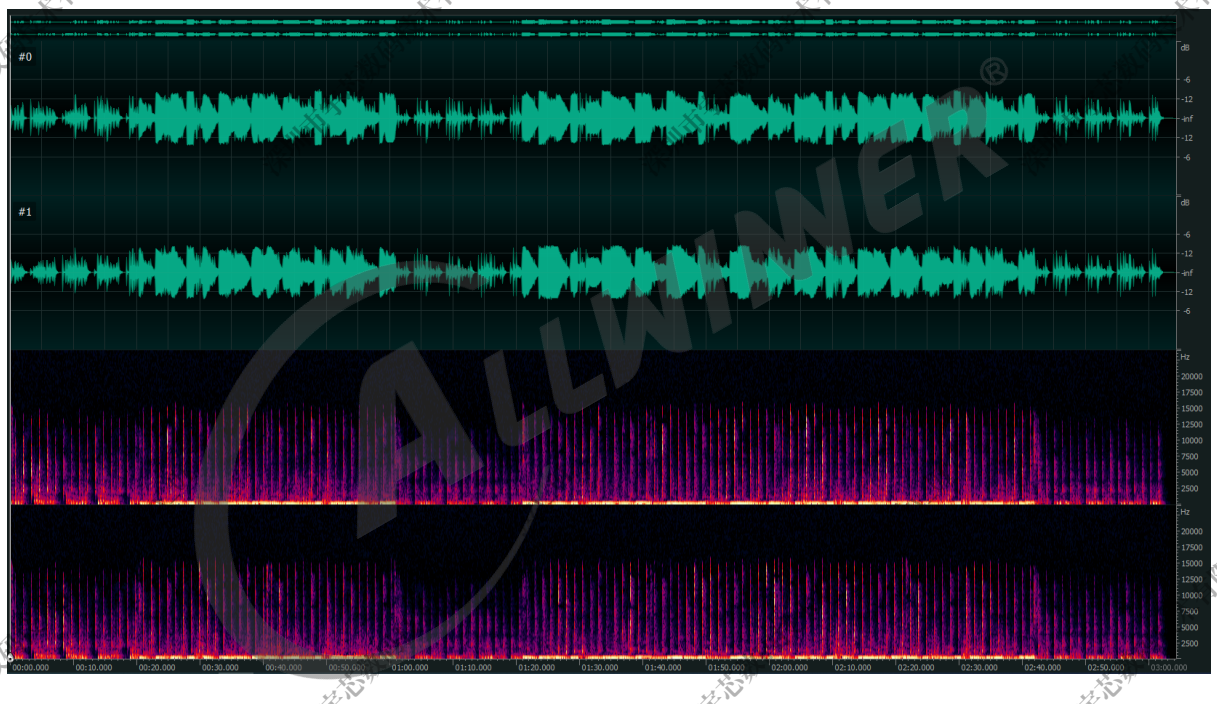

图 6-3: 喇叭没有添加 rtc_agc 算法时的播放波形频谱图

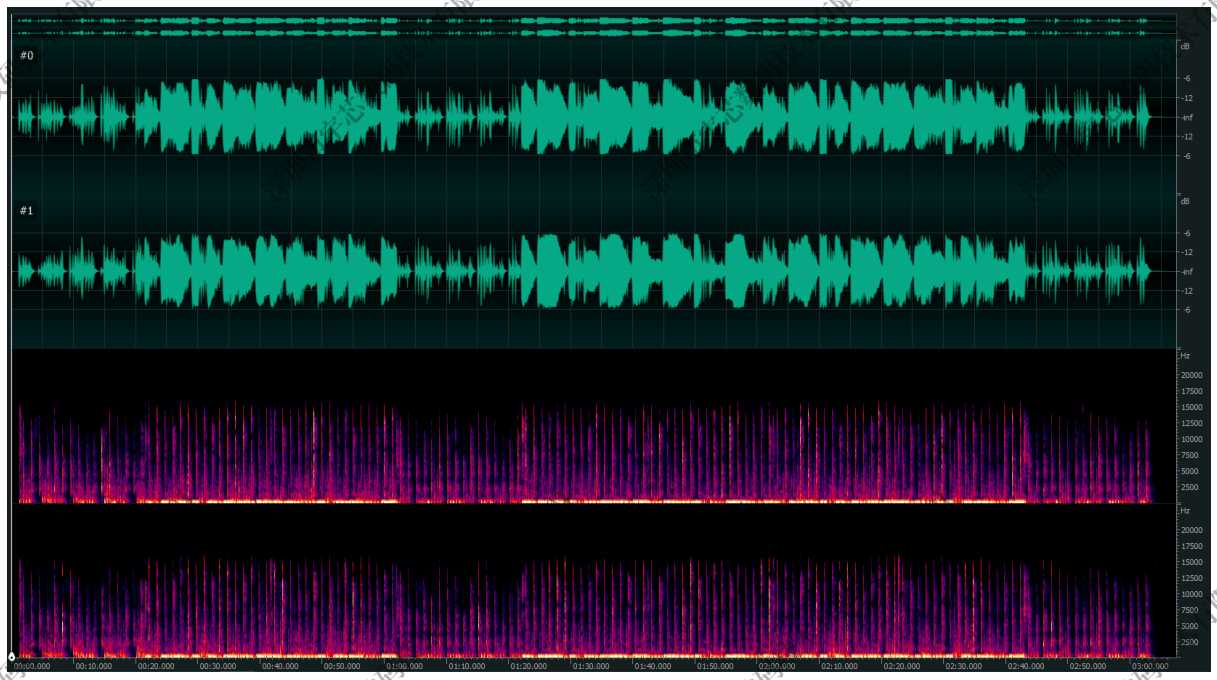

图 6-4: 喇叭添加 rtc_agc 算法后的播放波形频谱图

（2）添加 3d 环绕音音效算法，具体打开方式参照上面的使能虚拟环绕声。

注意

3D 环绕音由于其实现原理，打开使用后会导致左、右声道都输出为混合后的左 + 右声道数据输出。

增加 3D 环绕音音效算法，喇叭输出功率也会稍微增大。

（3）在喇叭功率允许的范围内，可以适当增大 DACL Volume 和 DACR Volume 的值。

6.13 通话啸叫问题排查

1. 排查硬件参数是否达标，具体参考硬件开发指南。
2. 排查 MIC 录播效果。

（1）使用tinyplay播放1KHz 0db的音乐。
（2）同时使用tinycap进行录音。
（3）同时测试人员大声说话。
（4）播放tinycap录音音频，看是否能听到测试人员说话的声音。

正常是能听到测试人员说话的声音的，如果听不到，则要排查下MIC的摆放位置或者共振问题。

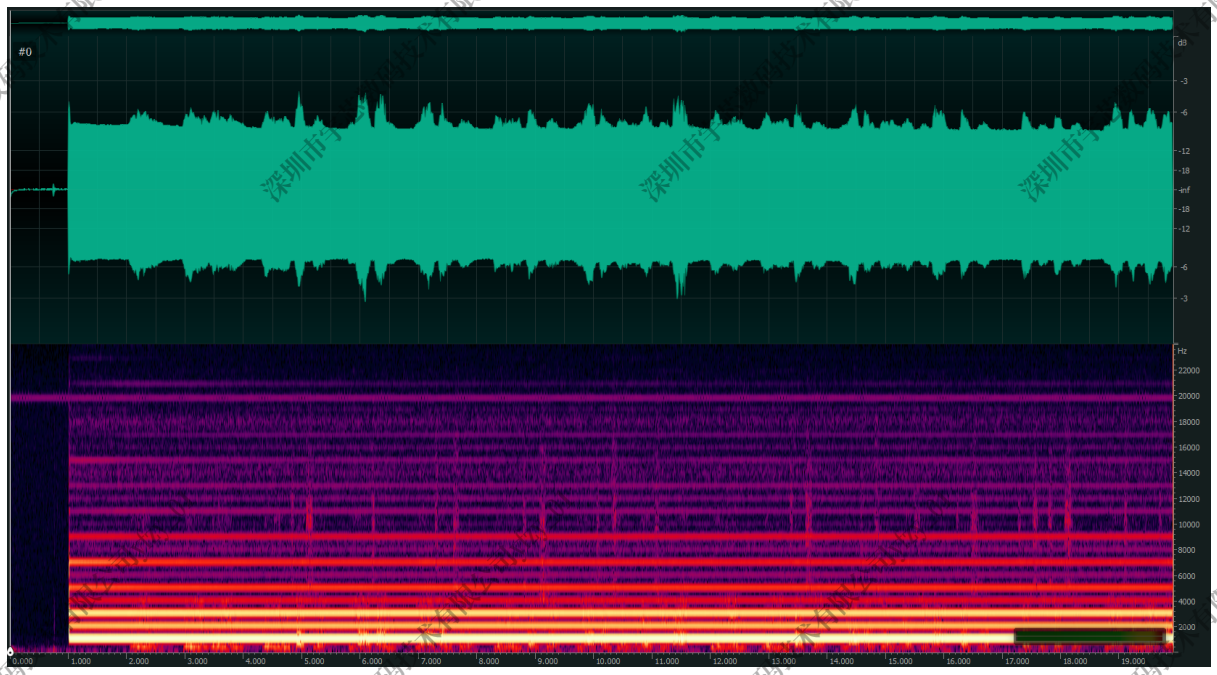

图 6-5: 正常能听到测试人员说话声音的波形频谱图

图 6-6: MIC 发生共振不能听到测试人员说话声音的波形频谱图

3. 使用算法

Google 原生单 mic aec 算法 + rtc_ans 算法（默认使用该配置）。

著作权声明

本文档及内容受著作权法保护，其著作权由珠海全志科技股份有限公司（“全志”）拥有并保留一切权利。

本文档是全志的原创作品和版权财产，未经全志书面许可，任何单位和个人不得擅自摘抄、复制、修改、发表或传播本文档内容的部分或全部，且不得以任何形式传播。

商标声明





（不完全列举）均为珠海全志科技股份有限公司的商标或者注册商标。在本文档描述的产品中出现的其它商标，产品名称，和服务名称，均由其各自所有人拥有。

免责声明

您购买的产品、服务或特性应受您与珠海全志科技股份有限公司（“全志”）之间签署的商业合同和条款的约束。本文档中描述的全部或部分产品、服务或特性可能不在您所购买或使用的范围内。使用前请认真阅读合同条款和相关说明，并严格遵循本文档的使用说明。您将自行承担任何不当使用行为（包括但不限于如超压，超频，超温使用）造成的不利后果，全志概不负责。

本文档作为使用指导仅供参考。由于产品版本升级或其他原因，本文档内容有可能修改，如有变更，恕不另行通知。全志尽全力在本文档中提供准确的信息，但并不确保内容完全没有错误，因使用本文档而发生损害（包括但不限于间接的、偶然的、特殊的损失）或发生侵犯第三方权利事件，全志概不负责。本文档中的所有陈述、信息和建议并不构成任何明示或暗示的保证或承诺。

本文档未以明示或暗示或其他方式授予全志的任何专利或知识产权。在您实施方案或使用产品的过程中，可能需要获得第三方的权利许可。请您自行向第三方权利人获取相关的许可。全志不承担也不代为支付任何关于获取第三方许可的许可费或版税（专利税）。全志不对您所使用的第三方许可技术做出任何保证、赔偿或承担其他义务。