

杰理 AD14N_AC104N 音效算法应用介绍

珠海市杰理科技股份有限公司
Zhuhai Jieli Technologyco.,LTD
版权所有，未经许可，禁止外传

2022 年 7 月 12 日

ZHUHAI JIELI TECHNOLOGY CO.,LTD

目录

| | |
|--|----|
| 第一章 音效应用说明..... | 4 |
| 1.1 混响（ECHO）..... | 4 |
| 1.1.1 配置参数说明..... | 4 |
| 1.1.2 相关函数..... | 5 |
| 1.1.2.1 函数 void *link_echo_sound(void *p_sound_out, void *p_dac_cbuf, void **pp_effect, u32 sr).... | 5 |
| 1.1.2.2 函数 void *echo_api(void *obuf, u32 sr, void **ppsound)..... | 5 |
| 1.2 移频啸叫抑制（PITCHSHIFTER_HOWLING）..... | 6 |
| 1.2.1 配置参数说明..... | 6 |
| 1.2.2 相关函数..... | 7 |
| 1.2.2.1 函数 void *link_pitchshift_howling_sound(void *p_sound_out, void *p_dac_cbuf, void **pp_effect, u32 sr)..... | 7 |
| 1.2.2.2 函数 void *pitchshift_howling_api(void *obuf, u32 sr, void **ppsound)..... | 7 |
| 1.3 自适应陷波啸叫抑制（NOTCH_HOWLING）..... | 8 |
| 1.3.1 配置参数说明..... | 8 |
| 1.3.2 相关函数..... | 9 |
| 1.3.2.1 函数 void *link_notch_howling_sound(void *p_sound_out, void *p_dac_cbuf, void **pp_effect, u32 sr)..... | 9 |
| 1.3.2.2 函数 void *notch_howling_api(void *obuf, u32 sr, void **ppsound)..... | 9 |
| 1.4 均衡器（PCM_EQ）..... | 10 |
| 1.4.1 配置参数说明..... | 10 |
| 1.4.2 相关函数..... | 11 |
| 1.4.2.1 函数 void *link_pcm_eq_sound(void *p_sound_out, void *p_dac_cbuf, void **pp_effect, u32 sr, u32 channel)..... | 11 |
| 1.4.2.2 函数 void *pcm_eq_api(void *obuf, u32 sr, u32 channel, void **ppsound)..... | 11 |
| 1.5 变音音效（VO_PITCH）..... | 12 |
| 1.5.1 可选音效..... | 12 |
| 1.5.2 配置参数说明..... | 12 |
| 1.5.3 相关结构体及函数..... | 13 |
| 1.5.3.1 结构体 VOICE_PITCH_PARA_STRUCT..... | 13 |
| 1.5.3.2 函数 void *link_voice_pitch_sound(sound_out_obj *p_curr_sound, void *p_dac_cbuf, void **pp_effect, u32 cmd)..... | 14 |
| 1.5.3.3 函数 void *voice_pitch_api(void *obuf, u32 cmd, void **ppsound)..... | 14 |
| 1.5.3.4 函数 rap_callback(void *priv, int pos)..... | 14 |
| 1.5.3.5 函数 void rap_reopen(void)..... | 14 |
| 第二章 EQ 工具使用说明..... | 15 |
| 2.1 EQ 工具使用介绍..... | 15 |
| 2.2 SDK 配置说明..... | 17 |

第一章 音效应用说明

1.1 混响 (echo)

混响 (echo) 音效一般用于普通扩音，可配置干湿音、回声等效果；一般需要与啸叫抑制算法一起使用。

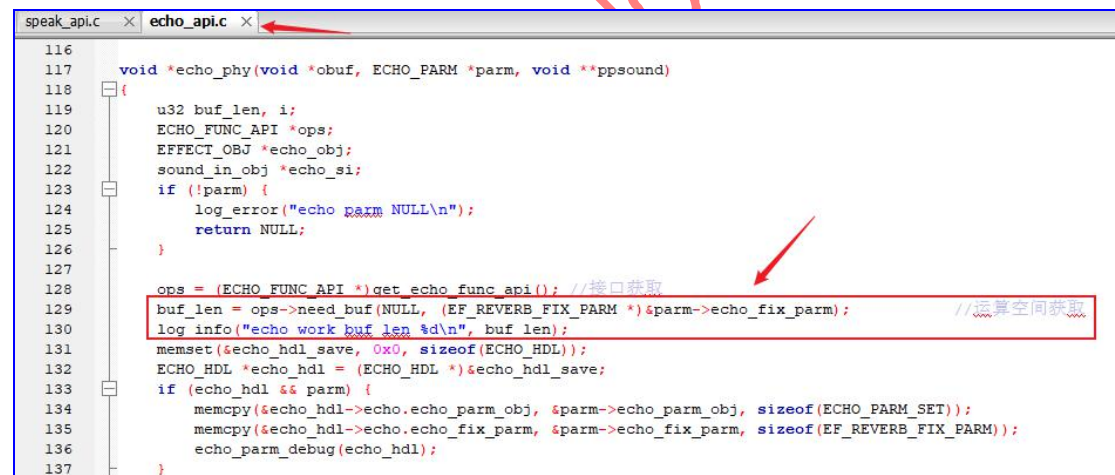
1.1.1 配置参数说明

- DOWN_S_FLAG: 下采参数

1 代表下采，0 代表不下采；下采申请的 buff 会较小，但是回声部分高频会丢失；

- echo_work_buf_len: echo 运算需要的 buff 大小

根据是否下采以及输入音频采样率决定，可通过 ops->need_buf()接口获取大小；



```
116 void *echo_phy(void *obuf, ECHO_PARM *parm, void **ppsound)
117 {
118     u32 buf_len, i;
119     ECHO_FUNC_API *ops;
120     EFFECT_OBJ *echo_obj;
121     sound_in_obj *echo_si;
122     if (!parm) {
123         log_error("echo parm NULL\n");
124         return NULL;
125     }
126     ops = (ECHO_FUNC_API *)get_echo_func_api(); //接口获取
127     buf_len = ops->need_buf(NULL, (EF_REVERB_FIX_PARM *)parm->echo_fix_parm); //运算空间获取
128     log_info("echo work buf len %d\n", buf_len);
129     memset(&echo_hdl_save, 0x0, sizeof(ECHO_HDL));
130     ECHO_HDL *echo_hdl = (ECHO_HDL *)&echo_hdl_save;
131     if (echo_hdl && parm) {
132         memcpy(&echo_hdl->echo.parm_obj, &parm->echo_parm_obj, sizeof(ECHO_PARM_SET));
133         memcpy(&echo_hdl->echo.echo_fix_parm, &parm->echo_fix_parm, sizeof(EF_REVERB_FIX_PARM));
134         echo_parm_debug(echo_hdl);
135     }
136 }
```

1.1.2 相关函数

1.1.2.1 函数 void *link_echo_sound(void *p_sound_out, void *p_dac_cbuf, void **pp_effect, u32 sr)

该函数为串联音频链路的标准接口，实现响应算法的启动，并添加到音频链路上，其中参数：

- ① p_sound_out: 当前算法的前级音频通道句柄;
- ② p_dac_cbuf: 输出的音频 buff;
- ③ pp_effect: 存放当前音效算法控制句柄指针的指针;
- ④ sr: 当前输入音频的采样率;
- ⑤ 返回值: 启动之后最新的音频通道句柄;

1.1.2.2 函数 void *echo_api(void *obuf, u32 sr, void **ppsound)

该函数用于初始化混响（echo）的相关参数，由函数 link_echo_sound()调用即可，需要关注的其内部参数的初始化：

- ① decayval: 回升衰减比，范围：0~70;
- ② delay: 回声延时，范围：0~max_ms，超过 max_ms 会当场 max_ms;
- ③ energy vad threshold: mute 阈值，默认值为 512；低于该能量值的声音输出静音;

扩音方案中可以用该方式降低底噪：

- ④ direct_sound_enable: 干声叠加使能位;
- ⑤ max ms: 最大回声延时时间，最大值为 100;
- ⑥ sr: 输入音频的采样率，支持 16K / 24K，会影响混响算法运算 buff 的大小;
- ⑦ wetgain: 湿声增益，默认为 3000;
- ⑧ drygain: 干声增益，默认为 4096，干声叠加使能为 0 时不起作用;

1.2 移频啸叫抑制 (pitchshifter_howling)

如果在一个环路系统的 AD→...→DA 中存在某个频点明显的响应峰值,该位置容易形成啸叫。因此,AD→...→移频→...→DA,在一个 AD 到 DA 的环路中,移频模块将频率不断移动,加入非线性运算,来改变整个环路的频响,防止同一个频率被反复放大。

移频防啸叫模块,可以配合 EQ 模块,自适应陷波模块一起处理。具体根据实际样机的资源效果做选择。

1.2.1 配置参数说明

- FRESHIFT_SPEED_MODE_QUALITY: 滤波器音阶数

滤波器音阶数配置范围[2, 8], 音阶数越大, 运算量越大;

- howling_work_buf: 移频啸叫抑制运算需要的 buff

根据移频方式和滤波器音阶数决定 buff 大小, 可通过 ops->need_buf()接口获取大小;

```

14
15 void *pitchshift_howling_api(void *obuf, u32 sr, void **ppsound)
16 {
17     HOWLING_PITCHSHIFT_PARM phparam = {0};
18     phparam.ps_parm = -200; // 等比移频, 建议范围 -350 到 350, 归一化系数为 8192
19     phparam.fs_parm = -10; // 线性移频, 建议范围 -10 到 10 (Hz)
20     phparam.effect_v = EFFECT_HOWLING_PS; // 选择需要的移频方式
21
22     HOWLING_PITCHSHIFT_FUNC_API *ops;
23     ops = (HOWLING_PITCHSHIFT_FUNC_API *)get_howling_ps_func_api(); // 接口获取
24     u32 buf_len = ops->need_buf(phparam.effect_v); // 运算空间获取
25     log_info("howling work buf len %d\n", buf_len);
26     if (sizeof(howling_work_buf) < buf_len) {
27         log_error("howling work buf less %d, need len %d", sizeof(howling_work_buf), buf_len);
28         return NULL;
29     }
30
31     return howling_phy(obuf, &howling_work_buf[0], &phparam, sr, ppssound);
32 }
    
```

1.2.2 相关函数

1.2.2.1 函数 void *link_pitchshift_howling_sound(void *p_sound_out, void *p_dac_cbuf, void **pp_effect, u32 sr)

该函数为串联音频链路的标准接口，实现响应算法的启动，并添加到音频链路上，其中参数：

- ① p_sound_out: 当前算法的前级音频通道句柄;
- ② p_dac_cbuf: 输出的音频 buff;
- ③ pp_effect: 存放当前音效算法控制句柄指针的指针;
- ④ sr: 当前输入音频的采样率;
- ⑤ 返回值: 启动之后最新的音频通道句柄;

1.2.2.2 函数 void *pitchshift_howling_api(void *obuf, u32 sr, void **ppsound)

该函数用于初始化移频抑制啸叫（pitchshifter_howling）的相关参数，由函数 link_pitchshift_howling_sound()调用即可，需要关注的其内部参数的初始化：

- ① effect_v: 移频方式, 可选 EFFECT_HOWLING_PS 或 EFFECT_HOWLING_FS;
- ② ps_parm: 等比移频, 建议范围[-350, 350], 归一化系数为 8192, effect_v 置上 EFFECT_HOWLING_PS 有效;
- ③ fs_parm: 线性移频, 建议范围[-10, 10] (Hz), effect_v 置上 EFFECT_HOWLING_FS 有效;

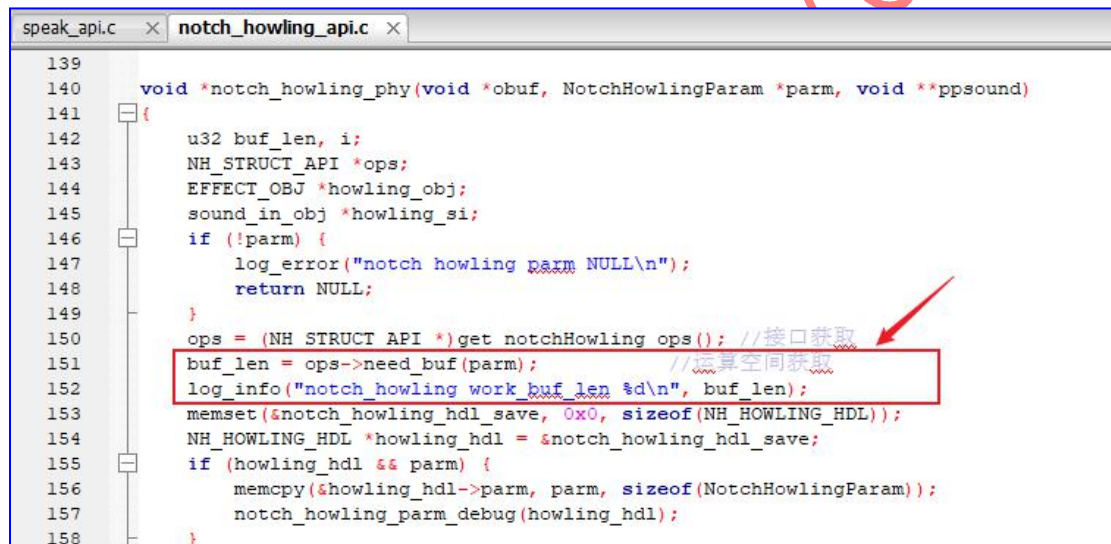
1.3 自适应陷波啸叫抑制（notch_howling）

使用扩音时，当麦克风到喇叭整体环路反馈大于 1 的时候，就会发生啸叫。只要改变整个环路的反馈，就可以抑制啸叫。自适应陷波抑制啸叫是其中一种算法，可以抑制改变环路反馈声音的频率，环路反馈一直改变，起到了平衡磨损的作用，防止某个频率一直放大。

1.3.1 配置参数说明

- notch_howling_work_buf: 自适应陷波啸叫抑制运算需要的 buff

根据配置参数决定 buff 大小，可通过 ops->need_buf()接口获取大小；



```
139
140 void *notch_howling_phy(void *obuf, NotchHowlingParam *parm, void **ppsound)
141 {
142     u32 buf_len, i;
143     NH_STRUCT_API *ops;
144     EFFECT_OBJ *howling_obj;
145     sound_in_obj *howling_si;
146     if (!parm) {
147         log_error("notch howling parm NULL\n");
148         return NULL;
149     }
150     ops = (NH_STRUCT_API *)get_notchHowling_ops(); //接口获取
151     buf_len = ops->need_buf(parm); //运算空间获取
152     log_info("notch howling work buf len %d\n", buf_len);
153     memset(&notch_howling_hdl_save, 0x0, sizeof(NH_HOWLING_HDL));
154     NH_HOWLING_HDL *howling_hdl = &notch_howling_hdl_save;
155     if (howling_hdl && parm) {
156         memcpy(&howling_hdl->parm, parm, sizeof(NotchHowlingParam));
157         notch_howling_parm_debug(howling_hdl);
158     }
```


1.3.2 相关函数

1.3.2.1 函数 void *link_notch_howling_sound(void *p_sound_out, void *p_dac_cbuf, void **pp_effect, u32 sr)

该函数为串联音频链路的标准接口，实现响应算法的启动，并添加到音频链路上，其中参数：

- ① p_sound_out: 当前算法的前级音频通道句柄;
- ② p_dac_cbuf: 输出的音频 buff;
- ③ pp_effect: 存放当前音效算法控制句柄指针的指针;
- ④ sr: 当前输入音频的采样率;
- ⑤ 返回值: 启动之后最新的音频通道句柄;

1.3.2.2 函数 void *notch_howling_api(void *obuf, u32 sr, void **ppsound)

该函数用于初始化自适应陷波抑制啸叫（notch_howling）的相关参数，由函数 link_notch_howling_sound()调用即可，需要关注的其内部参数的初始化：

- ① gain: 陷波器压制程度，该值越大防啸叫效果越好，但发声啸叫频点误检时音质会更差;
- ② Q: 陷波器带宽，该值越小防啸叫效果越好，但发声啸叫频点误检时音质会更差;
- ③ fade_time: 陷波器启动与释放时间，该值越小启动与释放越快，但可能导致杂音出现和音质变差
- ④ threshold: 频点啸叫判定阈值，该值越小防啸叫效果越好，但越容易误检;
- ⑤ SampleRate: 输入音频的采样率;

1.4 均衡器 (PCM_EQ)

AD14N / AC104N 可以使用杰理 EQ 工具进行离线 EQ 调试, 将需要的频段增益抬高或者压低, 使音质感受更佳。

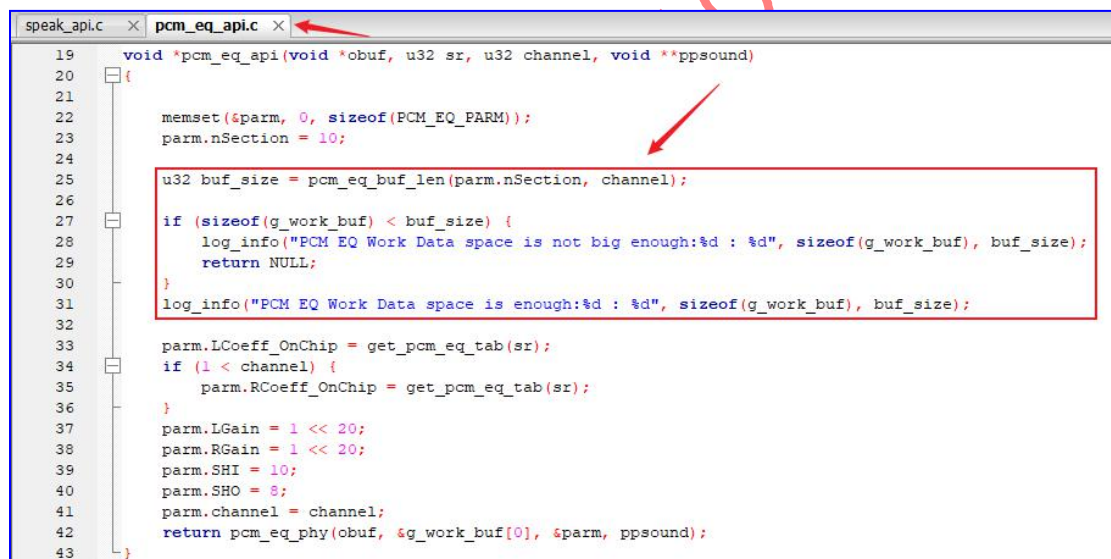
1.4.1 配置参数说明

- eq_filt_32000: PCM_EQ 处理 32K 采样率音频的运算数据

该数据由 EQ 工具生成, 最多可以有 10 段 EQ 数据 (seg0~seg9), 详情请见 EQ 工具使用说明章节, 不同采样率拥有各自的 eq_filt 表;

- g_work_buf: PCM_EQ 运算需要的 buff

根据 EQ 使用的频点数 (段数) 决定 buff 大小, 可通过 pcm_eq_buf_len() 接口获取大小;



```
19 void *pcm_eq_api(void *obuf, u32 sr, u32 channel, void **ppsound)
20 {
21
22     memset(&parm, 0, sizeof(PCM_EQ_PARM));
23     parm.nSection = 10;
24
25     u32 buf_size = pcm_eq_buf_len(parm.nSection, channel);
26
27     if (sizeof(g_work_buf) < buf_size) {
28         log_info("PCM EQ Work Data space is not big enough:%d : %d", sizeof(g_work_buf), buf_size);
29         return NULL;
30     }
31     log_info("PCM EQ Work Data space is enough:%d : %d", sizeof(g_work_buf), buf_size);
32
33     parm.LCoeff_OnChip = get_pcm_eq_tab(sr);
34     if (1 < channel) {
35         parm.RCoeff_OnChip = get_pcm_eq_tab(sr);
36     }
37     parm.LGain = 1 << 20;
38     parm.RGain = 1 << 20;
39     parm.SHI = 10;
40     parm.SHO = 8;
41     parm.channel = channel;
42     return pcm_eq_phy(obuf, &g_work_buf[0], &parm, ppsound);
43 }
```

1.4.2 相关函数

1.4.2.1 函数 void *link_pcm_eq_sound(void *p_sound_out, void *p_dac_cbuf, void **pp_effect, u32 sr, u32 channel)

该函数为串联音频链路的标准接口，实现响应算法的启动，并添加到音频链路上，其中参数：

- ① p_sound_out: 当前算法的前级音频通道句柄;
- ② p_dac_cbuf: 输出的音频 buff;
- ③ pp_effect: 存放当前音效算法控制句柄指针的指针;
- ④ sr: 当前输入音频的采样率;
- ⑤ channel: 声道数, ad14n 和 ac104n 只有单声道, 传1 即可;
- ⑥ 返回值: 启动之后最新的音频通道句柄;

1.4.2.2 函数 void *pcm_eq_api(void *obuf, u32 sr, u32 channel, void **ppsound)

该函数用于初始化 PCM_EQ 的相关参数，由函数 link_pcm_eq_sound()调用即可，需要关注的其内部参数的初始化：

- ① nSection: EQ 频点数(段数), 不可超过各采样率的 eq_filt 表中的段数, 范围[1, 10];
- ② channal: 陷波器带宽, 该值越小防啸叫效果越好, 但发声啸叫频点误检时音质会更差;

1.5 变音音效 (vo_pitch)

AD14N / AC104N SDK 中自带变音音效，可自行调整参数做到不同的效果。

1.5.1 可选音效

- VP_CMD_ROBOT: 平调机器音模式
- VP_CMD_ROBOT2: 变调机器音模式
- VP_CMD_PITCHSHIFT: 变调模式
- VP_CMD_RAP: RAP 模式
- VP_CMD_PITCHSHIFT2: 变声模式 2
- VP_CMD_RAP_REALTIME: 你说我唱模式
- VP_CMD_CARTOON: 卡通模式

各音效的参数配置详见 SDK;

1.5.2 配置参数说明

- VC_ENABLE_FLAG: 变声模块使能, 该变量为 1 时, 使能变声模块;
- EXTRA_DATA_SIZE: RAP 模式可以包含的音源长度, 单位是 short;
- VO_RAP_LOOPEN: RAP 模式是否 repeat 音源
- VO_RAP_COMPRESS_RATE: RAP 压缩比, 该值越大, RAP 音源时间压缩的越多;
- VP_DECAY_VAL: ECHO 模式的 decay 速度
- VP_HIS_LEN: ECHO 模式的 delay:复用其他模式的 Buf
- VP_BUFLen: 变音音效运算需要的 buff

根据所选模式以及参数决定 buff 大小, 可通过 ops->need_buf()接口获取大小;

```

vo_pitch_api.c x
268 void *vp_phy(void *obuf, VOICE_PITCH_PARA_STRUCT *pvp_parm, void **ppsound)
269 {
270     u32 buff_len, i;
271     VOICEPITCH_STUCT_API *ops;
272     log_info("voice pitch api\n");
273
274     ops = get_vopitch_context(); //获取变采样函数接口
275     buff_len = ops->need_buf(); //运算空间获取
276     log_info("yo pitch buff need len: %d\n", buff_len);
277     if (buff_len > sizeof(VP_BUFLen)) {
278         log_error("yo pitch buff is not enough big!\n");
279         return 0;
280     }

```

1.5.3 相关结构体及函数

1.5.3.1 结构体 VOICE_PITCH_PARA_STRUCT

该结构体用于配置各类变音音效，该结构体中部分成员的值只对特定的变音效果有效，其中成员：

- ① do_flag: 选择的音效，参考VP_CMD 枚举；
- ② samplerate: 输入音频的采样率，支持16K或24K；
- ③ noise_dc: 能量阈值，默认为2048，一些持续存在的噪声经过变声有可能听起来会更明显，所以该阈值是用来过滤是否有有效信号输入的情况。低于阈值则mute；
- ④ pitchrate: 音调高低，默认值128代表原始音调，小于128音调升高，大于128音调降低。建议范围[40, 256]；
- ⑤ midifile: RAP模式的输入音源，仅RAP模式有效；
- ⑥ midifile_len: RAP模式的输入音源大小，仅RAP模式有效；
- ⑦ callback: RAP模式的状态回调；
- ⑧ priv: 私有参数，一般传NULL；

1.5.3.2 函数 void *link_voice_pitch_sound(sound_out_obj *p_curr_sound, void *p_dac_cbuf, void **pp_effect, u32 cmd)

该函数为串联音频链路的标准接口，实现响应算法的启动，并添加到音频链路上，其中参数：

- ① p_sound_out: 当前算法的前级音频通道句柄;
- ② p_dac_cbuf: 输出的音频 buff;
- ③ pp_effect: 存放当前音效算法控制句柄指针的指针;
- ④ cmd: 选择的变音类型, 参考 VP_CMD 枚举;
- ⑤ 返回值: 启动之后最新的音频通道句柄;

1.5.3.3 函数 void *voice_pitch_api(void *obuf, u32 cmd, void **ppsound)

该函数用于初始化变音音效相关参数，由函数 link_voice_pitch_sound()调用即可；该函数内部会调用函数 vp_cmd_case(), 用于配置所选的变音音效以及其参数。

1.5.3.4 函数 rap_callback(void *priv, int pos)

该函数为 RAP 模式的回调函数，由解码器调用。其中的参数 pos 表示 RAP 模式的状态；

RAP 模式初始化后，pos 会处于 RAP_PREPARE 状态等待输入音源；

当输入音源大于所设置的 noise_dc 能量阈值后，会触发 RAP 模式，此时 pos 在会 PAR_START 状态；

当 RAP 播放结束后，pos 会在 RAP_END 状态循环。若需要重置 RAP 状态，需调用函数 rap_reopen()。

1.5.3.5 函数 void rap_reopen(void)

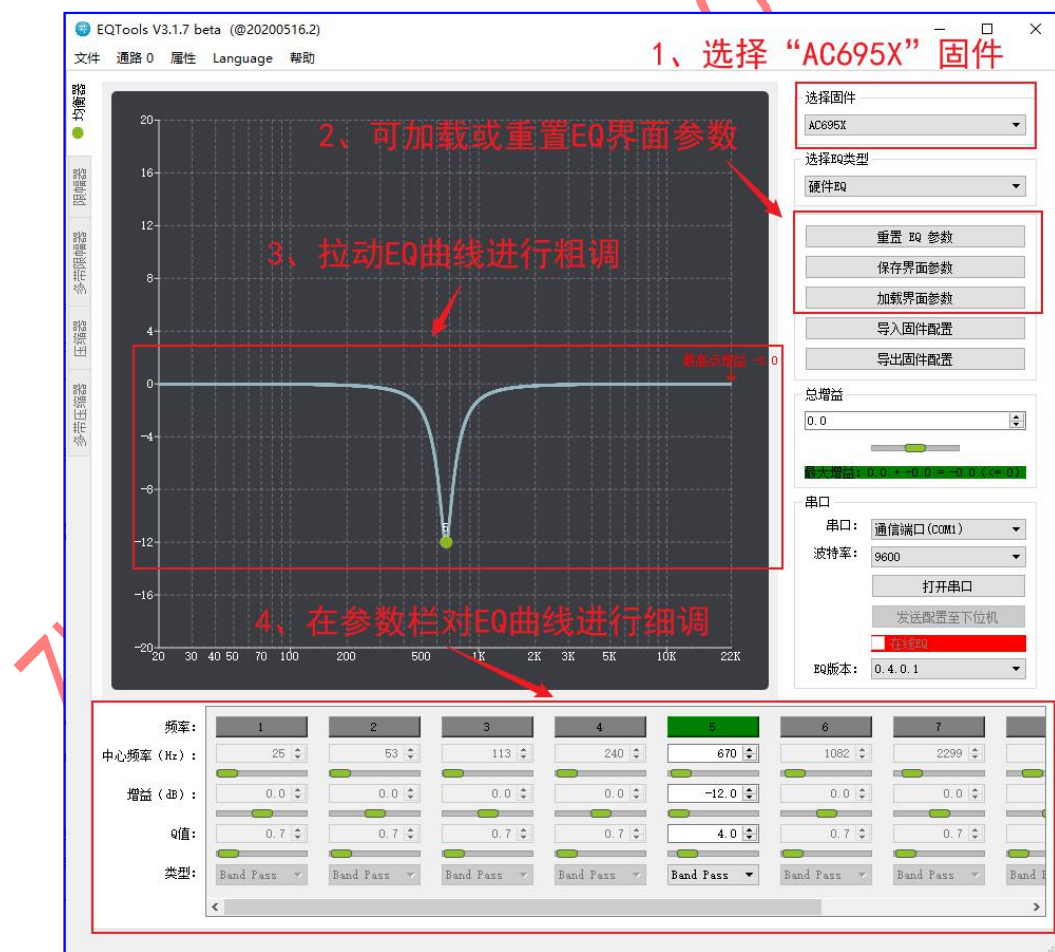
该函数用于 RAP 模式播放结束后，让 RAP 重回 RAP_PREPARE 状态。

第二章 EQ 工具使用说明

AD14N / AC104N 可以使用杰理 EQ 工具进行离线 EQ 调试，调试完成后可以导出当前的 EQ 配置，并加载到程序中验证效果。

2.1 EQ 工具使用介绍

1. 打开“EQ-3.1.7-beta-20200516.2.exe”工具并选择固件“AC695X”；
2. 选择需要的频点数，对于不需要的频点可以点击下方频率行对应跑按钮使其变灰；
3. 拉动 EQ 曲线进行粗调；
4. 粗调结束后，可在下方参数框对 EQ 曲线进行细调，可设置准确的中心频率、增益、Q 值以及滤波器类型；

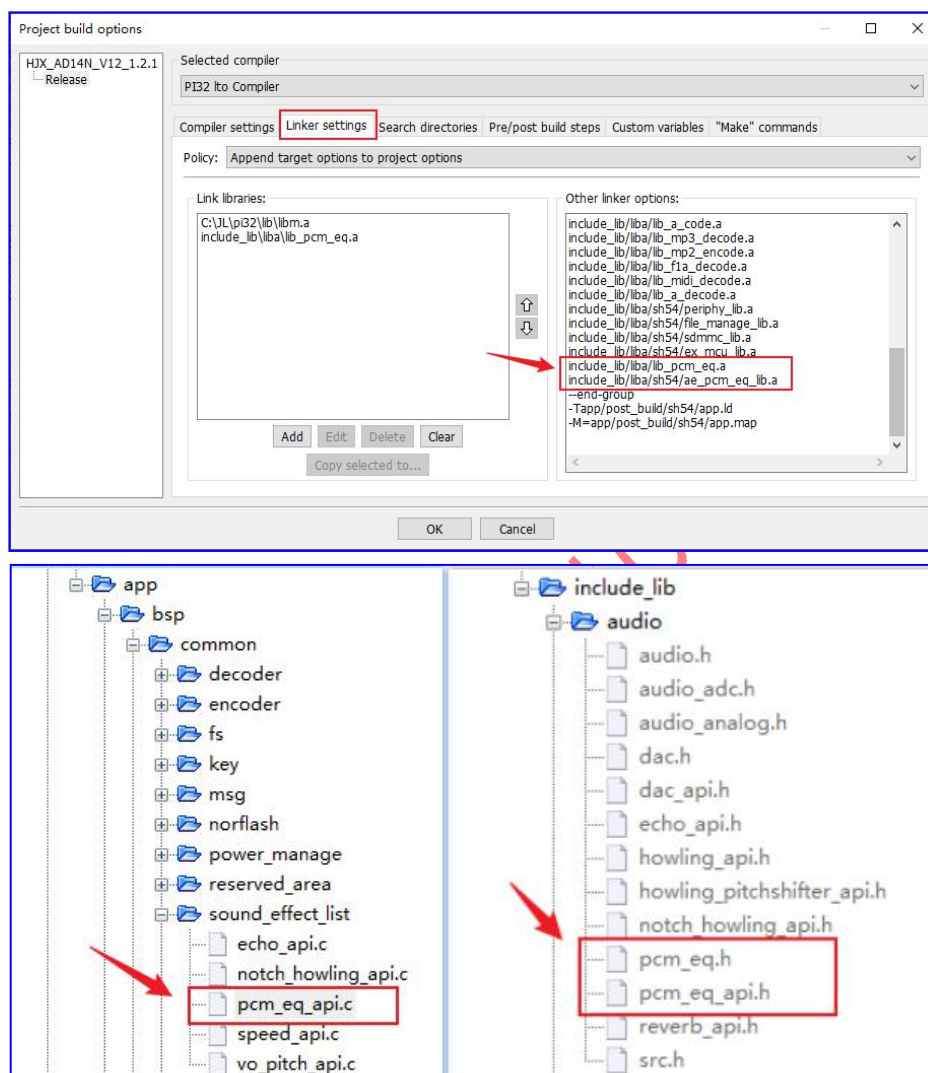


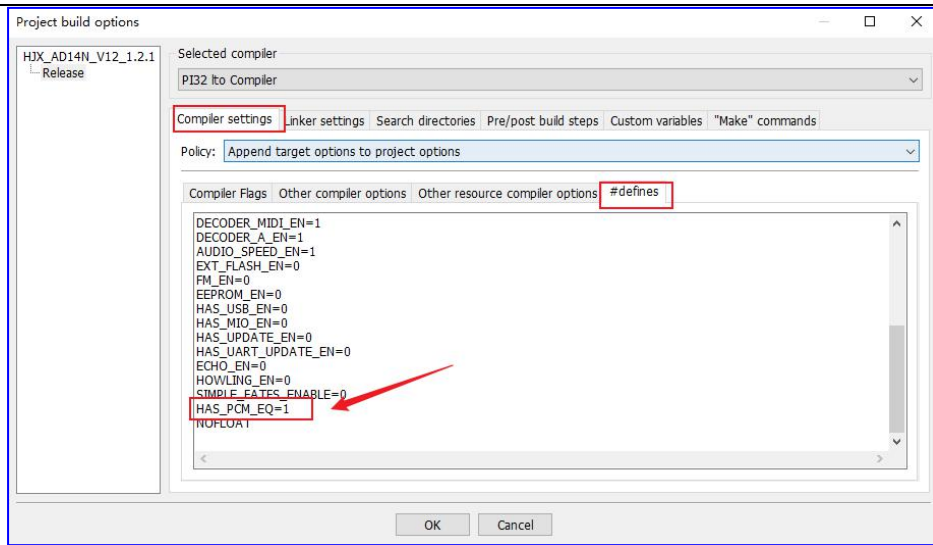
5. EQ 调节结束后，点击右边“保存界面参数”按钮对编辑的 EQ 界面进行保存；
6. 保存后，再点击“导出固件配置”按钮，工具根目录会生成新的“hw_eq_table.h”文件；



2.2 SDK 配置说明

1. SDK 工程中导入 pcm_eq 相关文件；





2. 打开 `pcm_eq_api.c` 文件, 并将 EQ 工具生成的 `hw_eq_table.h` 文件中的 eq 数据覆盖原本的 eq 数据;



3. 打开 `pcm_eq_api.c` 文件，根据 EQ 工具选择的频点数调整 `nSection` 值，并根据 `buf_size` 打印调整 `g_work_buf[]` 大小；

```

1  /* #include "asm_api.h" */
2  #include "pcm_eq.h"
3  /* #include "asm_api.h" */
4  #include "decoder_api.h"
5  #include "config.h"
6  #include "sound_effect_api.h"
7
8
9  #define LOG_TAG_CONST    NORM
10 #define LOG_TAG          "[normal]"
11 #include "debug.h"
12
13
14 static u32 g_work_buf[660 / 4] AT(.pcm_eq_data);
15
16 const int *get_pcm_eq_tab(u32 sr);
17
18 PCM_EQ_PARM parm;
19 void pcm_eq_api(void *obuf, u32 sr, u32 channel, void **p_sound)
20 {
21     memset(&parm, 0, sizeof(PCM_EQ_PARM));
22     parm.nSection = 10;
23     u32 buf_size = pcm_eq_buf_len(parm.nSection, channel);
24
25     if (sizeof(g_work_buf) < buf_size) {
26         log_info("PCM EQ Work Data space is not big enough:%d : %d", sizeof(g_work_buf), buf_size);
27         return NULL;
28     }
29     log_info("PCM EQ Work Data space is enough:%d : %d", sizeof(g_work_buf), buf_size);
30 }
31
32

```

4. 打开 `decoder_api.c` 文件，将 EQ 串入音频解码链路中，运行测试效果；需要注意 eq 所用的 ram 是否与其他资源复用了，建议不要同时开启多种音效；

```

165 #if AUDIO_SPEED_EN
166 //变速变调
167 if (dec_ctl & BIT_SPEED) {
168     p_curr_sound->effect = speed_api(cbuff_o, p_dec->sr, (void **) &p_next_sound);
169     if (NULL != p_curr_sound->effect) {
170         p_curr_sound->enable |= B_DEC_EFFECT;
171         p_curr_sound = p_next_sound;
172         p_curr_sound->obuf = cbuff_o;
173         p_next_sound = 0;
174         /* log_info("src init succ\n"); */
175     }
176 }
177 #endif
178
179 log_info("001 link_pcm_eq_sound!\n");
180 extern void link_pcm_eq_sound(void *p_sound_out, void *p_dec_buf, void **pp_effect, u32 sr, u32 channel);
181 p_curr_sound = link_pcm_eq_sound(p_curr_sound, cbuff_o, (void **)NULL, 32000, 1);
182 log_info("002 link_pcm_eq_sound!\n");
183
184 //硬件xxx
185 p_curr_sound->enable = 0;
186 if (32000 != p_dec->sr) {
187     p_curr_sound = link_src_sound(p_curr_sound, cbuff_o, (void **) &p_dec->src_effect, p_dec->sr, 32000);
188 } else {
189     void *src_tmp = src_hld_malloc();
190     src_reless((void **) &src_tmp);
191     /* log_info("xxx need xxx\n"); */
192 }

```