V3 架构应用指导说明

V1.2

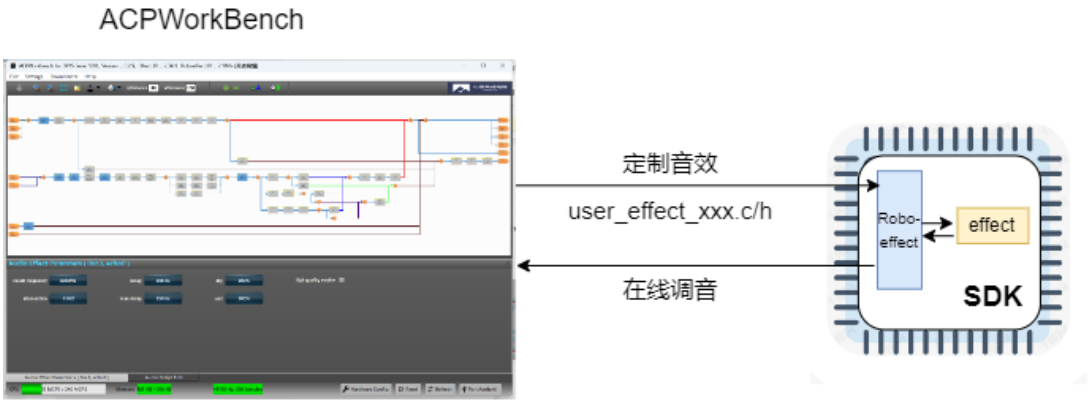
版本记录：

版本号	日期	作者	备注
V1.0	2023-10-18	Yangyu	初版
V1.1	2023-11-21	Yangyu	架构调整更新
V1.2	2023-12-15	Yangyu	格式及内容优化

目录

1. Roboeffect 介绍

Roboeffect 引擎是 V3 版本提出的新模型，提供所见即所得的可视化图形能力，只需简单的操作即可完成复杂的音效定制化开发。



1.1. roboeffect 库文件说明

Roboeffect 引擎核心代码文件：

/middle/roboeffect

+--- inc	
+--- roboeffect_api.h	roboeffect api 接口声明，以及若干引擎结构
+--- roboeffect_config.h	音效宏开关和音效接口声明
+--- src	
+--- roboeffect_api.c	包含音效属性的 template 表，音效 UI 定义等
+--- third_part_effect	第三方音效库
+--- user_defined_effect_api.h	第三方音效 API
+--- third_party_effects_data_gen.py	第三方离线音效库信息生成脚本
+--- libRoboeffect.a	
+--- roboeffect_library_info_v2.10.0.bin	离线音效库信息 for 上位机

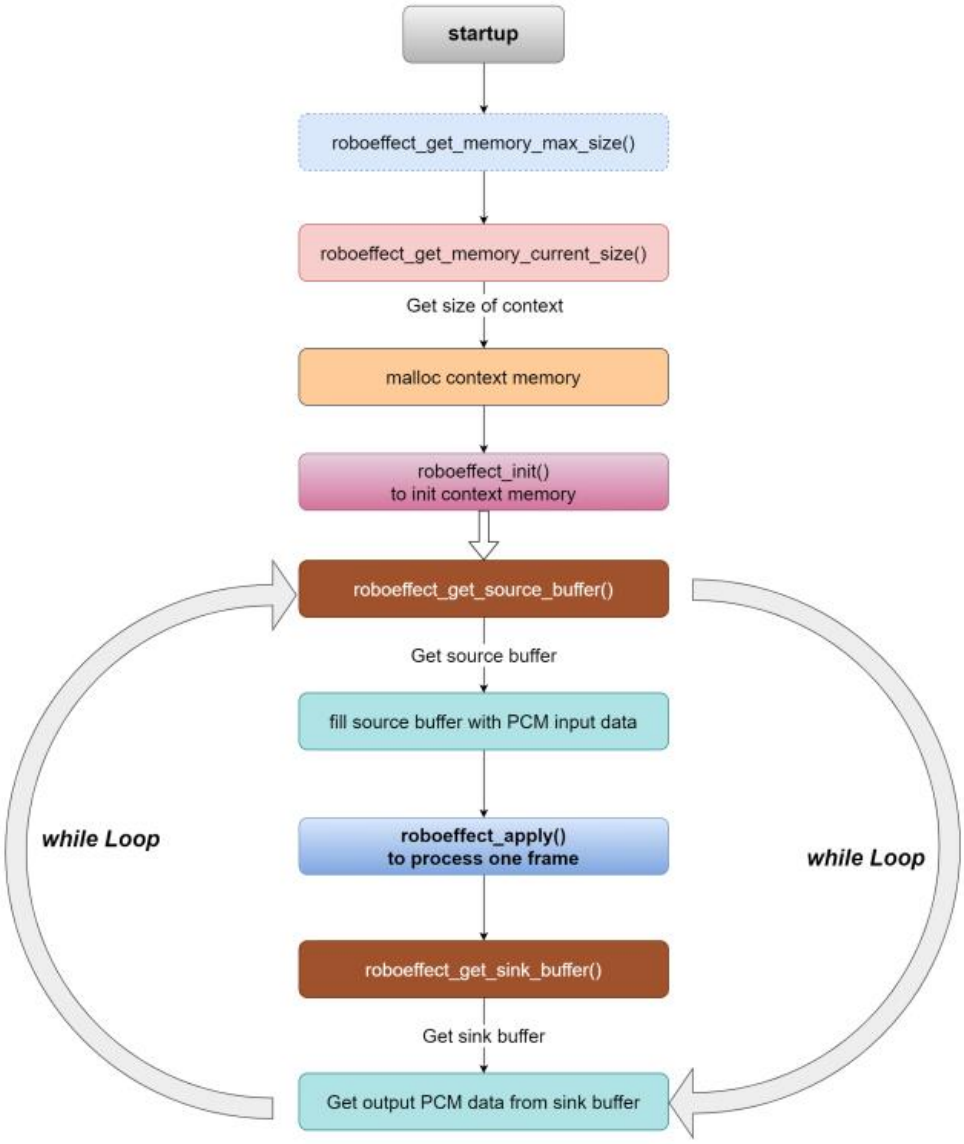
1.2. roboeffect API 介绍

Roboeffect 提供丰富的 API，使外部 SDK 可灵活调用操作整个引擎库。

API	说明
roboeffect_get_memory_max_size()	获取当前框图所有音效开启所需内存
roboeffect_get_memory_current_size()	获取当前框图默认开启的音效所需内存

roboeffect_get_effect_memory_size()	获取一个音效开启所需内存
roboeffect_init()	初始化
roboeffect_apply()	音效处理
roboeffect_get_source_buffer()	获取输入 source buffer
roboeffect_get_sink_buffer()	获取输出 sink buffer
roboeffect_enable_effect()	开启一个音效
roboeffect_enable_all_effects()	开启所有音效
roboeffect_get_effect_status()	获取一个音效的状态
roboeffect_set_effect_parameter()	设置一个音效的参数
roboeffect_get_effect_parameter()	获取一个音效的参数
roboeffect_get_parameter_number()	获取一个音效的参数数量
roboeffect_get_effect_name()	获取一个音效名
roboeffect_get_effect_version()	获取音效库版本
roboeffect_get_suit_frame_size()	根据当前框图中音效开启状态获取合适的帧长

1.3. roboeffect 工作流程



roboeffect 工作流程说明：

1. 调用 roboeffect_get_memory_max_size( ) 估算最大内存使用量 roboeffect_get_memory_max_size( ) 返回的是 roboeffect 使用的 context_memory 最大内存，按所有音效全开，以及 delay 长度计算 delay buffer 大小得出的值。如果应用中不需要所有音效全开，可以不使用此接口。
2. 调用 roboeffect_get_memory_current_size( ) 估算当前参数配置下内存使用量 roboeffect_get_memory_current_size( ) 返回的是根据当前音效参数表（user_effect_flow.c 中定义的 effect_property_for_display[]）计算得出的 context_memory 内存使用量。
3. 分配 roboeffect 运行所使用的 context_memory 内存 此步骤由当前应用所依托的平台决定，可以是动态分配的 malloc，也可以静态分配的内存数组。
4. 调用 roboeffect_init( ) 对 roboeffect 进行初始化 在分配的内存 context_memory 上初

始化 roboeffect

5. 使用 roboeffect_get_source_buffer( ) 得到 source buffer；使用 roboeffect_get_sink_buffer( ) 得到 sink buffer；source_id 和 sink_id 由 user_effect_flow.h 定义，需要对照 acpworkbench 进行区分。
6. apply roboeffect 循环 每一帧调用一次 roboeffect_apply( )，具体流程如下：
   a) 将输入数据填充到 source buffer，此数据可以是用外设 DMA 中输入，也可以是 audio core 中的 source 数据
   b) 调用 roboeffect_apply( ) 处理一帧音频数据
   c) 从 sink buffer 中取出处理完的数据

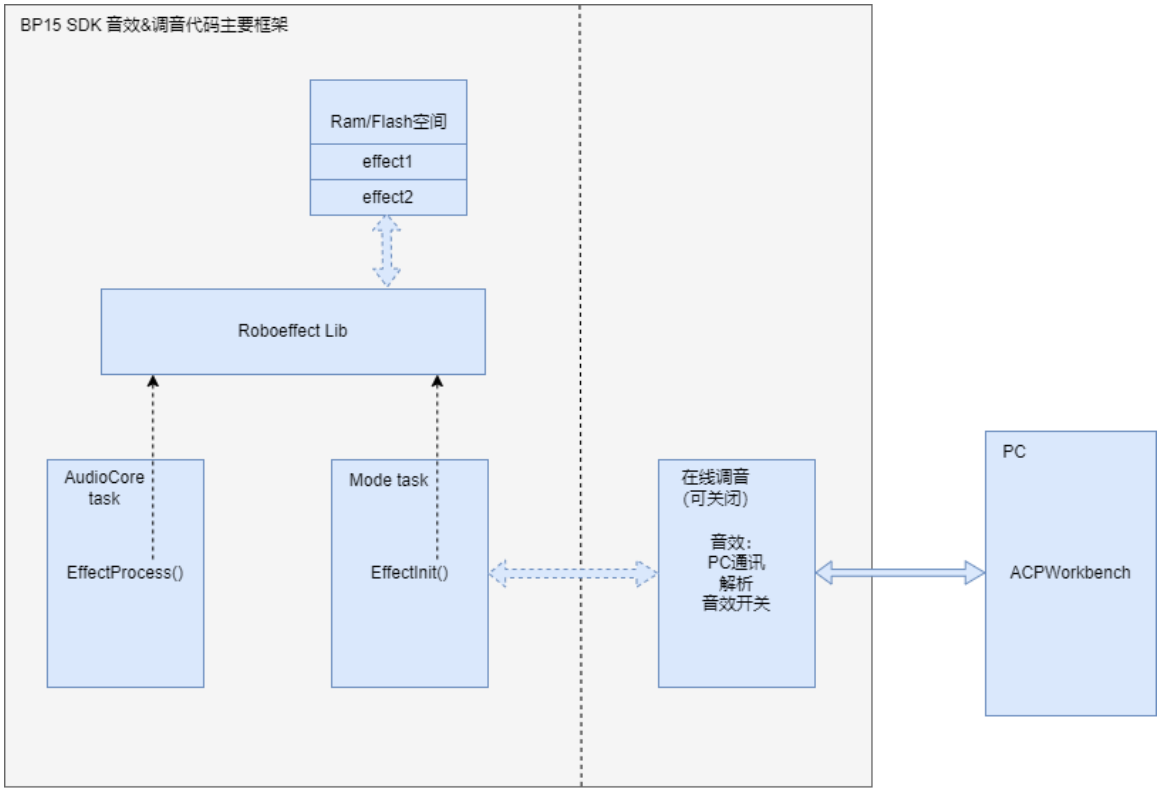
2. ACPWorkBench V3.x.x 版本介绍

可视化调音工具 ACPWorkbench 是一款可以实时绘制音效流， 实时调音的工具， 相比 ACPWorkbench V2 版本， 该版本从视觉和功能上有了直观的改变。 无论是熟悉山景 SDK 的用户还是刚刚接触的新用户， 都能受益于其直观的操作和快速的音效流定制。需注意 ACPWorkbench V3 版本不兼容 V2 版本。

更多细节可参考《ACPWorkbench-CHS.pdf》。

3. SDK 音效架构设计

SDK 音效和调音的软件设计架构如下图所示。



SDK 以 AudioCore 为音频流处理核心，以 Roboeffect 为音效处理核心，实现灵活多变的音效处理。将用户十分关注，需要经常修改的部分独立出来，方便进行二次开发。

3.1. 以 effect_mode 为核心

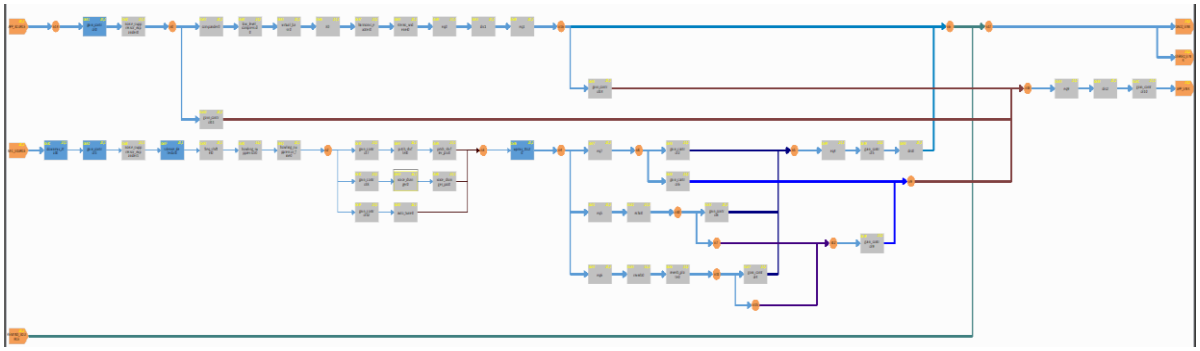
SDK 在设计的时候，将音效定制部分尽可能简单化、自动化，围绕 effect_mode 将**上位机导出的音效文件**通过 **effect_node.c** 与之深度绑定，定制音效只需简单修改 effect_node.c 即可完成自动初始化、音效处理以及在线调音等一系列动作。

```
typedef enum _EFFECT_MODE
{
    EFFECT_MODE_DEFAULT = 0,
    /**********mic node***************/
    EFFECT_MODE_MIC,
    /**********music node*************/
    EFFECT_MODE_MUSIC,
    /**********bypass node************/
    EFFECT_MODE_BYPASS,
    /**********hfp node***************/
    EFFECT_MODE_HFP_AEC,
    /**********karaoke node***********/
     EFFECT_MODE_HunXiang,
     EFFECT_MODE_DianYin,
     EFFECT_MODE_MoYin,
     EFFECT_MODE_HanMai,
     EFFECT_MODE_NanBianNv,
     EFFECT_MODE_NvBianNan,
     EFFECT_MODE_WaWaYin,

    EFFECT_MODE_COUNT,
     //User can add other effect mode
} EFFECT_MODE;
```

3.2. Roboeffect 音效文件

SDK 的一个音效框图在调音工具的展示如下：



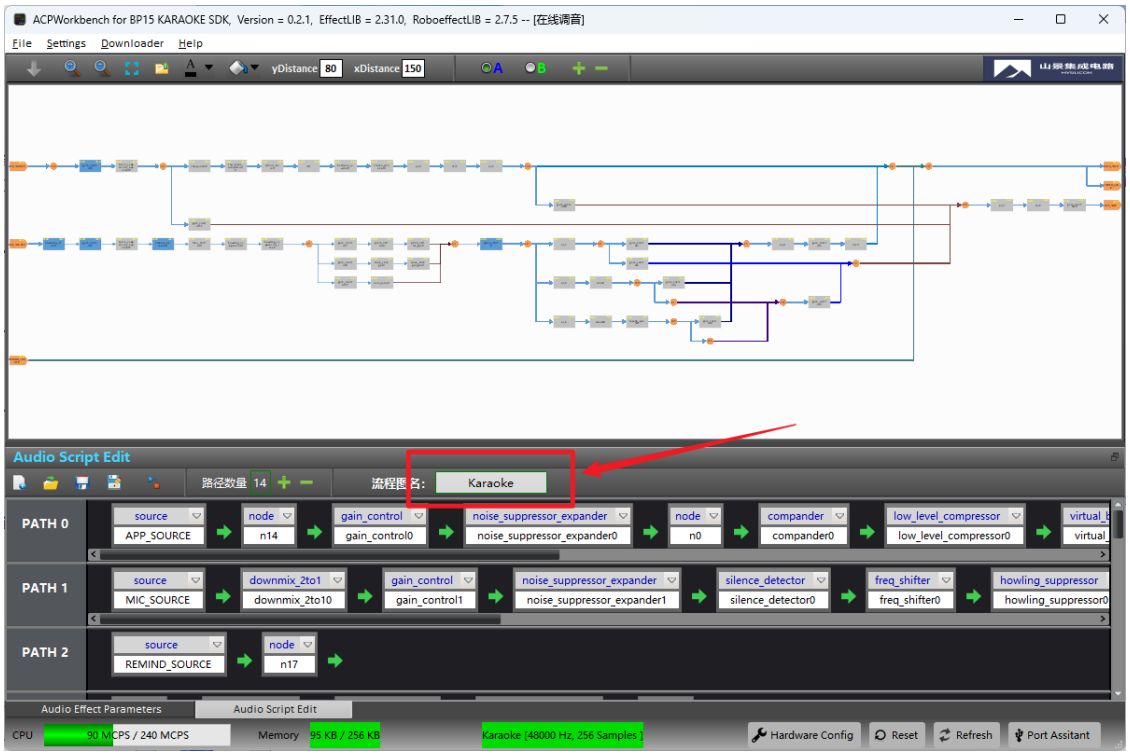
SDK 的音效处理由音效框图决定，根据该图会生成如下 C 和 H 代码文件：

如图，目前上位机导出的音效相关文件全部放在./app_src_components/audio/music_parameter/目录下。

3.2.1. 音效 flow 文件

音效 flow 文件（user_effect_flow_xxx.c/.h）由调音工具导出，主要包含设计完成的音效 flow 信息。

以 karaoke 模式为例，打开 karaoke 模式后连接调音工具，即可在下图标注位置中看到“Karaoke“字样，表示当前框图名是 Karaoke，在导出的 karaoke flow 文件中，所有结构的命名都是以 KARAOKE 为前缀。



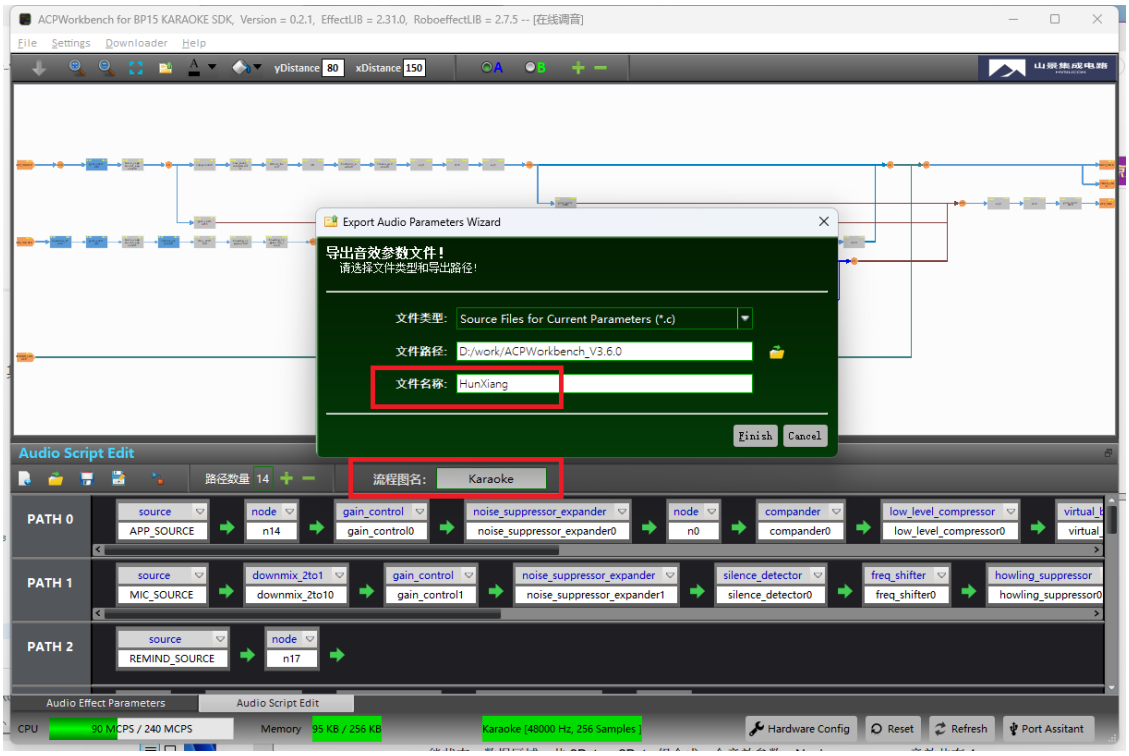
```c
/*******************************************************
 * @file    user_effect_flow_Karaoke.c
 * @brief   auto generated
 * @author  ACPWorkbench: 3.7.0
 * @version V1.1.0
 * @graphics: Karaoke
 * @suffix: Karaoke
 * @Created 2023-12-01T17:30:44
 * @copy; Shanghai Mountain View Silicon Technology Co.,Ltd. All rights reserved.
 *******************************************************/

#include "stdio.h"
#include "type.h"
#include "roboeffect_config.h"
#include "roboeffect_api.h"
#include "user_defined_effect_api.h"
#include "user_effect_flow_karaoke.h"

const unsigned char user_effects_script_Karaoke[] = {
0x03, 0x05, 0x02, 0x00, 0x07, 0x4b, 0x61, 0x72, 0x61, 0x6f, 0x6b, 0x65, 0x61, 0x75,
0x6e, 0x65, 0x20, 0x3d, 0x20, 0x61, 0x75, 0x74, 0x6f, 0x5f, 0x74, 0x75, 0x6e, 0x65,
```

3.2.2. 音效参数文件

音效参数文件（user_effect_param_xxx.c）的不同点在于，所有音效参数的结构都是以“**前缀 + flow 名 + 音效名**”组成，其中音效名即为导出时我们手动填写的命名。



```
/*****************************************************
 * @file    user_effect_param_HunXiang.c
 * @brief   auto generated
 * @author  ACPWorkbench: 3.5.3
 * @version V1.1.0
 * @Created 2023-09-08T19:46:22
 * @Graphics Name Karaoke
 * @copy; Shanghai Mountain View Silicon Technology Co.,Ltd. All rights reserved.
 *****************************************************/

#include "stdio.h"
#include "type.h"

const unsigned char user_effect_parameters_Karaoke_HunXiang[] = {
0xbl, 0x04, /*total data length*/

0x02, 0x1f, 0x00, /*Effect Version*/

0x81, /*gain_control0*/
0x05,/*length*/
0x01, /*enable*/
0x00, 0x00, /*mute*/
```

3.2.3. effect_node.c

effect_node.c 的主要作用就是在 SDK 和上位机导出的音效参数中间搭起一座桥梁，使得 SDK 能够知道该如何去选择正确的音效参数，以及拿到正确的 source、sink 和音效地址。

在 effect_node.c 中，我们需要手动指定当前音效下的 effect_mode，音效地址映射，source 和 sink 的映射。这样 SDK 就能够把自身的资源同音效框图绑定起来，SDK 可以更加方便的开启和调整音效。

以 Karaoke 为例，Karaoke 框图下默认有 7 组音效参数，分别对应 **EFFECT_MODE_HunXiang** 至 **EFFECT_MODE_WaWaYin** 7 个 effect mode。

```
//ROBOEFFECT effect ID 通过这个ID来搜索匹配
.effect_id     = EFFECT_MODE_HunXiang ,
//该框图下面有7个音效
.effect_id_count = EFFECT_MODE_WaWaYin - EFFECT_MODE_HunXiang + 1,
```

effect_para 中也存在 7 组与实际 effect mode 相对应，同一时间只会加载其中一组音效参数。

```
//ROBOEFFECT effect 音效地址映射
.effect_addr =
{
    .REVERB_ADDR = KARAOKE_reverb0_ADDR,
    .REVERBPLATE_ADDR = KARAOKE_reverb_plate0_ADDR,
    .ECHO_ADDR = KARAOKE_echo0_ADDR,
    .SILENCE_DETECTOR_ADDR = KARAOKE_silence_detector0_ADDR,
    .VOICE_CHANGER_ADDR = KARAOKE_voice_changer0_ADDR,
    .APP_SOURCE_GAIN_ADDR = KARAOKE_gain_control0_ADDR,
    .MIC_SOURCE_GAIN_ADDR = KARAOKE_gain_control1_ADDR,
    .REMIND_SOURCE_GAIN_ADDR = KARAOKE_gain_control13_ADDR,
},

//ROBOEFFECT effect SOURCE映射
.audioeffect_source =
{
    .mic_source = KARAOKE_SOURCE_MIC_SOURCE,
    .app_source = KARAOKE_SOURCE_APP_SOURCE,
    .remind_source = KARAOKE_SOURCE_REMIND_SOURCE,
    .rec_source = KARAOKE_SOURCE_REC_SOURCE,
    .usb_source = KARAOKE_SOURCE_USB_SOURCE,
    .i2s_mix_source = KARAOKE_SOURCE_I2S_MIX_SOURCE,
    .linein_mix_source = KARAOKE_SOURCE_LINEIN_MIX_SOURCE,
},

//ROBOEFFECT effect SINK映射
.audioeffect_sink =
{
    .dac0_sink = KARAOKE_SINK_DAC0_SINK,
    .app_sink = KARAOKE_SINK_APP_SINK,
    .stereo_sink = KARAOKE_SINK_STEREO_SINK,
    .rec_sink = KARAOKE_SINK_REC_SINK,
    .i2s_mix_sink = KARAOKE_SINK_I2S_MIX_SINK,
    .spdif_sink = KARAOKE_SINK_SPDIF_SINK,
},
```

音效地址、source 和 sink 的映射我们只需手动将其正确匹配即可，SDK 在真正使用的时候会通过 get_audioeffect_addr() 、 AudioCoreSourceToRoboeffect() 、AudioCoreSinkToRoboeffect()三个 API 来自动查找。

3.3. Roboeffect Init

3.3.1. 选择正确的 effect mode

由于 source 和 sink 的缓存 buffer 都在 roboeffect 中集中管理，因此在 ModeCommonInit()中，需要首先执行 AudioEffectInit ()来完成 roboeffect 相关的初始化。

根据当前选择的音效，会判断并找到正确的音效 flow 和与之匹配的音效参数。目前该部分不需要手动做任何修改，SDK 会根据音效参数路径下的 effect_node.c 文件中的信息自动查找加载。

```
bool AudioEffectInit()
{
    if(AudioCore.Audioeffect.effect_addr)
    {
        uint8_t *params = AudioCore.Audioeffect.user_effect_parameters + 5;
        uint16_t data_len = *(uint16_t *)AudioCore.Audioeffect.user_effect_parameters - 5;
        uint8_t len = 0;
        while(data_len)
        {
            if(*params == AudioCore.Audioeffect.effect_addr)
            {
                params += 2;
                *params = AudioCore.Audioeffect.effect_enable;
                break;
            }
            else
            {
                params++;
                len = *params;
                params += (len + 1);
                data_len -= (len + 1);
            }
        };

        DBG("Audioeffect ReInit:0x%x\n", AudioCore.Audioeffect.effect_addr);
    }
    else
    {
        if(AudioCore.Audioeffect.user_effect_parameters)
        {
            //先释放资源
            osPortFree(AudioCore.Audioeffect.user_effect_parameters);
        }

        AUDIOEFFECT_EFFECT_PARA *para = get_user_effect_parameters(mainAppCt.EffectMode);

        AudioCore.Audioeffect.effect_count = para->user_effect_list->count + 0x80;
        AudioCore.Audioeffect.user_effect_steps = para->user_effect_steps;
        AudioCore.Audioeffect.user_effects_script = para->user_effects_script;
        AudioCore.Audioeffect.user_effects_script_len = para->get_user_effects_script_len();
        AudioCore.Audioeffect.user_effect_list = para->user_effect_list;
        AudioCore.Audioeffect.user_effect_parameters = osPortMalloc(get_user_effect_parameters_len(para->user_effect_parameters) * sizeof(uint8_t));
        memcpy(AudioCore.Audioeffect.user_effect_parameters, para->user_effect_parameters,get_user_effect_parameters_len(para->user_effect_parameters) * sizeof(uint8_t
        AudioCore.Audioeffect.user_module_parameters = para->user_module_parameters;
        AudioCore.Audioeffect.audioeffect_frame_size = para->user_effect_list->frame_size;
```

3.3.2. 内存申请

roboeffect 正常运行需要的所有内存都在这一步进行申请，我们只需按照roboeffect_get_memory_current_size()获取到的大小申请内存即可。

```
/**
 * malloc context memory
*/
if(AudioCore.Audioeffect.audioeffect_memory_size < xPortGetFreeHeapSize())
{
    AudioCore.Audioeffect.context_memory = osPortMallocFromEnd(AudioCore.Audioeffect.audioeffect_memory_size);
    if(AudioCore.Audioeffect.context_memory == NULL)
    {
        return FALSE;
    }
    /**
     * initial roboeffect context memory
    */
    if(ROBOEFFECT_ERROR_OK != roboeffect_init(AudioCore.Audioeffect.context_memory,
                                              AudioCore.Audioeffect.audioeffect_memory_size,
                                              AudioCore.Audioeffect.user_effect_steps,
                                              AudioCore.Audioeffect.user_effect_list,
                                              AudioCore.Audioeffect.user_effect_parameters) )
    {
        DBG("roboeffect_init failed.\n");
        return FALSE;
    }
    else
    {
        DBG("roboeffect_init ok.\n");
        AudioCore.Audioeffect.effect_addr = 0;
        AudioEffect_GetAudioEffectMaxValue();

        ////Audio Core & Audioeffect音量配置
        SystemVolSet();
    }
}
else
{
    DBG("****************************************\n");
    DBG("Error:memory is not enough!!!\n");
    DBG("malloc:%ld, leave:%ld\n", AudioCore.Audioeffect.audioeffect_memory_size, xPortGetFreeHeapSize());
    DBG("****************************************\n");
    return FALSE;
}

roboeffect_prot_init();
return TRUE;
```

3.3.3. roboeffect_init()

roboeffect_init()会根据我们提供的参数来进行其核心引擎的初始化。

3.4. Source&Sink Init

V3 架构中，source 和 sink 的缓存 buffer 统一在 roboeffect 内部管理，因此在外部我们不再需要另外申请 buffer。在 source 和 sink 初始化的时候我们做如下操作即可。这一步也会自动完成，无需特别关注。

```
//Source
Source->PcmInBuf = roboeffect_get_source_buffer(AudioCore.Roboeffect.context_memory, AudioCoreSourceToRoboeffect(Index));
//Sink
Sink->PcmOutBuf = roboeffect_get_sink_buffer(AudioCore.Roboeffect.context_memory, AudioCoreSinkToRoboeffect(Index));
```

3.5. Effect Process

V3 版本的 effect process 函数中，除去必要的逻辑判断之外，我们无需再做多余的操作，直接执行下面函数即可，有关音效实际的执行和 downmix 等操作全部在其中完成。

```
roboeffect_apply();
```

除此之外，我们还提供如下函数来方便 debug，该函数不包含任何 roboeffect 的动作，仅做 source buffer 到 sink buffer 的 copy。

```
AudioBypassProcess()
```

3.6. 在线调音

在线调音的实现基本都在 communication.c 中。该部分逻辑本质上是对《固件与用户应用程序通信协议 V3.x.x.pdf》的实现，感兴趣可以进一步详细阅读。

4. 快速定制音效

V3 版本音效处理的核心是音效框图 + 音效参数，两者互相搭配来实现理想的音效运行效果。下面音效的定制说明均以 Karaoke 为例。

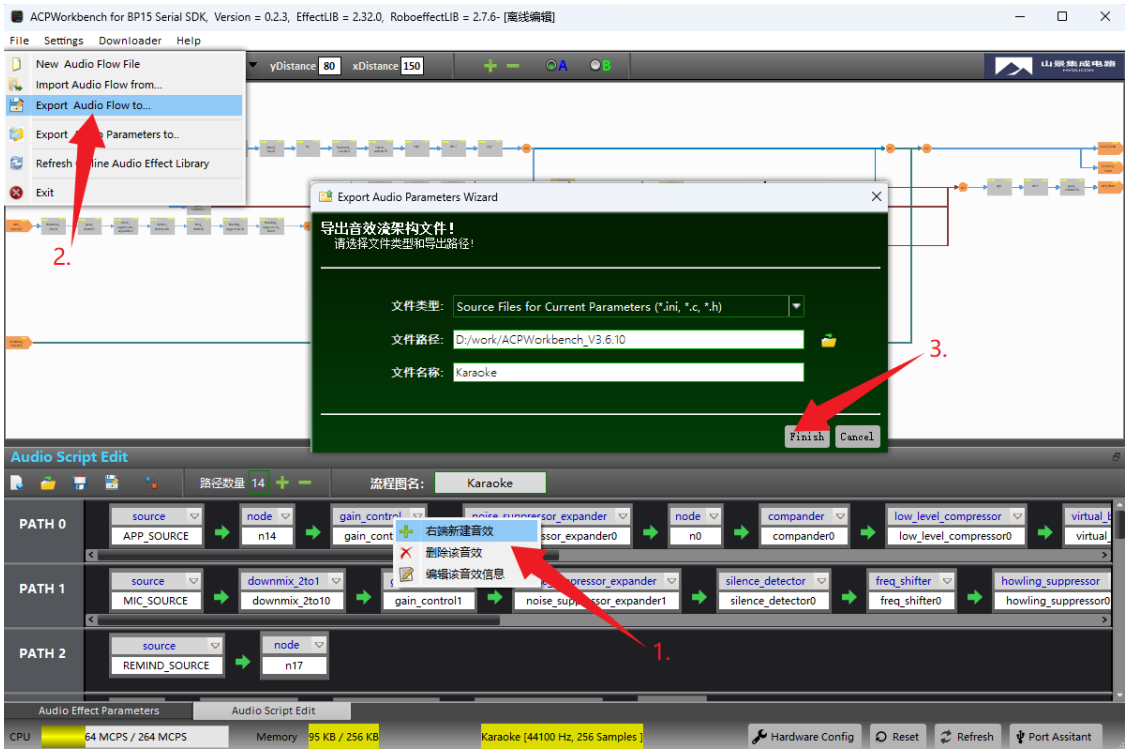
4.1. 音效宏开关

SDK 中对于各种音效用宏进行了管理，当某些音效确定不会使用时，将 roboeffect_config.h 文件中对应音效的宏配置为“0”，这样这部分代码以及相关的音效库函数均不会被包含到 SDK 代码中来，可以减少代码量。

4.2. 定制框图

在使用 SDK 进行音效定制时，我们会经常要进行框图架构的调整，注意在每次确定好框图之后，除了音效框图文件之外，还需要从调音工具导出音效参数到 SDK 进行整合。

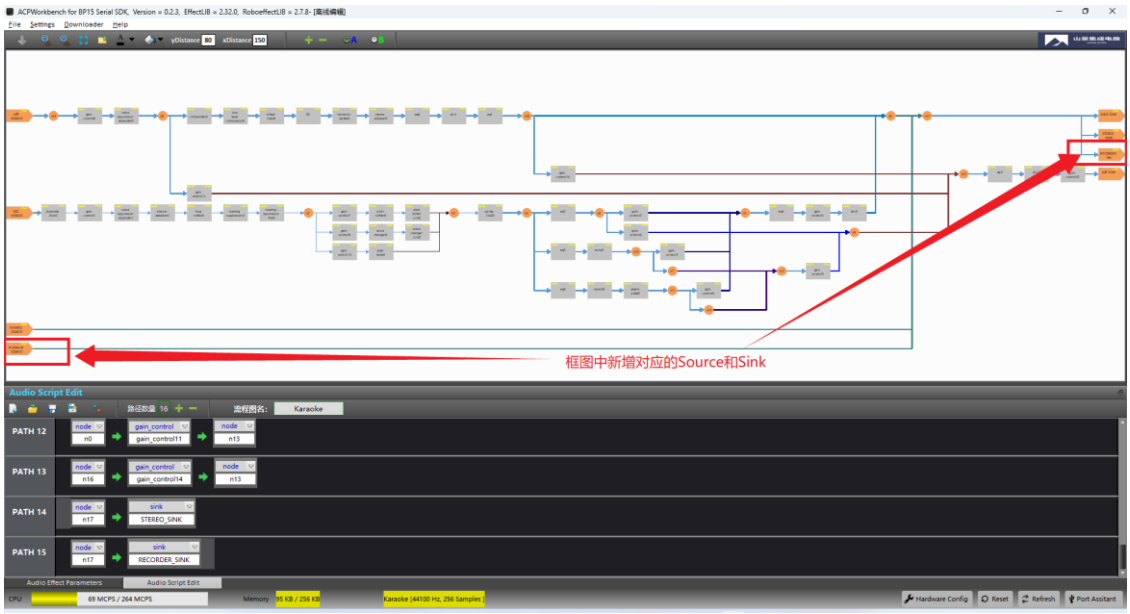
4.2.1. 增加/删除音效



将生成.c 和.h 文件替换至 SDK 目录(./app_src/components/audio/music_parameter)下的对应路径，导入对应文件到 SDK 后，请参考 **4.3** 小节的流程继续更新音效参数。如对个别音效有更多需求，还需参考 **4.4** 小节更新音效地址映射。

4.2.2. 新增/删掉输入输出源

以 Karaoke 模式下增加录音功能为例，对应打开宏 **CFG_FUNC_RECORDER_EN**。



然后按照 4.2.1 小节的流程导出框图文件到 SDK 对应目录下。

注 意 ： 框 图 中 source 和 sink 改 动 之 后 一 定 要 更 新./app_src/components/audio/music_parameter/user_effect_parameter.c/.h 如下部分代码。

```
typedef struct _AUDIOEFFECT_SOURCE_NUM
{
    uint8_t mic_source;     //MIC_SOURCE_NUM    //麦克风通路
    uint8_t app_source;     //APP_SOURCE_NUM    //app主要音源通道
    uint8_t remind_source;  //PLAYBACK_SOURCE_NUM //提示音使用固定混音通道
    uint8_t rec_source;     //REMIND_SOURCE_NUM //录音回放通道
    uint8_t usb_source;     //USB_SOURCE        //USB MIX通道
    uint8_t i2s_mix_source; //I2S_MIX_SOURCE    //I2S MIX通道
    uint8_t linein_mix_source; //LINEIN_MIX_SOURCE_NUM    //LINE IN MIX通道
} AUDIOEFFECT_SOURCE_NUM;

typedef struct _AUDIOEFFECT_SINK_NUM
{
    uint8_t dac0_sink;      //AUDIO_DAC0_SINK_NUM    //主音频输出在audiocore Sink中的通道,
    uint8_t app_sink;       //AUDIO_APP_SINK_NUM
    uint8_t stereo_sink;    //AUDIO_STEREO_SINK_NUM    //模式无关Dac0之外的 立体声输出
    uint8_t rec_sink;       //AUDIO_RECORDER_SINK_NUM    //录音通道
    uint8_t i2s_mix_sink;   //AUDIO_I2S_MIX_OUT_SINK_NUM //I2S MIX OUT通道
    uint8_t spdif_sink;     //AUDIO_SPDIF_SINK_NUM
} AUDIOEFFECT_SINK_NUM;
```

```
uint8_t AudioCoreSourceToRoboeffect(int8_t source)
{
    AUDIOEFFECT_EFFECT_PARA_TABLE *param = GetCurEffectParaNode();

    switch (source)
    {
        case MIC_SOURCE_NUM:
            return param->audioeffect_source.mic_source;
        case APP_SOURCE_NUM:
            return param->audioeffect_source.app_source;
        case REMIND_SOURCE_NUM:
            return param->audioeffect_source.remind_source;
        case PLAYBACK_SOURCE_NUM:
            return param->audioeffect_source.rec_source;
        case I2S_MIX_SOURCE_NUM:
            return param->audioeffect_source.i2s_mix_source;
        case USB_SOURCE_NUM:
            return param->audioeffect_source.usb_source;
        case LINEIN_MIX_SOURCE_NUM:
            return param->audioeffect_source.linein_mix_source;
        default:
            break;//handle error
    }
    return AUDIOCORE_SOURCE_SINK_ERROR;
}

uint8_t AudioCoreSinkToRoboeffect(int8_t sink)
{
    AUDIOEFFECT_EFFECT_PARA_TABLE *param = GetCurEffectParaNode();

    switch (sink)
    {
        case AUDIO_DAC0_SINK_NUM:
            return param->audioeffect_sink.dac0_sink;
#if (defined(CFG_APP_BT_MODE_EN) && (BT_HFP_SUPPORT == ENABLE)) || defined(C
        case AUDIO_APP_SINK_NUM:
            return param->audioeffect_sink.app_sink;
#endif

#ifdef CFG_FUNC_RECORDER_EN
        case AUDIO_RECORDER_SINK_NUM:
            return param->audioeffect_sink.rec_sink;
#endif
#if defined(CFG_RES_AUDIO_I2SOUT_EN)
```

以及 effect_node.c 中如下映射。

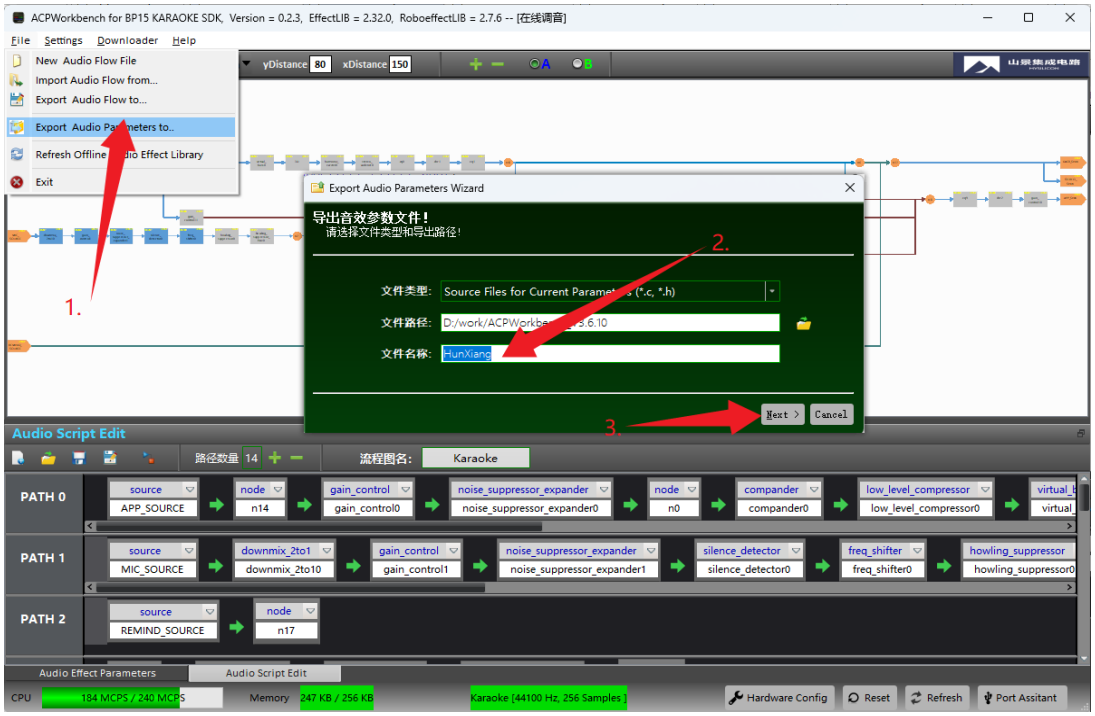
```
//ROBOEFFECT effect SOURCE映射
.audioeffect_source =
{
    .mic_source = KARAOKE_SOURCE_MIC_SOURCE,
    .app_source = KARAOKE_SOURCE_APP_SOURCE,
    .remind_source = KARAOKE_SOURCE_REMIND_SOURCE,
    .rec_source = KARAOKE_SOURCE_REC_SOURCE,
    .usb_source = KARAOKE_SOURCE_USB_SOURCE,
    .i2s_mix_source = KARAOKE_SOURCE_I2S_MIX_SOURCE,
    .linein_mix_source = KARAOKE_SOURCE_LINEIN_MIX_SOURCE,
},

//ROBOEFFECT effect SINK映射
.audioeffect_sink =
{
    .dac0_sink = KARAOKE_SINK_DAC0_SINK,
    .app_sink = KARAOKE_SINK_APP_SINK,
    .stereo_sink = KARAOKE_SINK_STEREO_SINK,
    .rec_sink = KARAOKE_SINK_REC_SINK,
    .i2s_mix_sink = KARAOKE_SINK_I2S_MIX_SINK,
    .spdif_sink = KARAOKE_SINK_SPDIF_SINK,
},
```

同理，**删掉**输入输出源需修改删掉上述位置相应部分的代码逻辑。

4.3. 定制音效参数

当框图确定之后，我们还需要按如下步骤导出音效参数到 SDK。



更新已有音效参数，将生成文件替换至 SDK 目录(./app_src/components/audio/music_parameter/)重新编译烧录即可。

如果是新增音效，导入音效参数文件到 SDK 后，需参考已有音效修改 SDK 如下部分代码。

1. ctrlvars.h 中 **EFFECT_MODE** 新增 SDK 音效名；
2. 对应音效路径下的 effect_node.c 中更新结构 **effect_para**、**音效地址映射**、**source 映射**和 **sink 映射**。

4.4. 更新 effect_node.c

在 3.1.3 小节中，我们对 effect_node.c 做了简单的介绍，effect_node.c 是 SDK 对接 roboeffect 引擎以及音效的核心文件。

在完成音效的定制之后，我们还需更新 effect_node.c 中的 effect_mode 和一些映射信息，如果是新增音效，参考已有的框架复制后同步实际音效修改即可。

4.5. 第三方音效添加

请参考《roboeffect_user_manual_publish.pdf》文档。

5. SDK 音效控制

5.1. 音效控制相关宏

SDK 通过音效功能宏来控制音效，便于用户开关宏来调试音效，量产或调试时开关 app_config.h 中的部分宏来节省代码和内存或者使能部分特殊功能，具体见下表。

宏	说明
CFG_FUNC_AUDIO_EFFECT_EN	音效宏总开关
CFG_FUNC_AUDIO_EFFECT_ONLINE_TUNING_EN	在线调音宏开关
CFG_FUNC_EFFECT_BYPASS_EN	Bypass 音效，用于音频指标测试
CFG_FUNC_MUSIC_EQ_MODE_EN	Music EQ mode 功能
CFG_FUNC_MIC_KARAOKE_EN	Karaoke 模式
CFG_FUNC_MIC_TREB_BASS_EN	Karaoke Mic 高低音调节功能
CFG_FUNC_MUSIC_TREB_BASS_EN	Karaoke Music 高低音调节功能
CFG_FUNC_SHUNNING_EN	Karaoke 模式闪避功能

5.2. 音效控制相关代码文件

SDK 中的音效和调音相关文件如下表。

代码文件	说明
communication.c/communication.h	在线调音功能代码
ctrlvars.c/ctrlvars.h	音频硬件通路的数据结构；变量初始化
audio_vol.c/audio_vol.h	音量相关控制
audio_effect_process.h	音效处理
user_effect_parameter.c/user_effect_parameter.h	SDK 自定义若干音效控制接口
components/audio/music_parameter/xxx/effect_node.c	音效框图中间描述文件，专用于 SDK 做接口适配

5.3. 音效控制接口

为了更好的使用 roboeffect 引擎库，SDK 在 roboeffect 基础 API 之上还定制了如下接口函数，可以更加方便快捷的控制内部音效。

API 函数	说明
AudioEffect_GainControl_Get()	获取指定 gain 的大小
AudioEffect_GainControl_Set()	设置指定 gain
AudioEffect_effect_status_Get()	获取指定音效开关状态
AudioEffect_effect_enable()	开关指定音效
get_audioeffect_addr ()	获取指定音效地址

更多接口函数及细节请参考 user_effect_parameter.c/h。

5.4. 音效控制示例

5.4.1. 获取音效地址

想要控制某个音效，我们首先需要知道该音效位于音效框图中的位置。SDK 有对应如下接口来快速获取音效地址。

```
uint8_t get_audioeffect_addr(AUDIOEFFECT_EFFECT_TYPE effect_name)
{
    AUDIOEFFECT_EFFECT_PARA_TABLE *param = GetCurEffectParaNode();
    uint8_t addr = 0;

    switch(effect_name)
    {
        case MUSIC_EQ:
            addr = param->effect_addr.MUSIC_EQ_ADDR;
            break;
        case MIC_EQ:
            addr = param->effect_addr.MIC_EQ_ADDR;
            break;
        case REVERB:
            addr = param->effect_addr.REVERB_ADDR;
            break;
        case REVERBPLATE:
```

这里我们并没有真正的去指定每个音效的地址，事实上在 effect_node.c 中已经完成了音效地址的映射匹配，这里只是查找返回。

同理，用户可基于此方便快捷的新增所需。

```
const ROBOEFFECT_EFFECT_PARA_TABLE karaoke_node =
{
    //ROBOEFFECT effect ID 通过这个ID来搜索匹配
    .effect_id    = EFFECT_MODE_HunXiang ,
    //该框图下面有7个音效
    .effect_id_count = EFFECT_MODE_WaWaYin - EFFECT_MODE_HunXiang + 1,

    //ROBOEFFECT effect 音效地址映射
    .effect_addr =
    {
        .REVERB_ADDR = KARAOKE_reverb0_ADDR,
        .REVERBPLATE_ADDR = KARAOKE_reverb_plate0_ADDR,
        .ECHO_ADDR = KARAOKE_echo0_ADDR,
        .SILENCE_DETECTOR_ADDR = KARAOKE_silence_detector0_ADDR,
        .VOICE_CHANGER_ADDR = KARAOKE_voice_changer0_ADDR,
        .APP_SOURCE_GAIN_ADDR = KARAOKE_gain_control0_ADDR,
        .MIC_SOURCE_GAIN_ADDR = KARAOKE_gain_control1_ADDR,
        .REMIND_SOURCE_GAIN_ADDR = KARAOKE_gain_control13_ADDR,
    },
```

5.4.2. 开关音效

```
case MSG_VOCAL_CUT:
    APP_DBG("MSG_VOCAL_CUT\n");
    AudioEffect_effect_enable(VOCAL_CUT, !AudioEffect_effect_status_Get(VOCAL_CUT));
```

SDK 对于开关音效的操作已经特别定义专门的接口函数，我们只需调用**AudioEffect_effect_enable()**即可。

```
void AudioEffect_effect_enable(AUDIOEFFECT_EFFECT_TYPE type, uint8_t enable)
{
    AudioCore.Audioeffect.effect_addr = get_audioeffect_addr(type);
    AudioCore.Audioeffect.effect_enable = enable;

    MessageContext msgSend;
    msgSend.msgId = MSG_EFFECTREINIT;
    MessageSend(GetMainMessageHandle(), &msgSend);
}
```

5.4.3. 调整音效参数

由于每个音效的音效参数不同，因此 SDK 没有再另外去定义通用的音效参数接口函数，这里以音量调节为例，列举下音效参数调节的规范操作。

```
void AudioEffect_GainControl_Set(uint8_t node, uint16_t gain)
{
    if(node < 0x81 || node > 0xfb)检查音效地址是否合法
    {
        return;
    }
    if(AudioCore.Audioeffect.context_memory == NULL) roboeffect引擎是否初始化
        return;

    int16_t param = gain;  更新参数至音效
    roboeffect_set_effect_parameter(AudioCore.Audioeffect.context_memory, node, 1, &param);
    AudioEffect_update_local_params(node, 1, &param, 2); update本地参数

    gCtrlVars.AutoRefresh = node;
}                                                        通知上位机更新
```

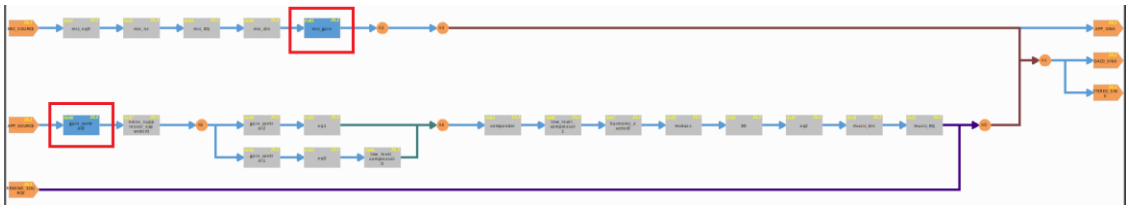
如果不知道想要修改的参数索引号，可参考《audio_effects_overview.pdf》。如果想要同时设置多个参数，则建议一次性刷新全部参数，更改上述部分逻辑如下。

```
调整传入参数格式
GainControlUnit param = {0, 0};
roboeffect_set_effect_parameter(AudioCore.Audioeffect.context_memory, node, 0xff, &param);
AudioEffect_update_local_params(node, 1, &param, 4);
                                                    修改参数索引
                                                    调整实际参数长度
```

6. 注意事项和常见问题

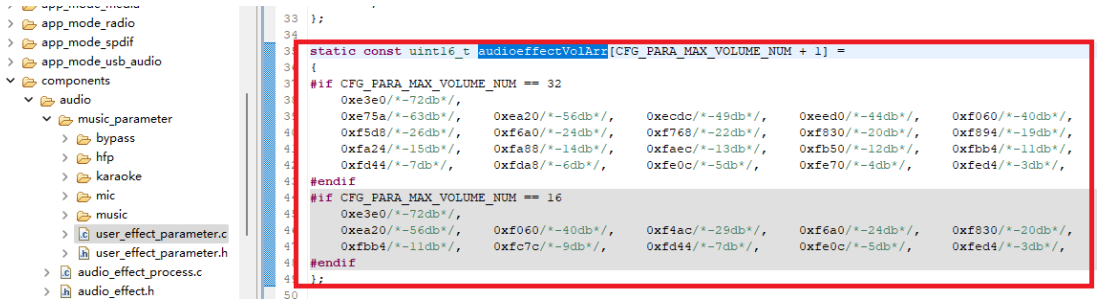
6.1. 音量控制

1. 音量控制依赖于音效框图中的 gain control 音效；



2. 原则上必须保证所有场景下用于音量控制的 gain control 处于默认开启的状态；
3. 修改框图或者新加框图，需在代码中如下位置更新音量控制 gain 地址，否则会导致音量控制不生效甚至死机；

4. 音量曲线定制：目前默认的音量调节 step 可选 16 或者 32，如需定制可修改如下地方

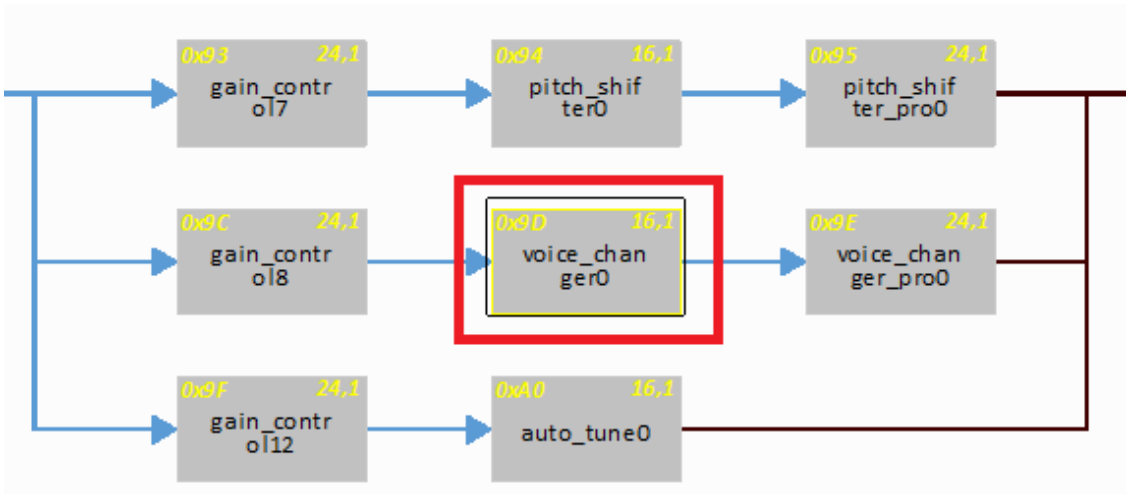


6.2. frame size 和 sample rate 的修改

在修改系统 sample rate 和 frame size 时，需要修改 user_effect_flow_xxx.c 中 user_effect_list_xxx 中的对应参数，sample rate 还需要修改 app_config.h 中的宏。

6.3. 帧长切换

通常情况下，帧长的大小由宏音效框图决定。在 Karaoke 模式下，系统帧长还会受 voice_changer 音效开关的影响。在使用调音工具在线调音时，手动打开 voice_changer，系统的帧长会自动切换至 512，再次关闭 voice_changer，系统帧长会切换回框图默认定义大小。



6.4. 调音工具与 USB debug 工具的冲突

在线调音时请关闭该宏 **CFG_FUNC_USBDEBUG_EN**，否则会导致调音异常。

6.5. roboeffect 的内存管理

Roboeffect 的内存申请主要由音效 + 控制逻辑 + 输入输出 buffer 组成。音效和控制逻辑都与实际的音效 flow 设计相关，输入输出 buffer 则与我们定义的位宽以及声道数强相关。

为了内存的合理使用以及避免不必要的浪费，建议在音效定制时同步注意以下几点：

1. 未使用到的音效在 roboeffect_config.h 中关闭对应宏；
2. 在 app_config.h 中通过宏关闭 mic 等输入输出功能时，同步删掉音效框图中的 Source & Sink

6.6. 音效库版本

上位机导出的音效参数文件 user_effect_param_xxx.c 中，有包含当前音效参数所匹配的音效库版本信息，该版本需与 SDK 中的音效库版本相匹配，否则会导致引擎库初始化失败。

```
/*****************************************************
 * @file    user_effect_param_HunXiang.c
 * @brief   auto generated
 * @author  ACPWorkbench: 3.7.0
 * @version V1.1.0
 * @Created 2023-12-01T17:30:58
 * @Graphics Name Karaoke
 * @copy; Shanghai Mountain View Silicon Technology Co.,Ltd. All rights reserved.
 *****************************************************/

#include "stdio.h"
#include "type.h"

const unsigned char user_effect_parameters_Karaoke_HunXiang[] = {
0xb8, 0x04, /*total data length*/

0x02, 0x22, 0x01, /*Effect Version*/
```

6.7. AUDIOCORE_SOURCE_SINK_ERROR

SDK 在发布时默认会包含所有的 source 和 sink 逻辑，对于框图中未使用到的 source 和 sink，在 effect_node.c 中需要将其赋为 **AUDIOCORE_SOURCE_SINK_ERROR**，让 SDK 能够正确的去处理。

```
//ROBOEFFECT effect SOURCE映射
.audioeffect_source =
{
    .mic_source = MUSIC_SOURCE_MIC_SOURCE,
    .app_source = MUSIC_SOURCE_APP_SOURCE,
    .remind_source = MUSIC_SOURCE_REMIND_SOURCE,
    .rec_source = MUSIC_SOURCE_REC_SOURCE,
    .usb_source = AUDIOCORE_SOURCE_SINK_ERROR,
    .i2s_mix_source = AUDIOCORE_SOURCE_SINK_ERROR,
    .linein_mix_source = AUDIOCORE_SOURCE_SINK_ERROR,
},

//ROBOEFFECT effect SINK映射
.audioeffect_sink =
{
    .dac0_sink = MUSIC_SINK_DAC0_SINK,
    .app_sink = MUSIC_SINK_APP_SINK,
    .stereo_sink = MUSIC_SINK_STEREO_SINK,
    .rec_sink = MUSIC_SINK_REC_SINK,
    .i2s_mix_sink = AUDIOCORE_SOURCE_SINK_ERROR,
    .spdif_sink = MUSIC_SINK_SPDIF_SINK,
},
```