V3 架构应用指导说明

V1.1

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

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 文件说明

Roboeffect 引擎核心代码文件：

```
./middleware/roboeffect
+--- inc
|   +--- roboeffect_api.h        #roboeffect api 接口声明，以及若干结构
|   +--- roboeffect_config.h     #音效宏开关和音效接口声明
+--- libRoboeffect.a
+--- src
|   +--- roboeffect_api.c        #包含音效属性的 template 表，音效 UI 定义
                    （for Acpworkbench），以及若干用户层面的 callback 函数实现
|   +--- user_defined_effect_api.c     #自定义音效 api 实现
|   +--- user_defined_effect_api.h     #自定义音效 api 声明
```

1.2 roboeffect 工作流程

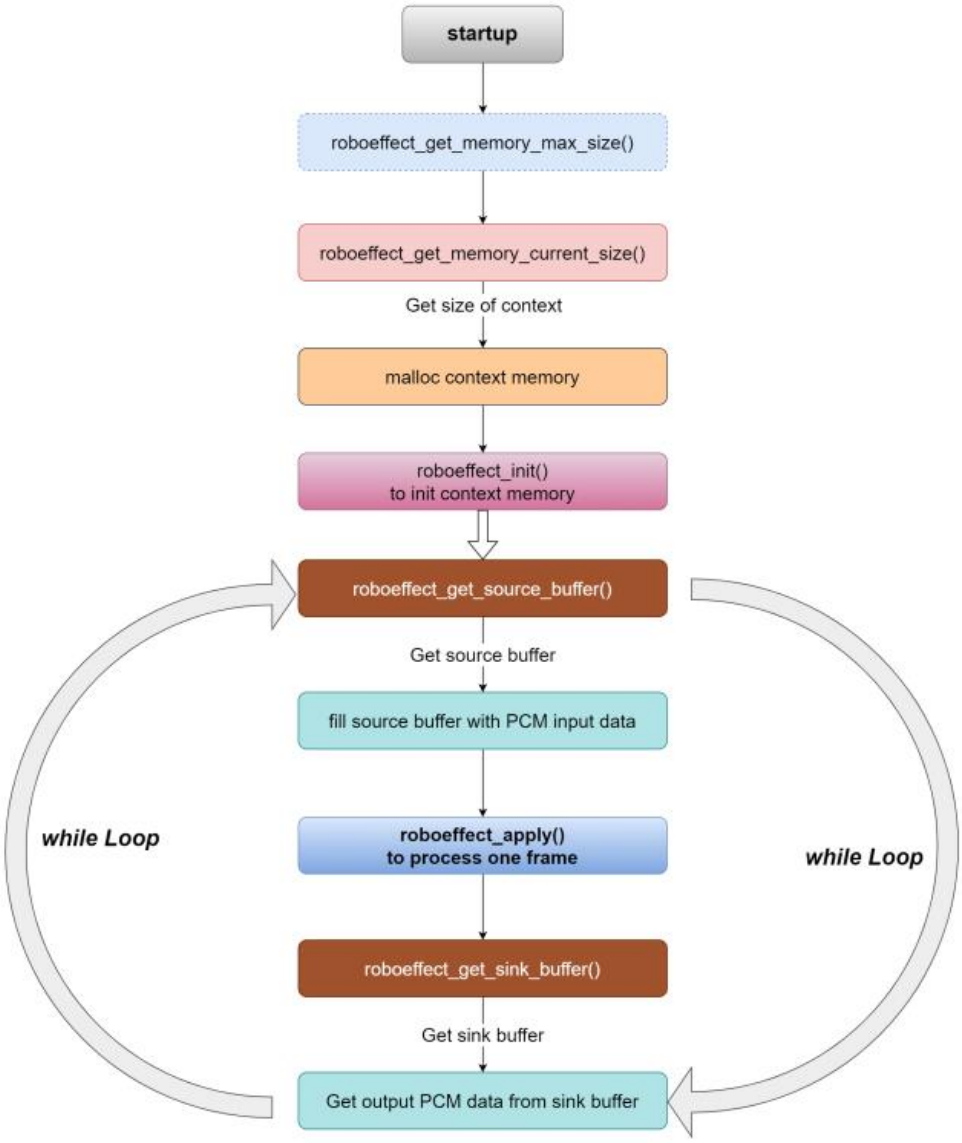

图 1-1 Roboeffect 工作流程

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

1.3 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

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 。

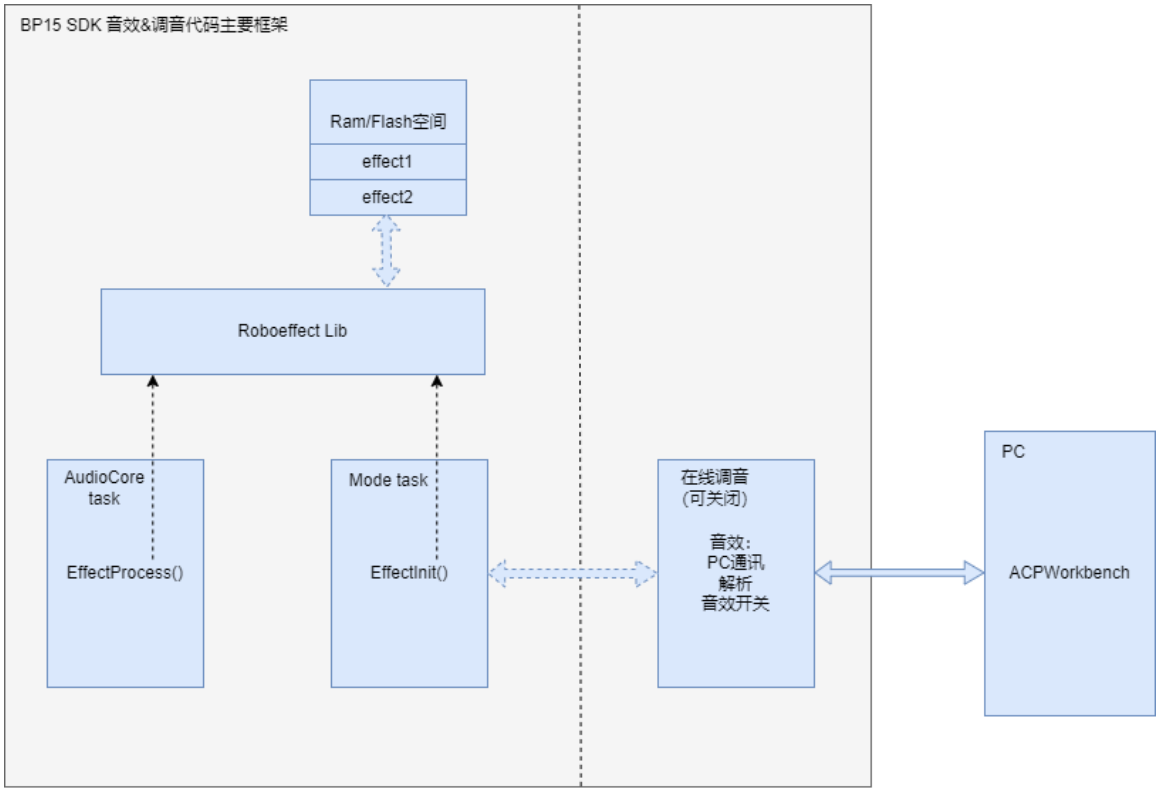

图 3-1 SDK 音效架构

SDK 以 AudioCore 为音频流处理核心，以 Roboeffect 为音效处理核心，实现灵活多变的音频音效处理。音效和调音的软件代码层次清晰，高内聚低耦合；将用户十分关注，需要经常修改的部分独立出来，方便客户进行二次开发。

3.1 Roboeffect 音效文件

SDK 的一个音效框图在调音工具的展示如下：

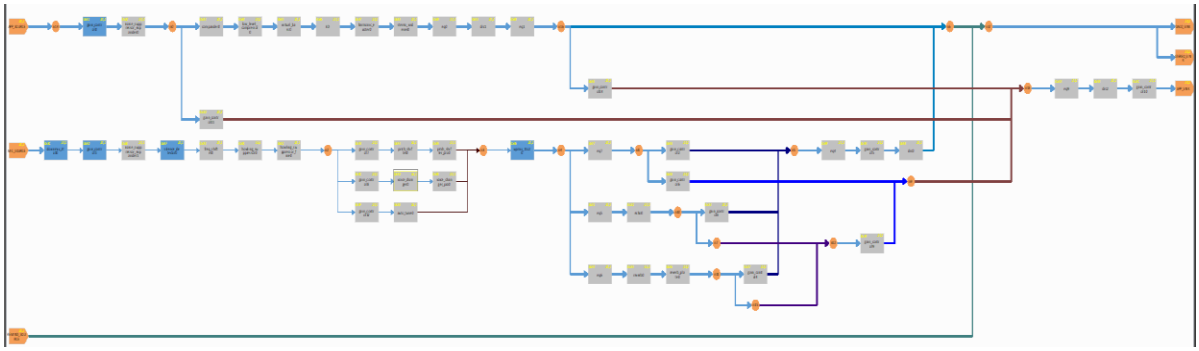

图 3-2 音效框图展示

SDK 的音效 flow 由音效框图决定，根据该图会生成如下 C 和 H 头文件：

```
./app_src/components/audio/music_parameter
+--- xxx
|    +--- user_effect_flow_xxx.h   #若干结构定义声明
|    +--- user_effect_flow_xxx.c   #音效框图描述以及若干结构定义
|    +--- user_effect_param_xxx.c  #音效参数和硬件配置参数
|    +--- effect_node.c            #自定义音效框图描述文件，适配 SDK 快速调用
```

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	在线调音功能宏
CFG_FUNC_EFFECT_BYPASS_EN	Bypass 音效，用于音频指标测试

SDK 中的音效和调音相关文件如下表。

音效文件	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
user_effect_parameter.c/user_effect_parameter.h	SDK 自定义若干调用 roboeffect 库功能的函数
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

3.1.1 音效 flow 文件

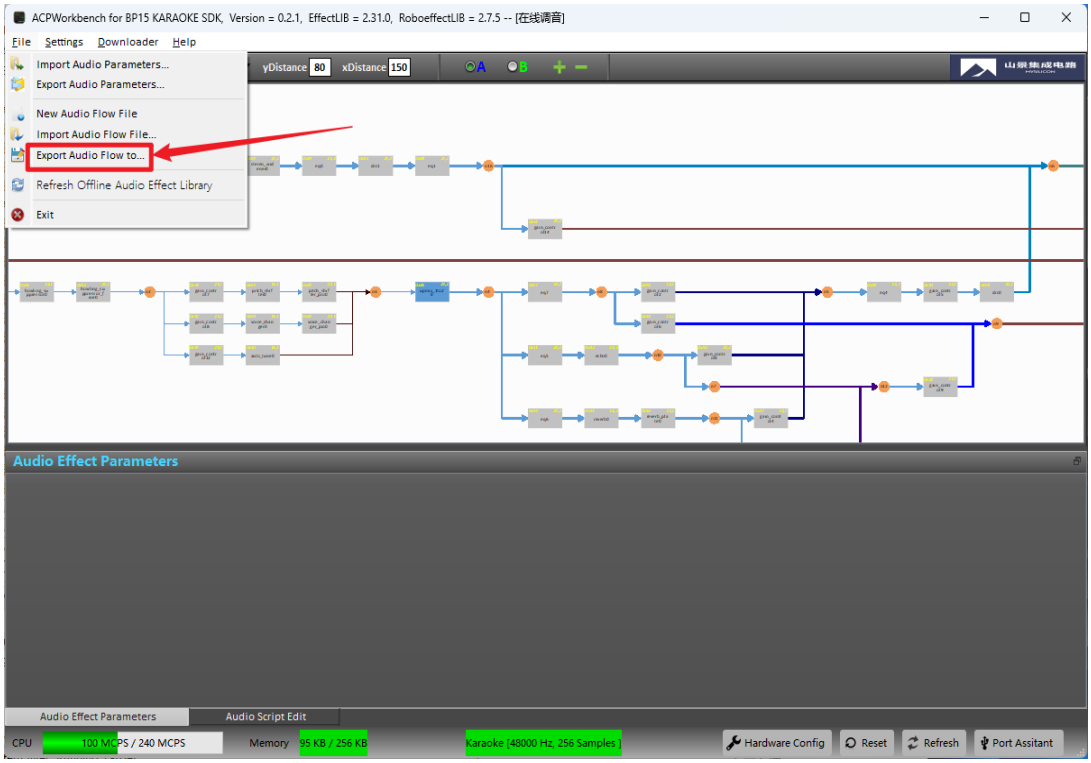

图 3-3 音效 flow 文件导出

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。

user_effect_flow_music.h

```
//输入输出定义
typedef enum _MUSIC_roboeffect_io_enum
{
    xxxxxx,
    xxxxxx,
    ...
} MUSIC_roboeffect_io_enum;
//框图使用的音效列表
typedef enum _MUSIC_roboeffect_effect_list_enum{
    xxxxxx,
    xxxxxx,
    ...
} MUSIC_roboeffect_effect_list_enum;
```

user_effect_flow_music.c

```
//音效框图加密描述
const unsigned char user_effects_script_music[] = {
    xxxx......
}
//音效细节描述
static const roboeffect_exec_effect_info user_effects_music[] = {
    {XXX   , XXX   , XXX   , XXX},  //mic_eq0
    ...
};
roboeffect_effect_list_info user_effect_list_music = {
    MUSIC_COUNT_ADDR - 0x81, //count
    48000, //sample rate
    256, //framse size
    user_effects_music,
    NULL,
};
//Source 细节描述
static const roboeffect_io_unit source_unit_music[] = {
    {XXX          ,  X , XXX         , XXX}, //{source, mem, bit_width, ch}
    ...
```

```
};
//Sink 细节描述
static const roboeffect_io_unit sink_unit_music[] = {
    {XXX          ,   X, XXX          , XXX}, //{sink, mem, bit_width, ch}
    ...
};
//音效 path 描述
static const roboeffect_step effect_flow_music[] = {
    {  X,     X,      X,      X,      X},
    ...
};
```

音效 flow 结构的命名由固定前缀+调音工具页面的流程图名组成。通常情况下，除采样率和帧长外上述信息应全部由调音工具导出，采样率和帧长可根据使用场景进行手动调整。

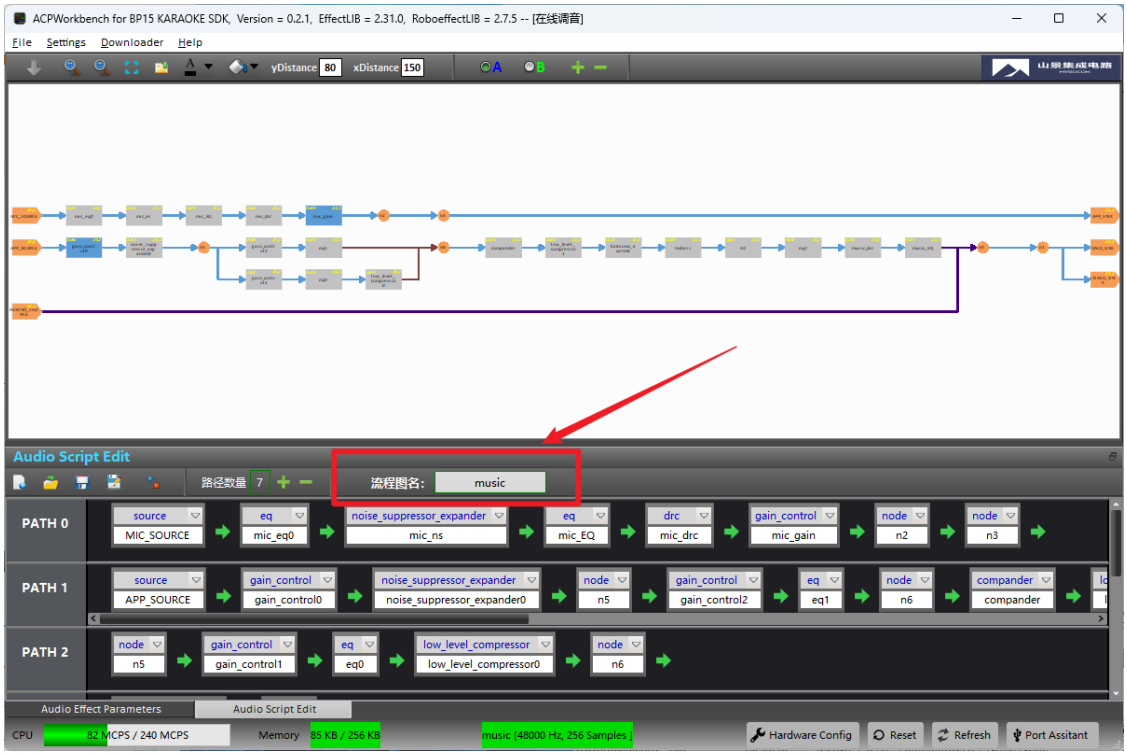

图 3-4 流程图名

3.1.2 音效参数文件

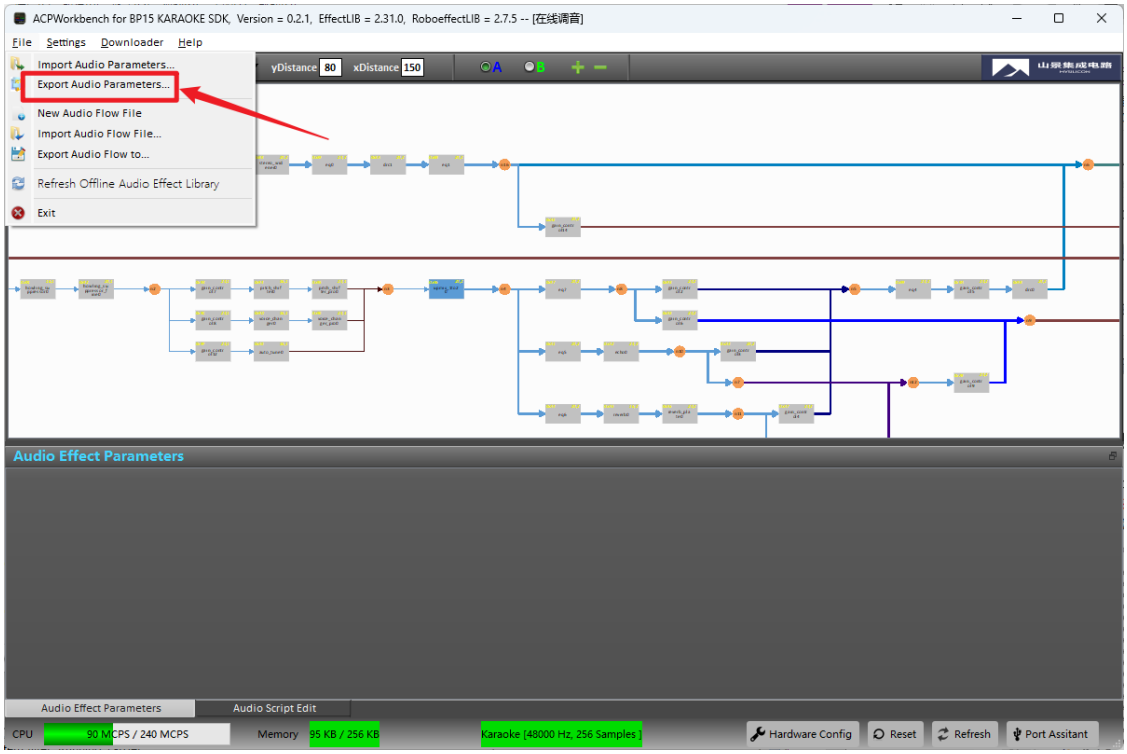

图 3-5 音效参数文件导出

音效参数文件（user_effect_param_xxx.c）由调音工具导出，主要包含调音完成的音效参数信息。一个音效框图可以有多组不同的音效参数。

```
//音效参数
const unsigned char user_effect_parameters_music_default[] = {
0x61, 0x03, /*total data length*/
0x02, 0x1f, 0x00, /*Effect Version*/
...
};
//硬件配置参数
const unsigned char user_module_parameters_music_default[] = {
...
};
```

所有的音效参数都由 addr + length + enable + params 的形式排列，硬件配置参数的具体信息请参考《固件与用户应用程序通信协议》。

音效参数结构的命名由固定前缀+音效 flow 名+音效名（导出时填写的文件名称）组成。

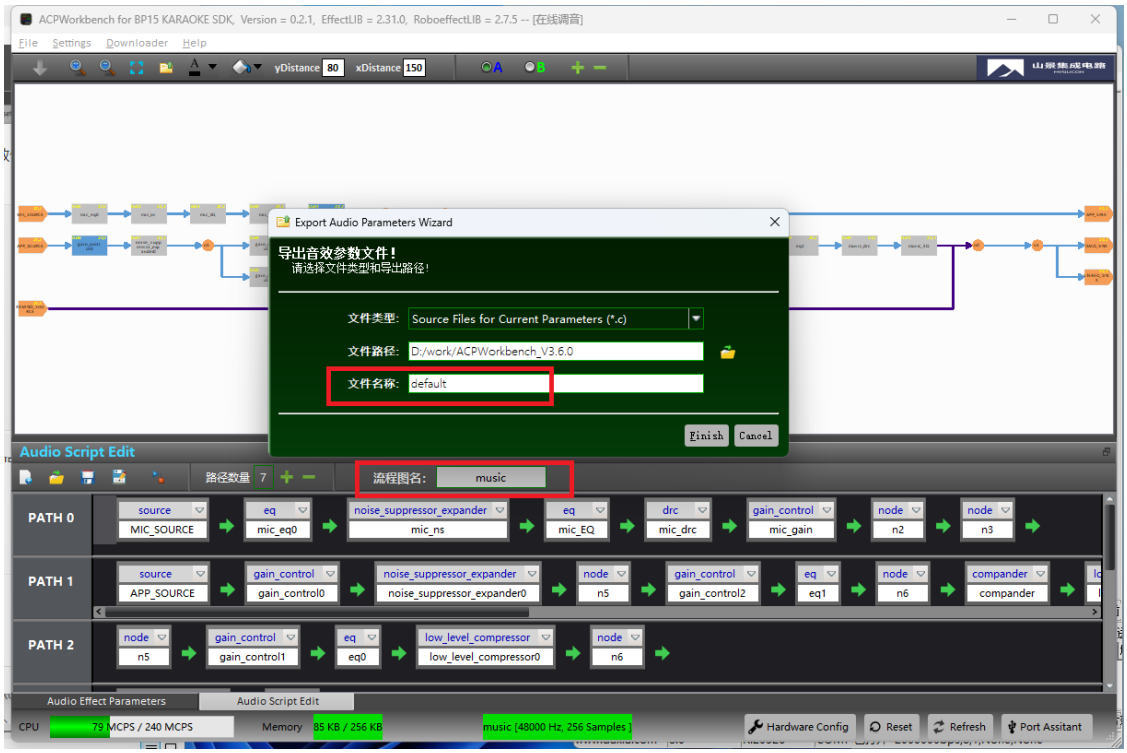

图 3-6 音效参数结构命名

3.1.3 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。

3.2 Roboeffect Init

3.2.1 选择正确的框图和音效参数

为了便于使用，roboeffect 相关的一些结构体放在 AudioCoreContext 中。

```
typedef struct _RoboeffectContext
{
    uint8_t *context_memory;
    roboeffect_effect_list_info *user_effect_list;
    roboeffect_effect_steps_table *user_effect_steps;
    uint8_t *user_effect_parameters;
    uint8_t *user_effects_script;
    uint16_t user_effects_script_len;
    uint8_t *user_module_parameters;
    int32_t roboeffect_size;
    int32_t roboeffect_size_max;
    uint8_t effect_count;
    uint8_t effect_addr;
    uint8_t effect_enable;
    //ROBOEFFECT_ERROR_CODE roboeffect_ret;
}RoboeffectContext;

typedef struct _AudioCoreContext
{
    uint32_t        AdaptIn[(MAX_FRAME_SAMPLES * sizeof(PCM_DATA_TYPE)) / 2];//转采样和软件微调输入buf 4字节对齐便于dmafifo衔接
    uint32_t        AdaptOut[(MAX_FRAME_SAMPLES * SRC_SCALE_MAX * sizeof(PCM_DATA_TYPE))/ 2];//转采样和软件微调输出buf
    MIX_NET         CurrentMix; //当前混音组合，旨在多通路异步处理和收发。
    uint16_t        FrameReady; //使用位段登记数据/空间帧可用
    uint32_t        SampleRate[MaxNet];//[DefaultNet]/[0]:主通路中心采样率。
    uint16_t        FrameSize[MaxNet];//[DefaultNet]/[0]:主通路采样帧，支持独立通路组合SeparateNet及独立采样帧。
    AudioCoreSource AudioSource[AUDIO_CORE_SOURCE_MAX_NUM];
    AudioCoreProcessFunc AudioEffectProcess;            //****流处理入口
    AudioCoreSink   AudioSink[AUDIO_CORE_SINK_MAX_NUM];
    RoboeffectContext   Roboeffect;
}AudioCoreContext;
```

图 3-7 roboeffect 结构体

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 RoboeffectInit()来完成 roboeffect 的初始化。

根据当前模式选择的音效，我们需要判断并找到正确的音效 flow 和与之匹配的音效参数。目前该部分也不需要手动修改，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 RoboeffectInit()
{
    if(AudioCore.Roboeffect.effect_addr)
    {
        uint8_t *params = AudioCore.Roboeffect.user_effect_parameters + 5;
        uint16_t data_len = *(uint16_t *)AudioCore.Roboeffect.user_effect_parameters - 5;
        uint8_t len = 0;
        while(data_len)
        {
            if(*params == AudioCore.Roboeffect.effect_addr)
            {
                params += 2;
                *params = AudioCore.Roboeffect.effect_enable;
                break;
            }
            else
            {
                params++;
                len = *params;
                params += (len + 1);
                data_len -= (len + 1);
            }
        };

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

        ROBOEFFECT_EFFECT_PARA *para = get_user_effect_parameters(mainAppCt.EffectMode);

        AudioCore.Roboeffect.user_effect_list = get_local_effect_list_buf();
        memcpy(AudioCore.Roboeffect.user_effect_list,para->user_effect_list,sizeof(roboeffect_effect_list_info));

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

图 3-8 音效模式选择及参数加载

3.2.2 计算需要的内存大小并尝试申请

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.Roboeffect.roboeffect_size < xPortGetFreeHeapSize())
{
    AudioCore.Roboeffect.context_memory = roboeffect_malloc(AudioCore.Roboeffect.roboeffect_size);
    if(AudioCore.Roboeffect.context_memory == NULL)
    {
        return FALSE;
    }
    /**
     * initial roboeffect context memory
    */
    if(ROBOEFFECT_ERROR_OK != roboeffect_init(AudioCore.Roboeffect.context_memory,
                                              AudioCore.Roboeffect.roboeffect_size,
                                              AudioCore.Roboeffect.user_effect_steps,
                                              AudioCore.Roboeffect.user_effect_list,
                                              AudioCore.Roboeffect.user_effect_parameters) )
    {
        DBG("roboeffect_init failed.\n");
        return FALSE;
    }
    else
    {
        DBG("roboeffect_init ok.\n");
        AudioCore.Roboeffect.effect_addr = 0;
        Roboeffect_GetAudioEffectMaxValue();
    }
}
else
{
    DBG("************************************\n");
    DBG("Error:memory is not enough!!!\n");
    DBG("malloc:%ld, leave:%ld\n", AudioCore.Roboeffect.roboeffect_size_max, xPortGetFreeHeapSize())
    DBG("************************************\n");
    return FALSE;
}
```

图 3-9 内存申请

3.2.3 roboeffect_init()初始化 roboeffect 引擎

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

3.2.4 初始化上位机交互模块

roboeffect_prot_init()

3.3 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));
```

3.4 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();
```

3.5 在线调音

在线调音的逻辑实现基本都在 communication.c 中，以如下函数为核心展开。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。

```
void Communication_Effect_Config(uint8_t Control, uint8_t *buf, uint32_t len)
{
	switch(Control)
	{
		case 0x00:
			Communication_Effect_0x00();
			break;
		case 0x01:
			Communication_Effect_0x01(buf, len);
			break;
		case 0x02:
			Communication_Effect_0x02();
			break;
		case 0x03:
```

```
                Communication_Effect_0x03(buf, len);
                break;
            case 0x04:
                Communication_Effect_0x04(buf, len);
                break;
            case 0x06:
                Communication_Effect_0x06(buf, len);
                break;
            case 0x07:
                Communication_Effect_0x07(buf, len);
                break;
            case 0x08:
                Communication_Effect_0x08(buf, len);
                break;
            case 0x09:
                Communication_Effect_0x09(buf, len);
                break;
            case 0x0A:
//                Communication_Effect_0x0A(buf, len);
                break;
            case 0x0B:
                Communication_Effect_0x0B(buf, len);
                break;
            case 0x0C:
                Communication_Effect_0x0C(buf, len);
                break;
            case 0x0D:
                Communication_Effect_0x0D(buf, len);
                break;
            case 0x80:
                Communication_Effect_0x80(buf, len);
                break;
            case 0xfc://user define tag
                 Communication_Effect_0xfc(buf, len);
                break;
            case 0xfd://user define tag
                 Communication_Effect_0xfd(buf, len);
                break;
            case 0xff:
                Communication_Effect_0xff(buf, len);
```

```
                    break;
              default:
                    if((Control >= 0x81) && (Control < 0xfb))
                    {
                          roboeffect_effect_update_params_entrance(Control, bu
f, len);
                    }
                    else
                    {
                    }
                    break;
        }
        //-----Send ACK ----------------------//
        if(Control > 0xf0)
        {
              return;
        }
        if((Control > 2)&&(Control != 0x80))
        {
              if(len > 0)// if(len = 0) {polling all parameter}
              {
                    memset(tx_buf, 0, sizeof(tx_buf));
                    tx_buf[0] = Control;
                    Communication_Effect_Send(tx_buf, 1);
              }
        }
}
```

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 音效宏的选择

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

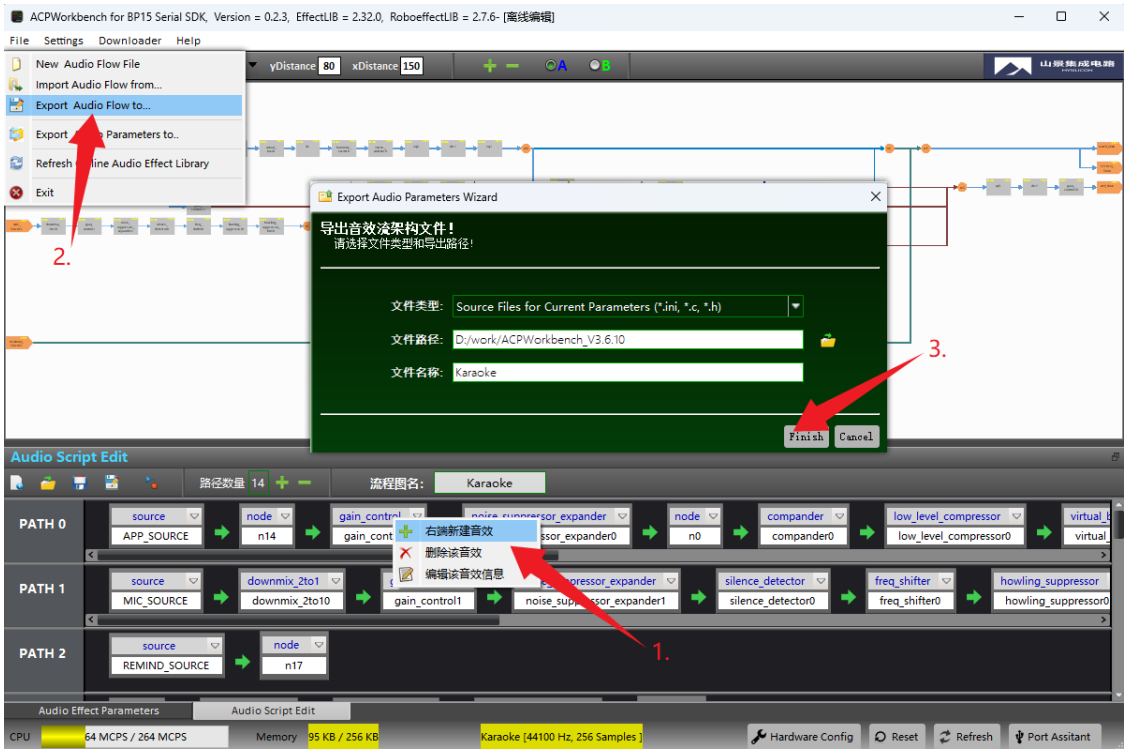

图 4-1 音效修改

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。

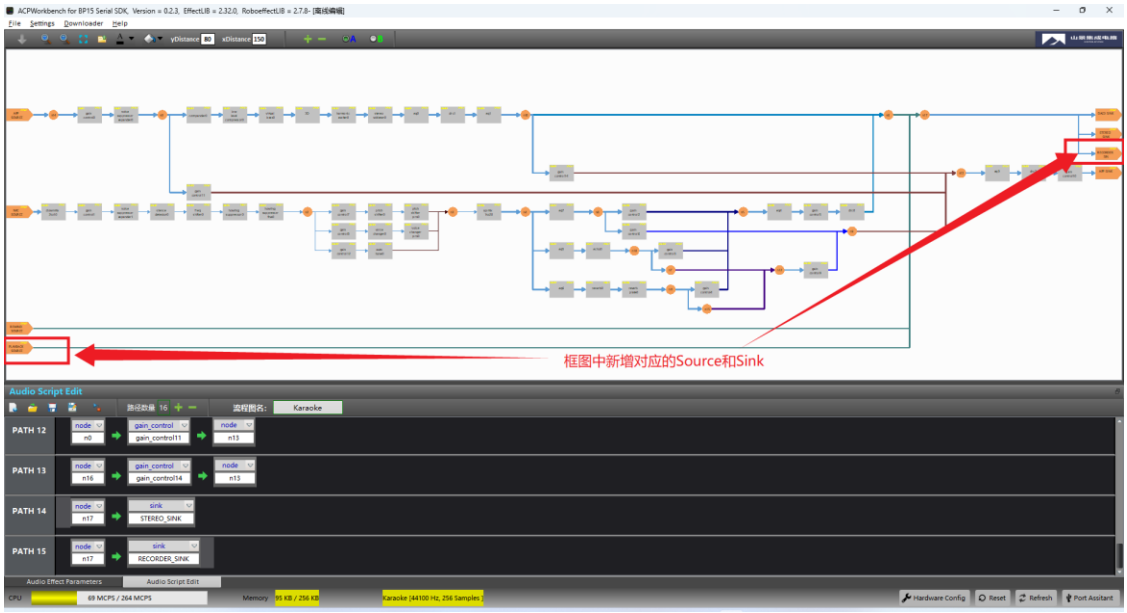

图 4-2 输入输出修改

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 _ROBOEFFECT_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;  //REMIND_SOURCE_NUM //提示音使用固定混音通道
    uint8_t playback_source;//PLAYBACK_SOURCE_NUM, //flashfs 录音回放通道      无音效
}ROBOEFFECT_SOURCE_NUM;
extern const ROBOEFFECT_SOURCE_NUM roboeffect_source[];

typedef struct _ROBOEFFECT_SINK_NUM
{
    uint8_t dac0_sink;      //AUDIO_DAC0_SINK_NUM      //主音频输出在audiocore Sink中的通道，必须配置，audiocore借用此通道buf处理数据
    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 recorder_sink;  //AUDIO_RECORDER_SINK_NUM   //录音专用通道      不叠加提示音音源。
}ROBOEFFECT_SINK_NUM;
extern const ROBOEFFECT_SINK_NUM roboeffect_sink[];
```

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

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

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

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

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

以及 effect_node.c 中如下映射。

```
    //ROBOEFFECT effect SOURCE映射
    .roboeffect_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
    .roboeffect_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,
    },
```

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。

以混响为例：

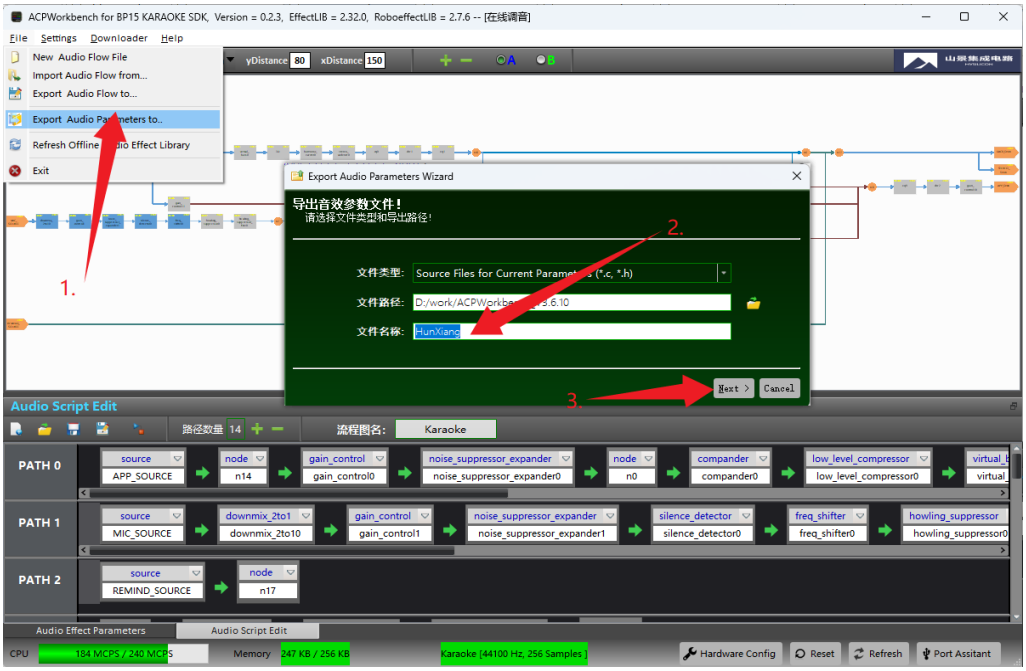

图 4-3 音效参数导出流程

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。

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,
    .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,
    .DAC0_SINK_GAIN_ADDR = KARAOKE_gain_control0_ADDR,  //框
    .APP_SINK_GAIN_ADDR = KARAOKE_gain_control10_ADDR,
},

//ROBOEFFECT effect SOURCE映射
.roboeffect_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
.roboeffect_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,
},
```

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_roboeffect_addr()**、**AudioCoreSourceToRoboeffect()**、**AudioCoreSinkToRoboeffect()**三个 API 来查找。

5 注意事项和常见问题

5.1 音量控制

1. 音量控制依赖于音效框图中的 gain control 音效；

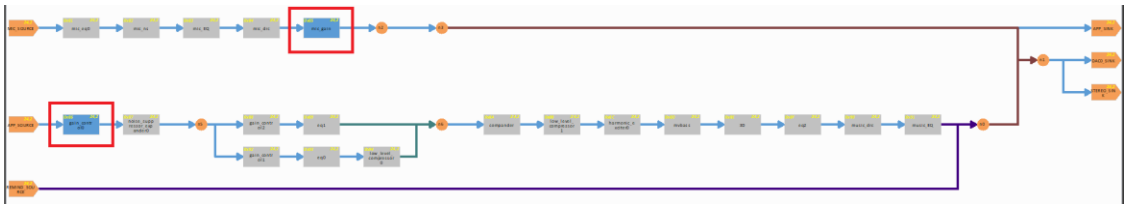

图 5-1 音量控制

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；

图 5-2 音效地址

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。

图 5-3 音量曲线

5.2 帧长的切换

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。

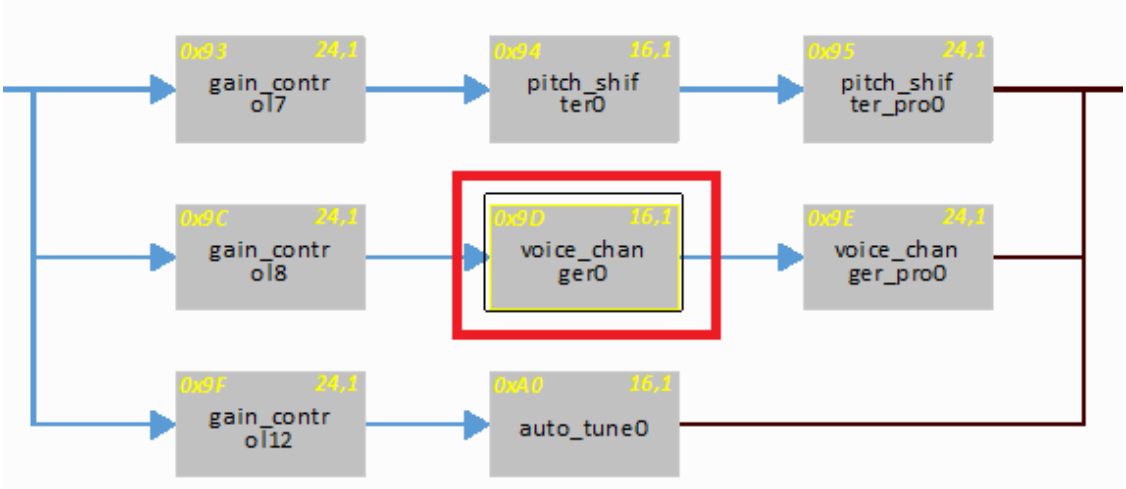

图 5-4 voice_changer 音效

5.3 调音文件的导入导出

调音文件主要分为音效 flow 和音效参数两种，需要注意的是音效参数是跟一些 flow 深度绑定的，在使用调音工具导出的时候一定要明确导出的音效参数对应的音效 flow 是哪一个。

5.3.1 音效 flow 文件

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，表示当前框图是 Karaoke。

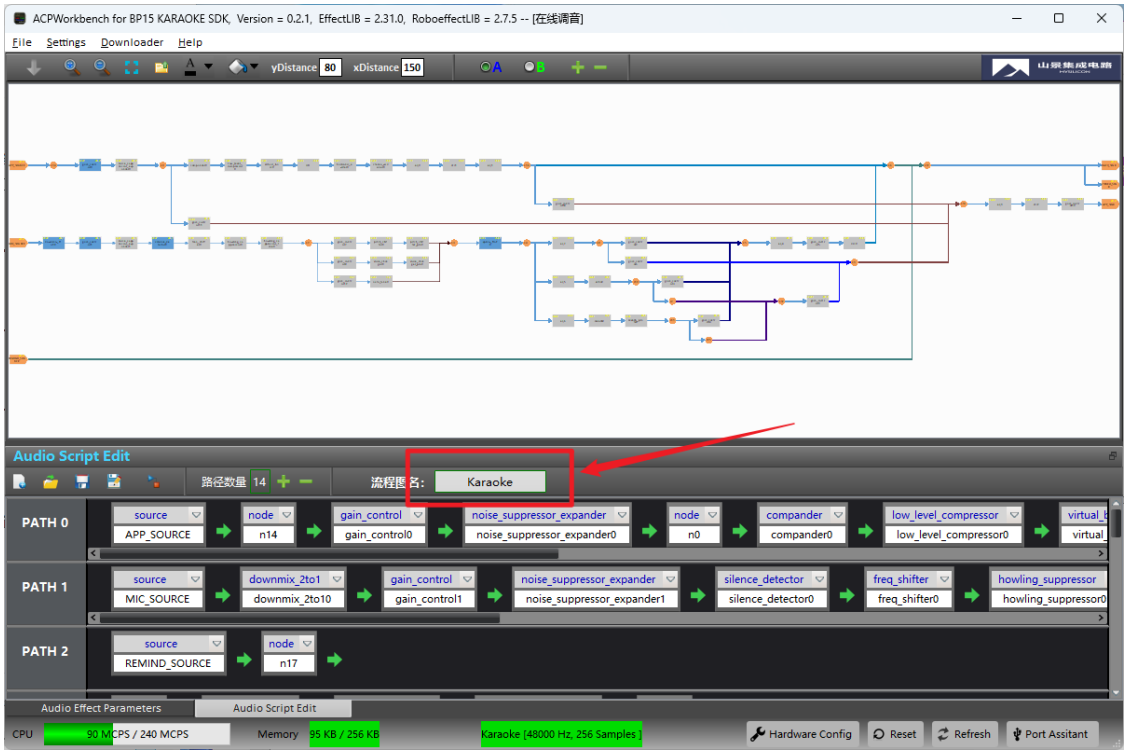

图 5-5 Karaoke flow

音效 flow 文件导出的命名为 user_effect_flow_xxx.c/h，可以看到在导出的 karaoke flow 中，所有结构的命名都是以 KARAOKE 为前缀。

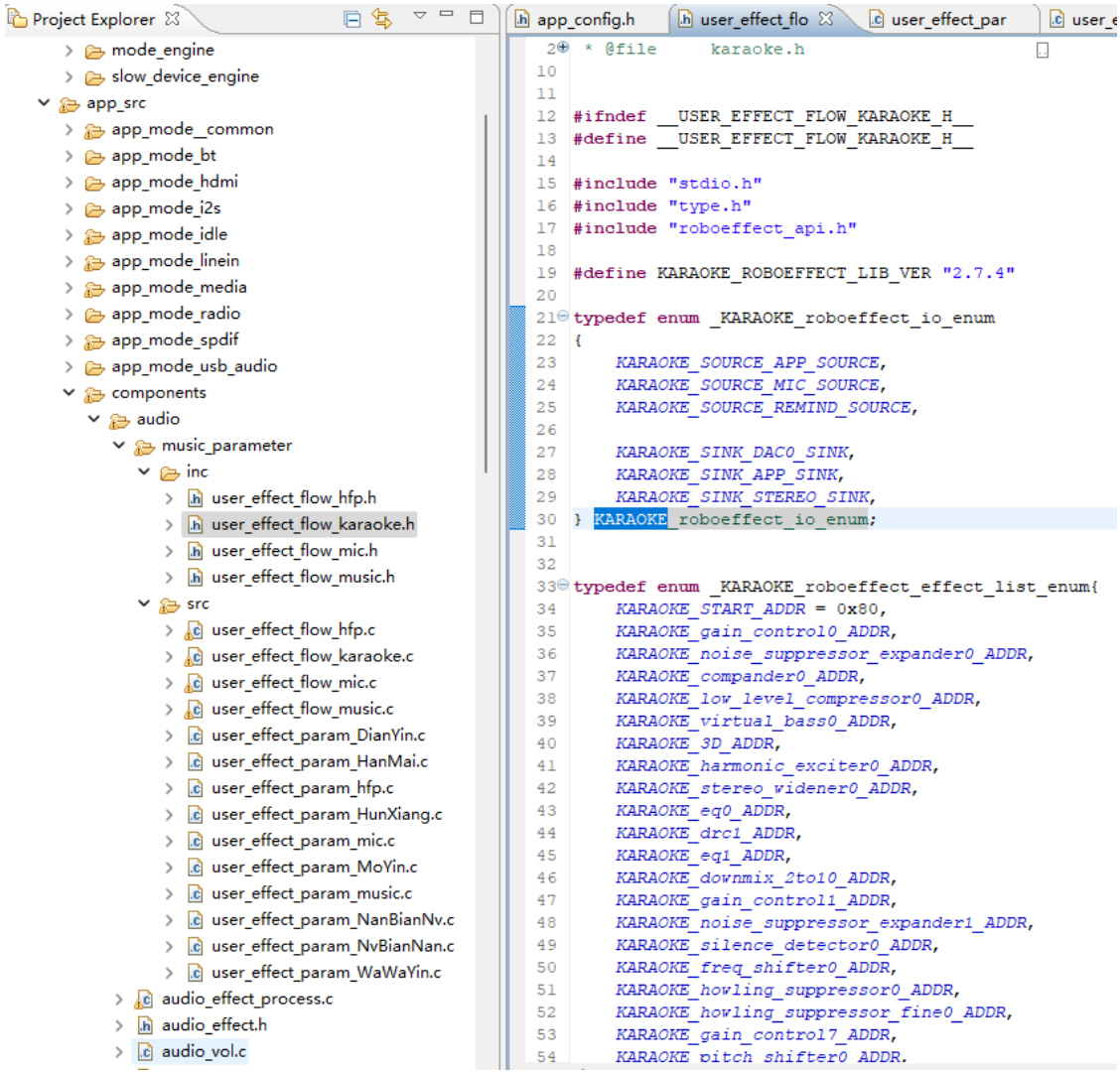

图 5-6 user_effect_flow_xxx.c

5.3.2 音效参数文件

音效参数文件的不同点在于，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。

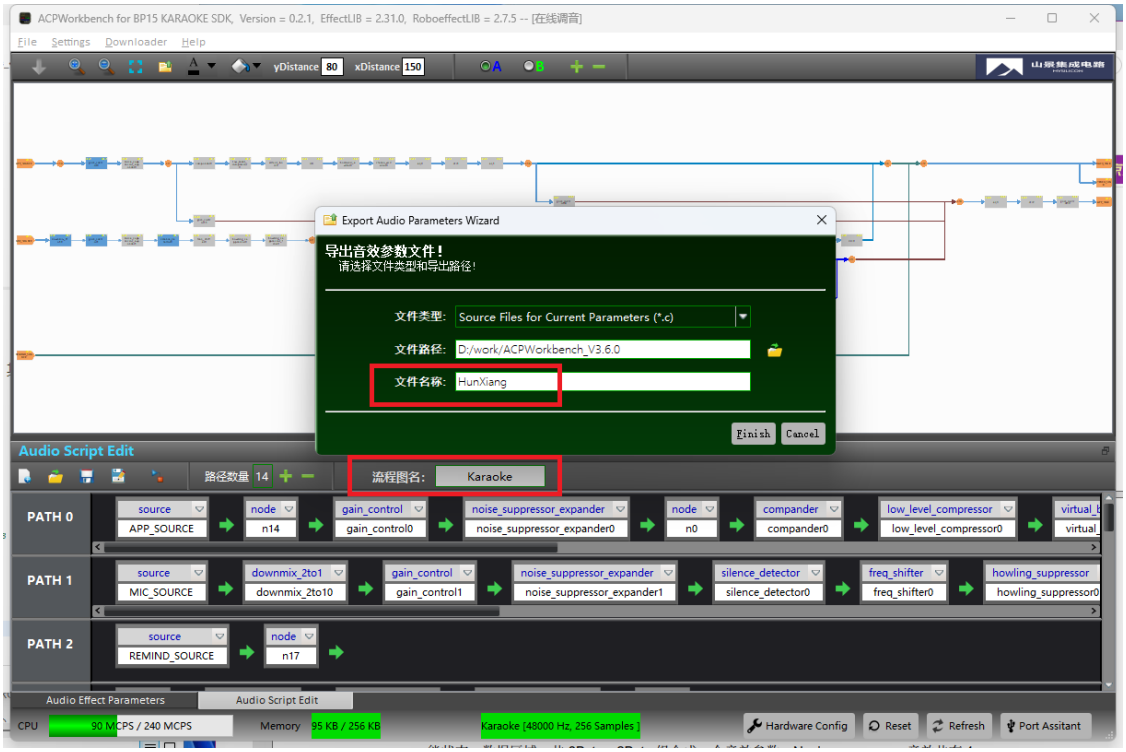

图 5-7 Karaoke 音效参数导出

```c
/*****************************************************
 * @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[] = {
0xb1, 0x04, /*total data length*/

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

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

图 5-8 user_effect_param_xxx.c

5.4 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。

5.5 frame size 和 sample rate 修改

在修改系统 sample rate 时，除了要修改 app_config.h 中的宏之外，还需要修改 user_effect_flow_xxx.c 中 user_effect_list_xxx 中的对应参数。frame size 已经完全依赖于音效框图，无需修改任何宏。

5.6 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。

5.7 Karaoke 模式

相较于普通 music 播放，Karaoke 模式下会一些比较特殊的功能，如高低音、回声、闪避等，这里重点介绍下高低音和回声功能。

5.7.1 高低音控制

高低音控制包含 mic 通路和 music 通路，通过如下消息来实现对应通路 EQ 的调节以实现想要达到的效果。

```
MSG_MIC_TREB_UP,
MSG_MIC_TREB_DW,
MSG_MIC_BASS_UP,
MSG_MIC_BASS_DW,
MSG_MUSIC_TREB_UP,
MSG_MUSIC_TREB_DW,
MSG_MUSIC_BASS_UP,
MSG_MUSIC_BASS_DW,
```

5.7.2 回声效果

回声主要针对于 mic 通路，通过下列消息可以调节 Echo 和 Reverb 音效对应的参数来实现更理想的回声效果。

```
MSG_MIC_EFFECT_UP,
MSG_MIC_EFFECT_DW,
```