V3 架构应用指导说明

V1.7

版本记录：

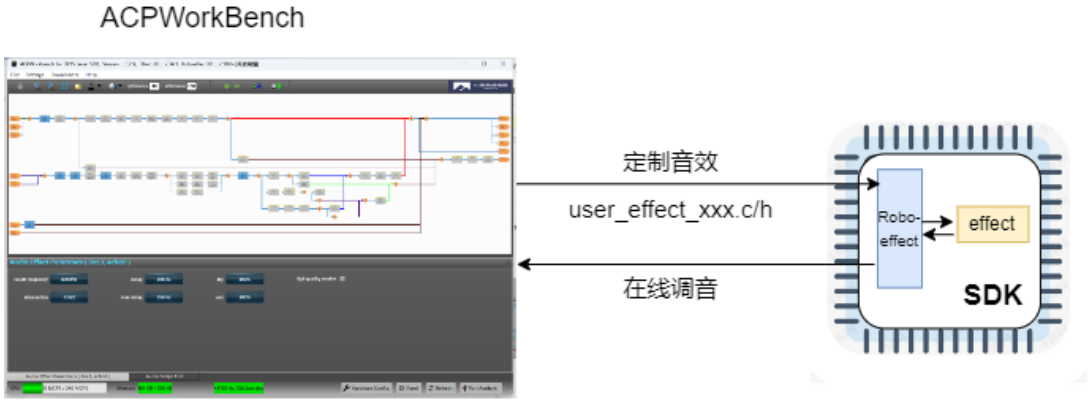
版本号	日期	作者	备注
V1.0	2023-10-18	Yangyu	初版
V1.1	2023-11-21	Yangyu	架构调整更新
V1.2	2023-12-15	Yangyu	格式及内容优化
V1.3	2024-01-29	Yangyu	内容更新
V1.4	2024-03-19	Yangyu	内容更新
V1.5	2024-04-07	Yangyu	注意事项更新
V1.6	2024-05-21	Yangyu	新增章节
V1.7	2024-07-11	Liangzx	内容更新

目录

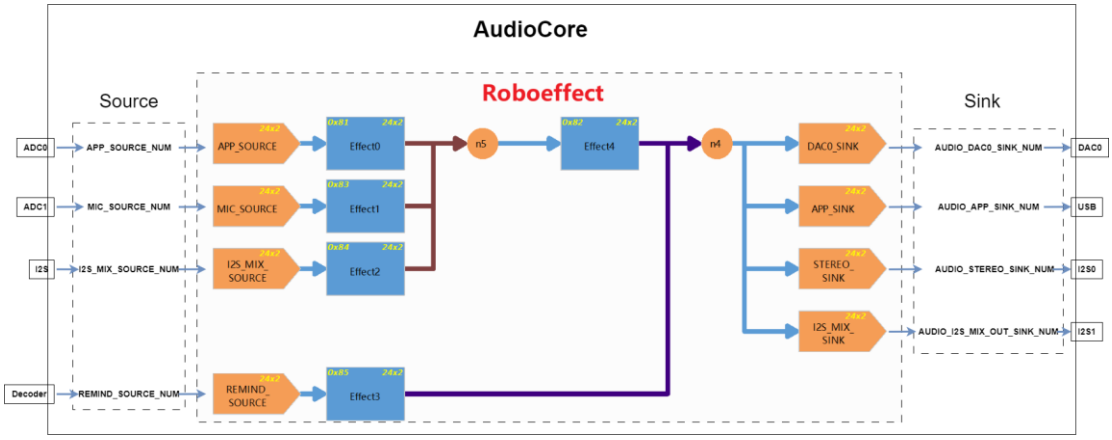
1. Roboeffect 介绍

面对日益复杂的应用场景，为了实现更好的音频效果，在原有 SDK 架构以及 V2 版本音效架构的基础上，推出新的 V3 版本音效架构，保留原有的 SDK 架构不变，使得音效处理流程更加简洁，开发过程更加方便快捷。

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。

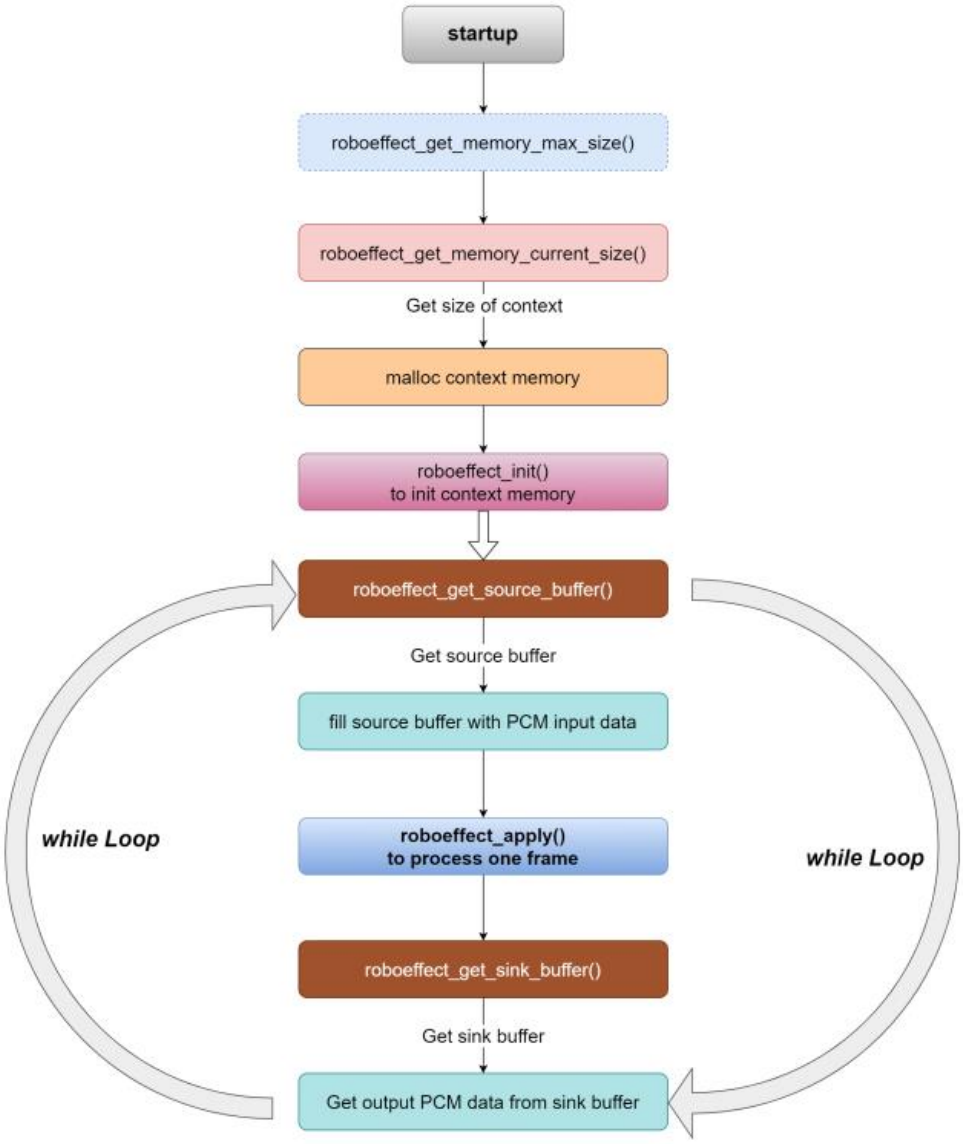


1.1. Roboeffect 与 SDK 的关系



Roboeffect 本质上还是作为一个音效运行在 AudioCore 中，Roboeffect 中的 source 和 sink 也并不是 SDK 的输入输出，需要与 AudioCore 中的 source 和 sink 互相绑定，方可正常运行。

1.2. 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 库文件说明

Roboeffect 引擎核心代码文件：

```
/middle/roboeffect
 +--- roboeffect_lib
    +--- 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
    +--- roboeffect_api.c                    包含音效属性的 template 表，音效 UI 定义等
    +--- libRoboeffect.a
    +--- roboeffect_library_info_vX.X.X.bin  离线音效库信息 for 上位机
 +--- 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
```

1.4. roboeffect API 介绍

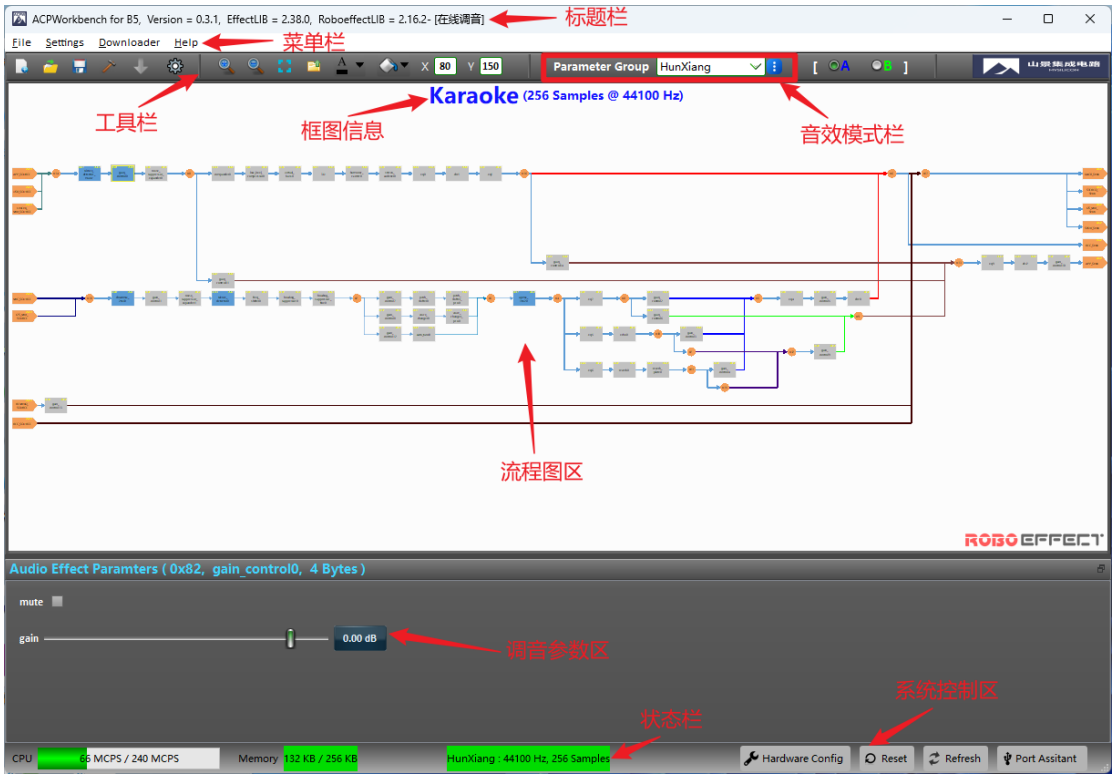
Roboeffect 提供丰富的 API，使外部 SDK 可灵活调用操作整个引擎库。

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_estimate_frame_size ()	根据当前框图中音效开启状态评估合适的帧长
roboeffect_get_frame_size_after_effect_change ()	根据将要开启/关闭音效状态获取合适的帧长

2. ACPWorkBench V3.x.x 版本介绍

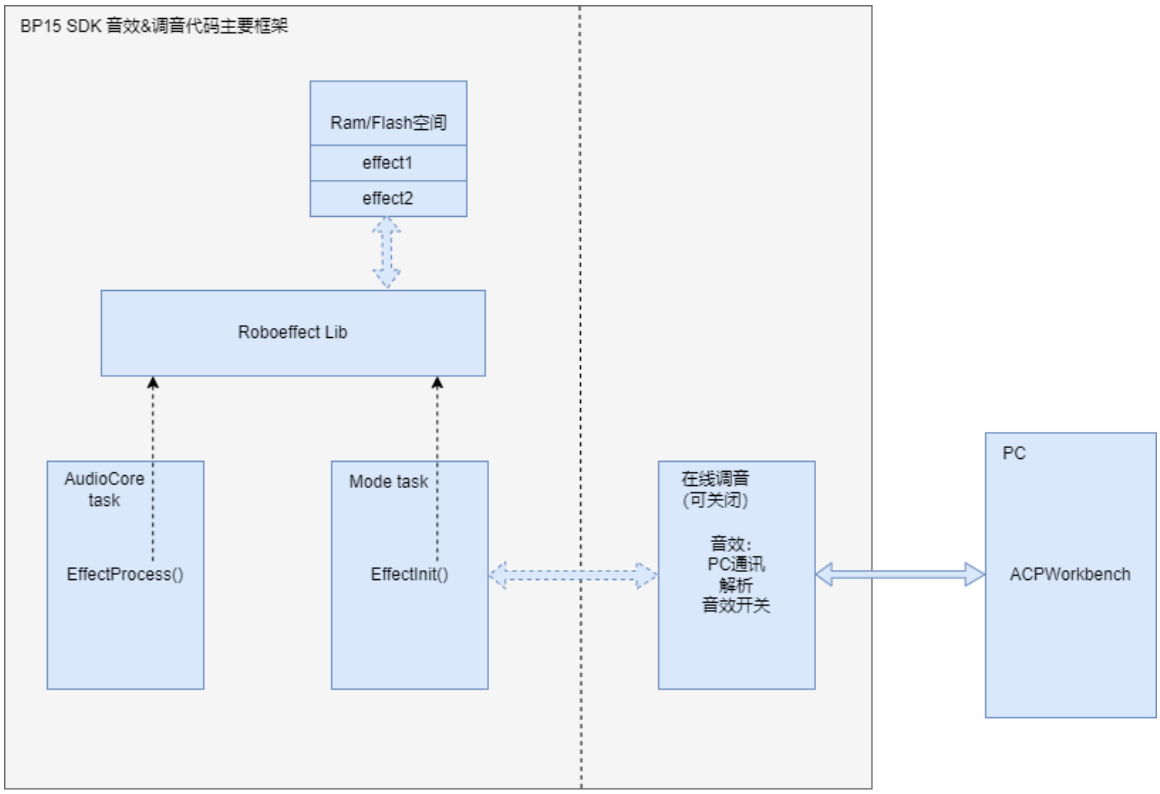
可视化调音工具 ACPWorkbench 是一款可以实时绘制音效流， 实时调音的工具， 相比 ACPWorkbench V2 版本， 该版本从视觉和功能上有了直观的改变。 无论是熟悉山景 SDK 的用户还是刚刚接触的新用户， 都能受益于其直观的操作和快速的音效流定制。需注意 ACPWorkbench V3 版本不兼容 V2 版本。



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。



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. SDK 宏

3.1.1. 音效宏

SDK 中对于各种音效用宏做了自动化管理，当 SDK 所有框图都不包含某音效时，预编译阶段会自动关闭对应音效宏，这样这部分代码以及相关的音效库函数均不会被编译，可以减少代码量。

3.1.2. 应用功能宏

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_EFFECT_PARAM_IN_FLASH_EN	音效参数在线下载宏开关
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

3.2. SDK 音效相关文件

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_mode.c	音效框图中间描述文件，专用于 SDK 做接口适配

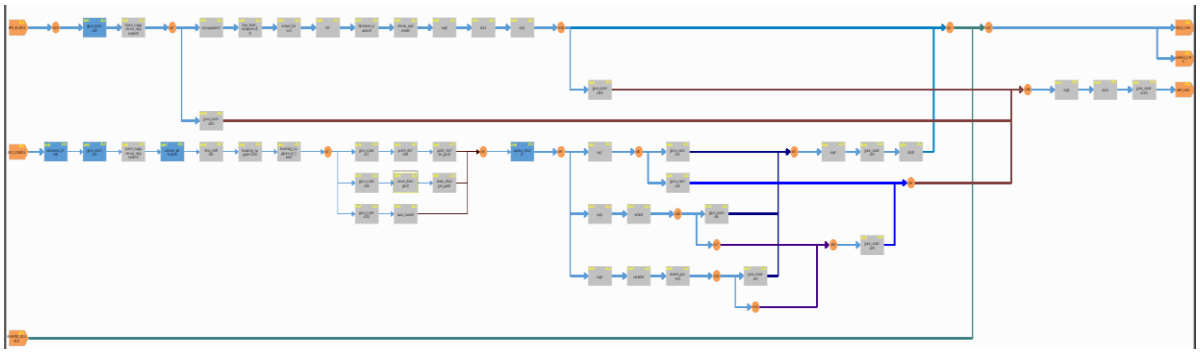
3.3. 以 effect_mode 为核心

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_mode.c** 与之深度绑定，定制音效只需简单修改 effect_mode.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 mode***************/
    EFFECT_MODE_MIC,
    /**********music mode*************/
    EFFECT_MODE_MUSIC,
    /**********bypass mode************/
    EFFECT_MODE_BYPASS,
    /**********hfp mode***************/
    EFFECT_MODE_HFP_AEC,
    /**********karaoke mode***********/
     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;
```

3.4. Roboeffect 音效文件

SDK 的一个音效框图在调音工具的展示如下：



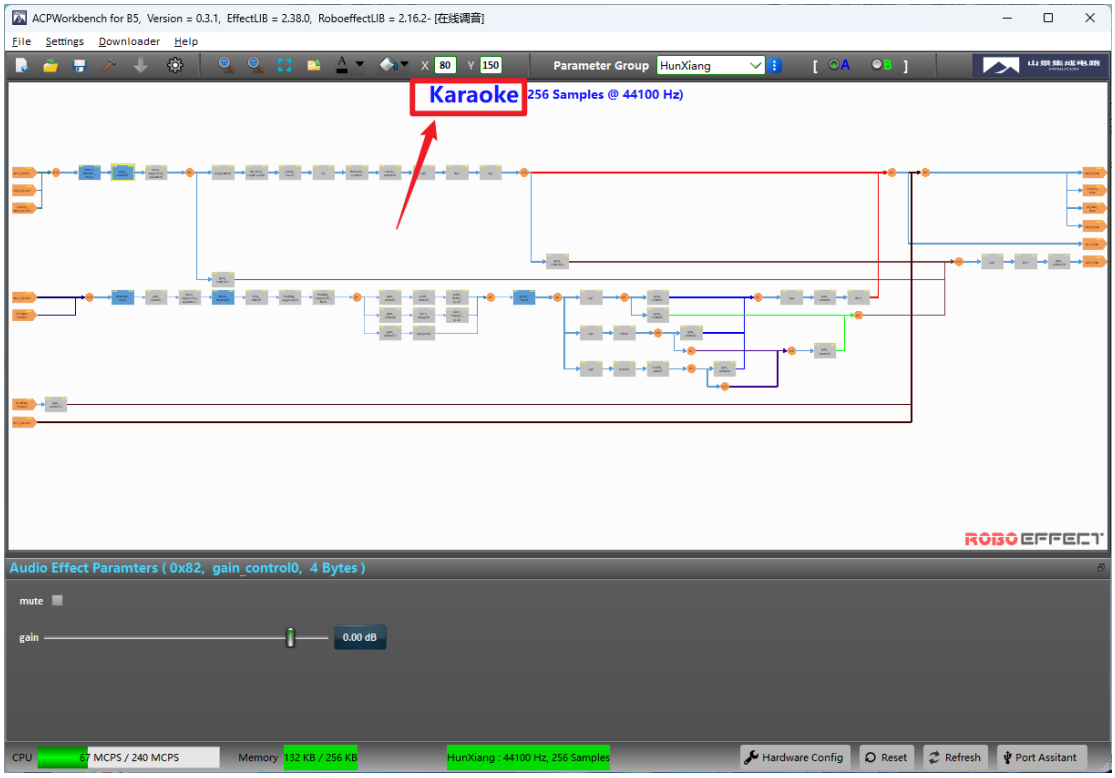
SDK 的音效处理由音效框图决定，根据该图会生成如下 C 和 H 代码文件：

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。

3.4.1. 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，即可调音工具中看到“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。

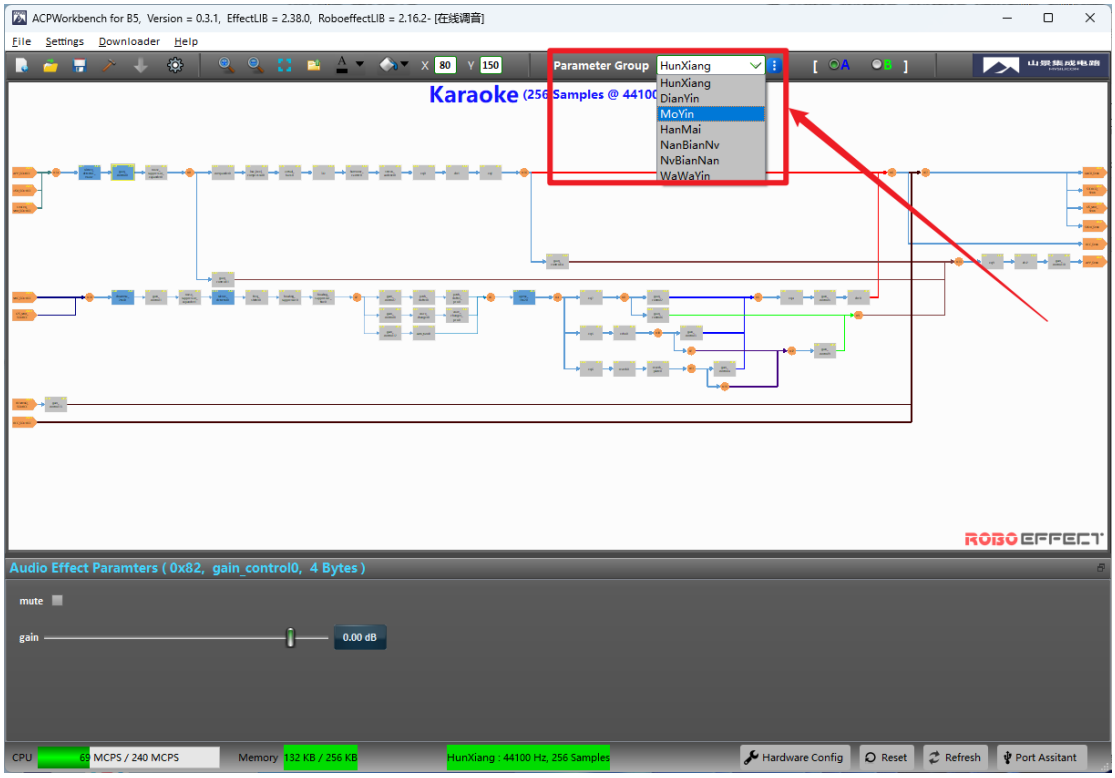



```
#include "stdio.h"
#include "type.h"
#include "roboeffect_config.h"
#include "roboeffect_api.h"
#include "user_defined_effect_api.h"
#include "user_effect_flow_karaoke.h"

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,
```

3.4.2. 音效参数

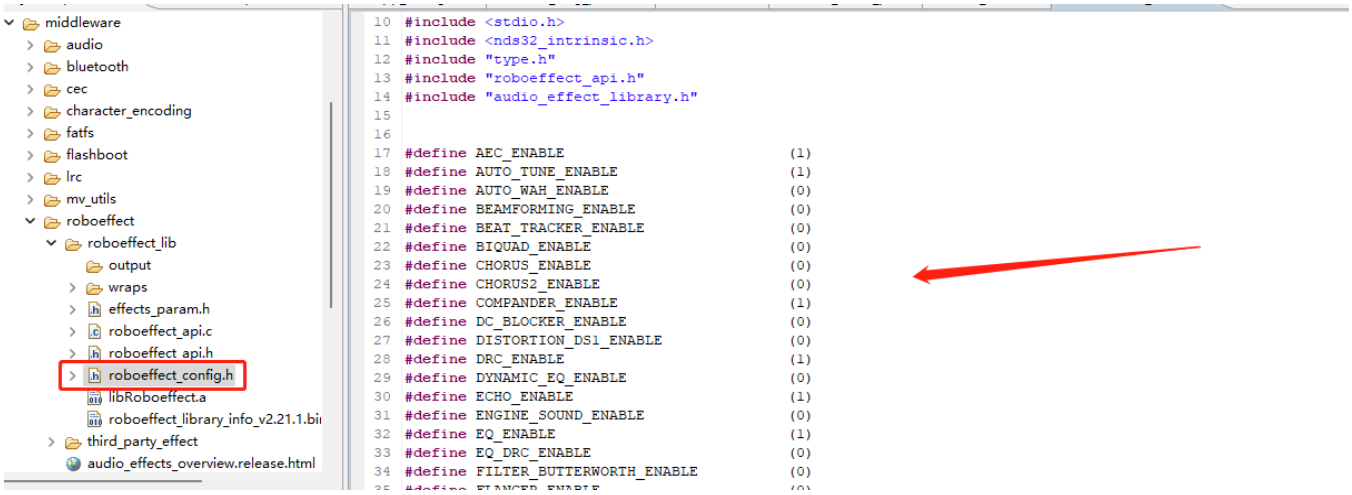
在最新的版本中，音效参数不再通过单独文件导出，直接在音效 flow 文件中一并导出，上位机会自动获取 SDK 当前框图下的所有 effect mode，在音效 flow 文件中也可以看到所有的音效名及对应音效参数。




```
user_effect_flow_karaoke.c
267
268 char *parameter_group_name_Karaoke[7] = { "HunXiang", "DianYin", "MoYin", "HanMai", "NanBianNv", "NvBianNan", "WaWaYin" };
269
270 const unsigned char user_effect_parameters_Karaoke_HunXiang[] = {
271 0xc3, 0x04, /*total data length*/
272
273 0x02, 0x26, 0x00, /*Effect Version*/
274
275 0x02, 0x10, 0x02, /* Roboeffect Version*/
```

3.4.3. roboeffect_config.h

上位机导出的 roboeffect_config.h 文件，主要包含当前框图所使用到的音效宏开关信息。如下图所示，我们需要核对 SDK 最终使用的 roboeffect_config.h，开启对应的音效宏。（路径：BT_Audio_APP\middleware\roboeffect\roboeffect_lib\ roboeffect_config.h）



3.4.4. effect_mode.c

effect_mode.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 和音效控制地址。

在 effect_mode.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。

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

```
const AUDIOEFFECT_EFFECT_PARA HunXiang_effect_para =
{
    .user_effect_name = (uint8_t *)"HunXiang",
    .user_effect_list = (roboeffect_effect_list_info *)&user_effect_list_Karaoke,
    .user_effect_steps = (roboeffect_effect_steps_table *)&user_effect_steps_Karaoke,
    .user_effects_script = (uint8_t *)user_effects_script_Karaoke,
    .user_effect_parameters = (uint8_t *)user_effect_parameters_Karaoke_HunXiang,
    .user_module_parameters = (uint8_t *)user_module_parameters_Karaoke_HunXiang,
    .get_user_effects_script_len = get_user_effects_script_len_Karaoke
};

const AUDIOEFFECT_EFFECT_PARA DianYin_effect_para =
{
    .user_effect_name = (uint8_t *)"DianYin",
    .user_effect_list = (roboeffect_effect_list_info *)&user_effect_list_Karaoke,
    .user_effect_steps = (roboeffect_effect_steps_table *)&user_effect_steps_Karaoke,
    .user_effects_script = (uint8_t *)user_effects_script_Karaoke,
    .user_effect_parameters = (uint8_t *)user_effect_parameters_Karaoke_DianYin,
    .user_module_parameters = (uint8_t *)user_module_parameters_Karaoke_DianYin,
    .get_user_effects_script_len = get_user_effects_script_len_Karaoke
};

const AUDIOEFFECT_EFFECT_PARA MoYin_effect_para =
{
    .user_effect_name = (uint8_t *)"MoYin",
    .user_effect_list = (roboeffect_effect_list_info *)&user_effect_list_Karaoke,
    .user_effect_steps = (roboeffect_effect_steps_table *)&user_effect_steps_Karaoke,
    .user_effects_script = (uint8_t *)user_effects_script_Karaoke,
    .user_effect_parameters = (uint8_t *)user_effect_parameters_Karaoke_MoYin,
    .user_module_parameters = (uint8_t *)user_module_parameters_Karaoke_MoYin,
    .get_user_effects_script_len = get_user_effects_script_len_Karaoke
};
```

source 和 sink 的映射：

```
const AUDIOEFFECT_SOURCE_SINK_NUM karaoke_mode =
{
    //不要删除，source/sink默认值
    AUDIOEFFECT_SOURCE_SINK_DEFAULT_INIT,

    //ROBOEFFECT effect SOURCE映射
    .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,
    .i2s_mix2_source = KARAOKE_SOURCE_I2S_MIX2_SOURCE,
    .linein_mix_source = KARAOKE_SOURCE_LINEIN_MIX_SOURCE,


    //ROBOEFFECT effect SINK映射
    .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,
};
```

音效地址的映射：

```
const uint8_t karaoke_effect_ctrl[AUDIOEFFECT_EFFECT_CONTROL_MAX] =
{
    [EQ_MODE_ADJUST] = KARAOKE_eq0_ADDR,
    [_3D_ENABLE] = KARAOKE_3D_ADDR,
    [ECHO_PARAM] = KARAOKE_echo0_ADDR,
    [MUSIC_VOLUME_ADJUST] = KARAOKE_gain_control0_ADDR,
    [MIC_VOLUME_ADJUST] = KARAOKE_gain_control1_ADDR,
    [MIC_SILENCE_DETECTOR_PARAM] = KARAOKE_silence_detector0_ADDR,
    [MUSIC_SILENCE_DETECTOR_PARAM] = KARAOKE_silence_detector_music_ADDR,
};
```

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 GetEffectControlIndex()、AudioCoreSourceToRoboeffect()、AudioCoreSinkToRoboeffect()三个 API 来自动查找。

3.4.5. effect_control

为了方便和规范 roboeffect 库中，对于参数调节 api 的使用，本工具提供了自动化代码的方法，用以生成一组消息驱动的操作。

修改 params_msg_mapping.xlsx

Message	Message Type	Range/Levels	Control Variable	Flowchart	Effect Name	Parameter	Comm
MSG_MIC_TREB_UP	UP_STOP	-700:800,16	MicTrebStep=0	karaoke	eq7	filter2_gain	
MSG_MIC_TREB_DW	DOWN_STOP	-700:800,16	MicTrebStep=0	karaoke	eq7	filter2_gain	
MSG_MIC_BASS_UP	UP_STOP	-700:800,16	MicBassStep=0	karaoke	eq7	filter1_gain	
MSG_MIC_BASS_DW	DOWN_STOP	-700:800,16	MicBassStep=0	karaoke	eq7	filter1_gain	
MSG_MUSIC_TREB_UP	UP_STOP	-700:800,16	MusicTrebStep=0	karaoke	eq0	filter2_gain	
MSG_MUSIC_TREB_DW	DOWN_STOP	-700:800,16	MusicTrebStep=0	karaoke	eq0	filter2_gain	
MSG_MUSIC_BASS_UP	UP_STOP	-700:800,16	MusicBassStep=0	karaoke	eq0	filter1_gain	
MSG_MUSIC_BASS_DW	DOWN_STOP	-700:800,16	MusicBassStep=0	karaoke	eq0	filter1_gain	
MSG_MIC_EFFECT_UP	UP_LOOP	0:300,32	ReverbStep=0	karaoke	echo0	delay	
^	UP_LOOP	0:1200,32	ReverbStep=0	karaoke	echo0	attenuation	
^	UP_LOOP	0:64,32	ReverbStep=0	karaoke	reverb0	wet	
^	UP_LOOP	0:64,32	ReverbStep=0	karaoke	reverb0	room	
MSG_MIC_EFFECT_DW	UP_LOOP	0:300,32	ReverbStep=0	karaoke	echo0	delay	
^	UP_LOOP	0:1200,32	ReverbStep=0	karaoke	echo0	attenuation	
^	UP_LOOP	0:64,32	ReverbStep=0	karaoke	reverb0	wet	
^	UP_LOOP	0:64,32	ReverbStep=0	karaoke	reverb0	room	
MSG_EFFECT_SYNC	NONE	-700:800,16	MicBassStep	karaoke	eq7	filter1_gain	
^	NONE	-700:800,16	MicTrebStep	karaoke	eq7	filter2_gain	
^	NONE	-700:800,16	MusicBassStep	karaoke	eq0	filter1_gain	
^	NONE	-700:800,16	MusicTrebStep	karaoke	eq0	filter2_gain	
^	NONE	0:300,32	ReverbStep	karaoke	echo0	delay	
^	NONE	0:1200,32	ReverbStep	karaoke	echo0	attenuation	
^	NONE	0:64,32	ReverbStep	karaoke	reverb0	wet	
^	NONE	0:64,32	ReverbStep	karaoke	reverb0	room	

运行自动化代码生成脚本 run.bat，会生成 auto_gen_msg_process.c\ auto_gen_msg_process.h

更多细节可参考《Effect_Control_CodeGen_Manual.release.html》。

3.4.6. EffectModeToggleMap

音效模式按键切换配置表，包含了框图切换的顺序，音效参数节点，MSG 消息处理。

```
typedef struct __UserEffectModeValidConfig
{
    uint8_t                             effect_mode;
    EffectModeState                     effect_mode_state;
    const AUDIOEFFECT_SOURCE_SINK_NUM *source_sink;
    const AUDIOEFFECT_EFFECT_PARA     *effect_para;
    const uint8_t                       *effect_control;
    uint8_t                             (*msg_process)(int msg);
}UserEffectModeValidConfig;
```

```
//音效模式按键切换配置表，顺序切换
//包含音效参数节点，MSG消息处理
static const UserEffectModeValidConfig EffectModeToggleMap[] =
{
#ifdef CFG_AI_DENOISE_EN
    {EFFECT_MODE_MICUSBAI,  EffectModeStateReady,   &micusbAI_mode, &micusbAI_effect_para,  NULL,   NULL},
#endif
#ifdef CFG_FUNC_EFFECT_BYPASS_EN
    {EFFECT_MODE_BYPASS,    EffectModeStateReady,   &bypass_mode,   &bypass_effect_para,    NULL,   NULL},
#else
    #ifdef CFG_FUNC_MIC_KARAOKE_EN
        {EFFECT_MODE_HunXiang,  EffectModeStateReady,   &mic_mode,      &mic_effect_para,       mic_effect_ctrl,    NULL},
        {EFFECT_MODE_DianYin,   EffectModeStateReady,   &karaoke_mode,  &DianYin_effect_para,   karaoke_effect_ctrl,    msg_process_karaoke},
        {EFFECT_MODE_MoYin,     EffectModeStateReady,   &karaoke_mode,  &MoYin_effect_para,     karaoke_effect_ctrl,    msg_process_karaoke},
        {EFFECT_MODE_HanMai,    EffectModeStateReady,   &karaoke_mode,  &HanMai_effect_para,    karaoke_effect_ctrl,    msg_process_karaoke},
        {EFFECT_MODE_NanBianNv, EffectModeStateReady,   &karaoke_mode,  &NanBianNv_effect_para, karaoke_effect_ctrl,    msg_process_karaoke},
        {EFFECT_MODE_NvBianNan, EffectModeStateReady,   &karaoke_mode,  &NvBianNan_effect_para, karaoke_effect_ctrl,    msg_process_karaoke},
        {EFFECT_MODE_WaWaYin,   EffectModeStateReady,   &karaoke_mode,  &WaWaYin_effect_para,   karaoke_effect_ctrl,    msg_process_karaoke},
    #else
        {EFFECT_MODE_MIC,       EffectModeStateReady,   &mic_mode,      &mic_effect_para,       mic_effect_ctrl,    NULL    },
        {EFFECT_MODE_MUSIC,     EffectModeStateReady,   &music_mode,    &music_effect_para,     music_effect_ctrl,  NULL    },
        {EFFECT_MODE_DianYin,   EffectModeStateReady,   &karaoke_mode,  &DianYin_effect_para,   karaoke_effect_ctrl,    NULL},
    #endif
#endif
#if defined(CFG_APP_BT_MODE_EN) && (BT_HFP_SUPPORT)
    {EFFECT_MODE_HFP_AEC,   EffectModeStateSuspend, &hfp_mode,      &hfp_effect_para,   hfp_effect_ctrl,    NULL},
#endif
};
```

effect_mode： 音效框图的编号名称，比如 EFFECT_MODE_HunXiang
effect_mode_state：运行状态，挂起的时候按键不能切换。
source_sink：source 和 sink 的映射关系
effect_para：音效参数
effect_control：音效地址的映射关系，可以没有
msg_process：effect_control 工具生成的消息处理函数入口，可以没有。

3.5. Roboeffect Init

3.5.1. 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_mode.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;
```

3.5.2. 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");
```

3.5.3. 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。

3.6. 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));
```

3.7. 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()
```

3.8. 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。

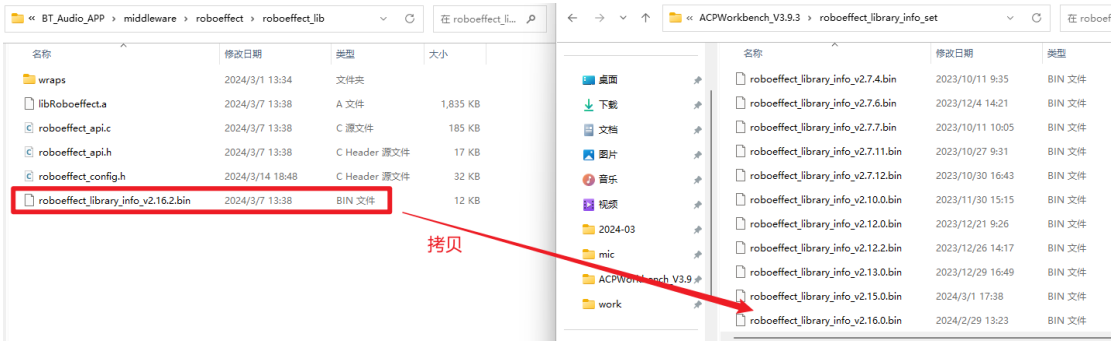
4. 定制音效

在这一章节，我们将从 0 开始，基于 SDK 添加一个名为“example”的音效。

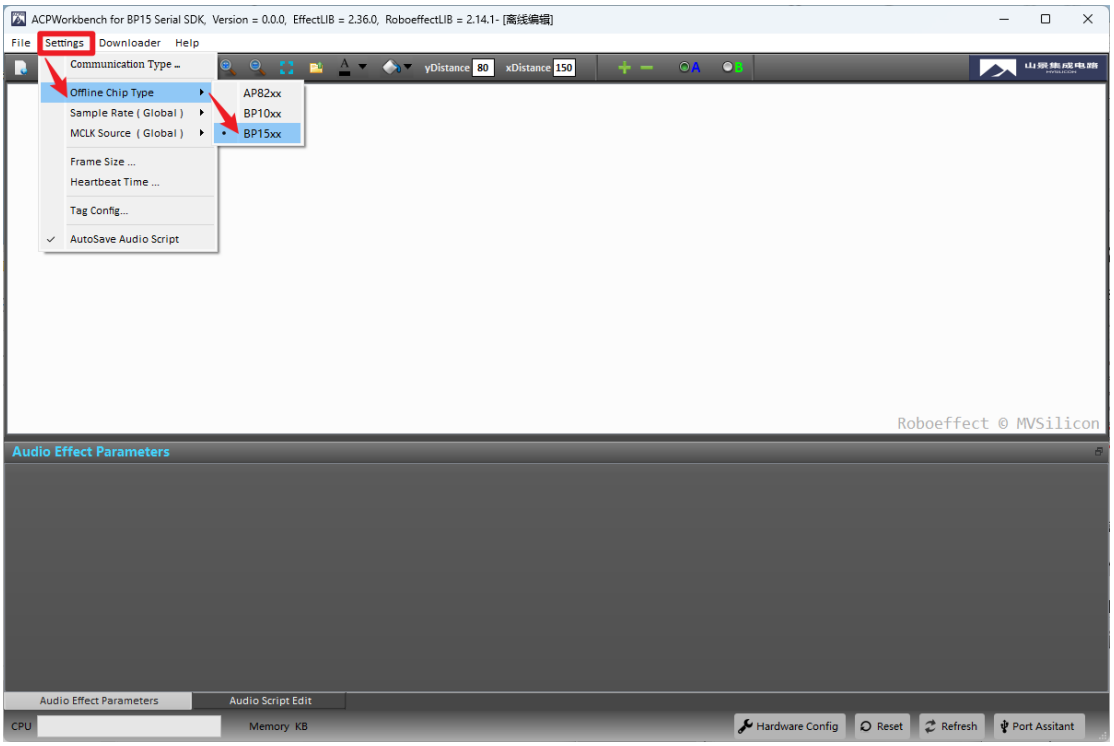
4.1. 上位机绘制框图

4.1.1. 准备工作

1. 拷贝 SDK roboeffect_library_info_vX.X.X.bin 至调音工具对应路径下

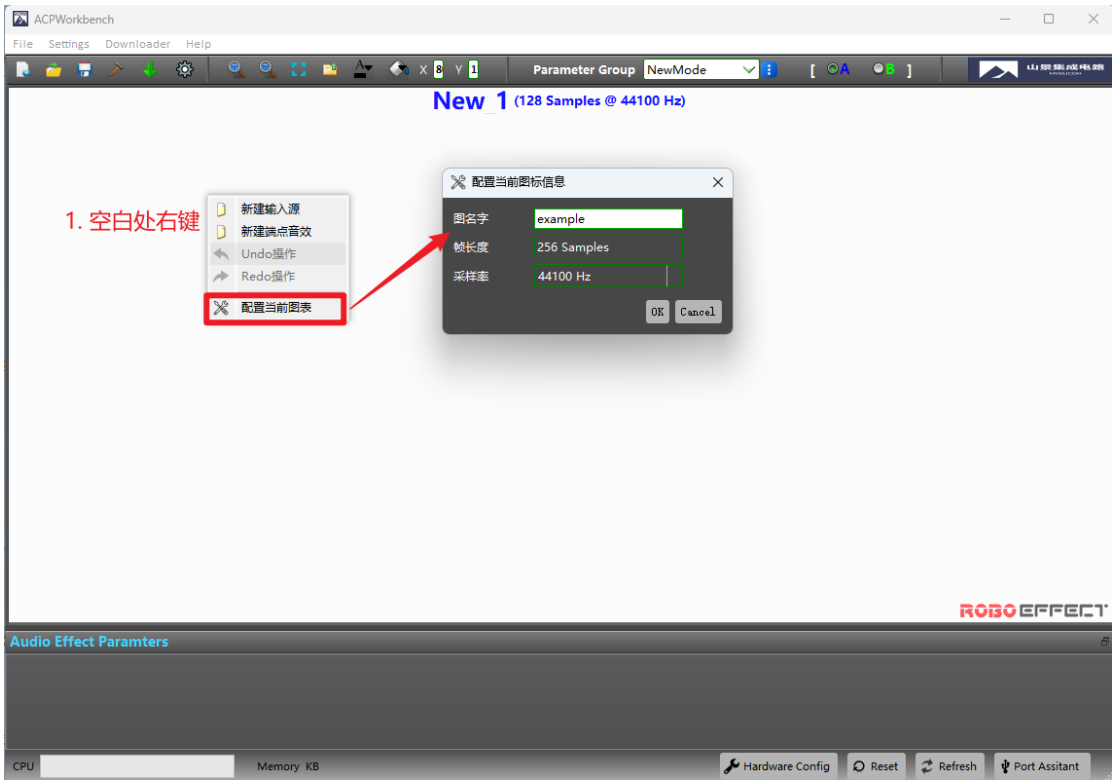


2. 打开调音工具选择正确的芯片型号

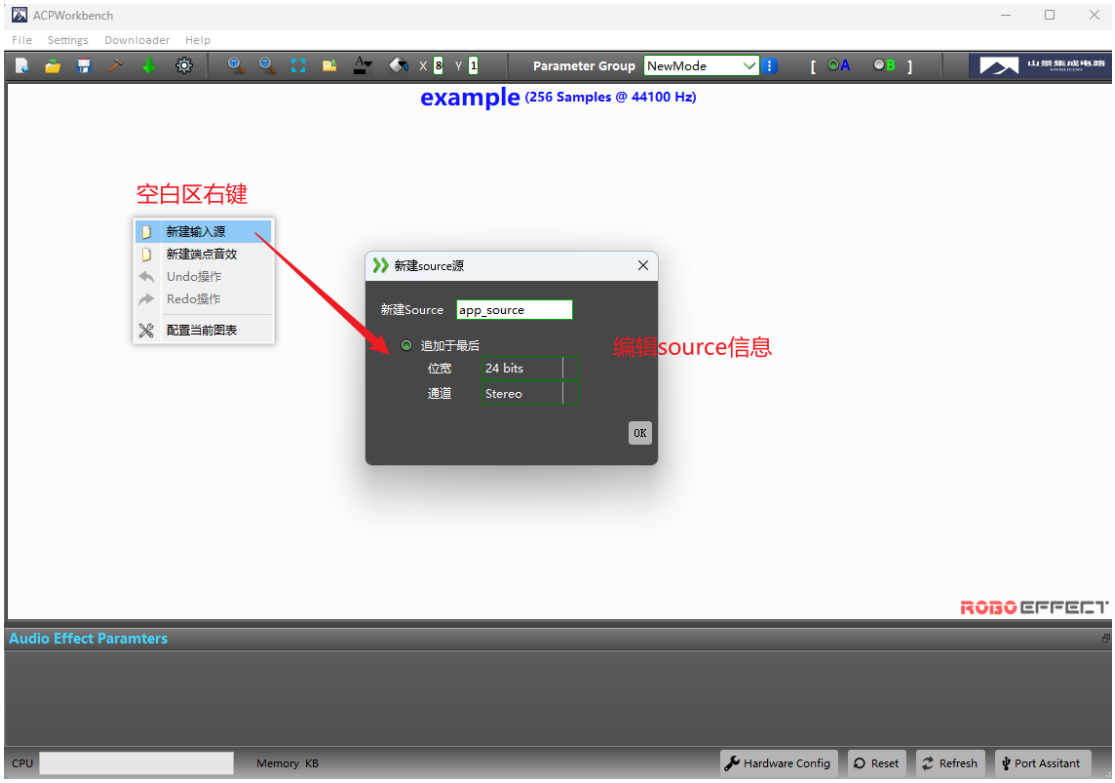


4.1.2. 修改图名称

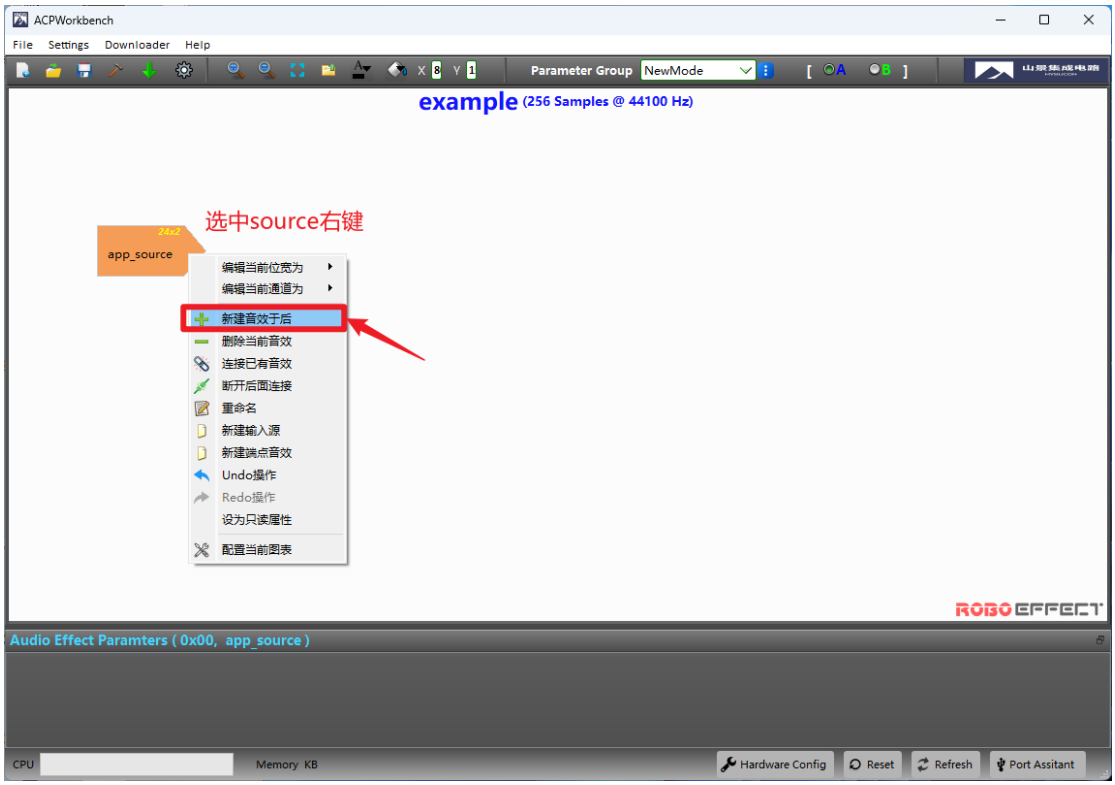
打开上位机，会有默认的框图名称及帧长和采样率，第一步首先将其修改。



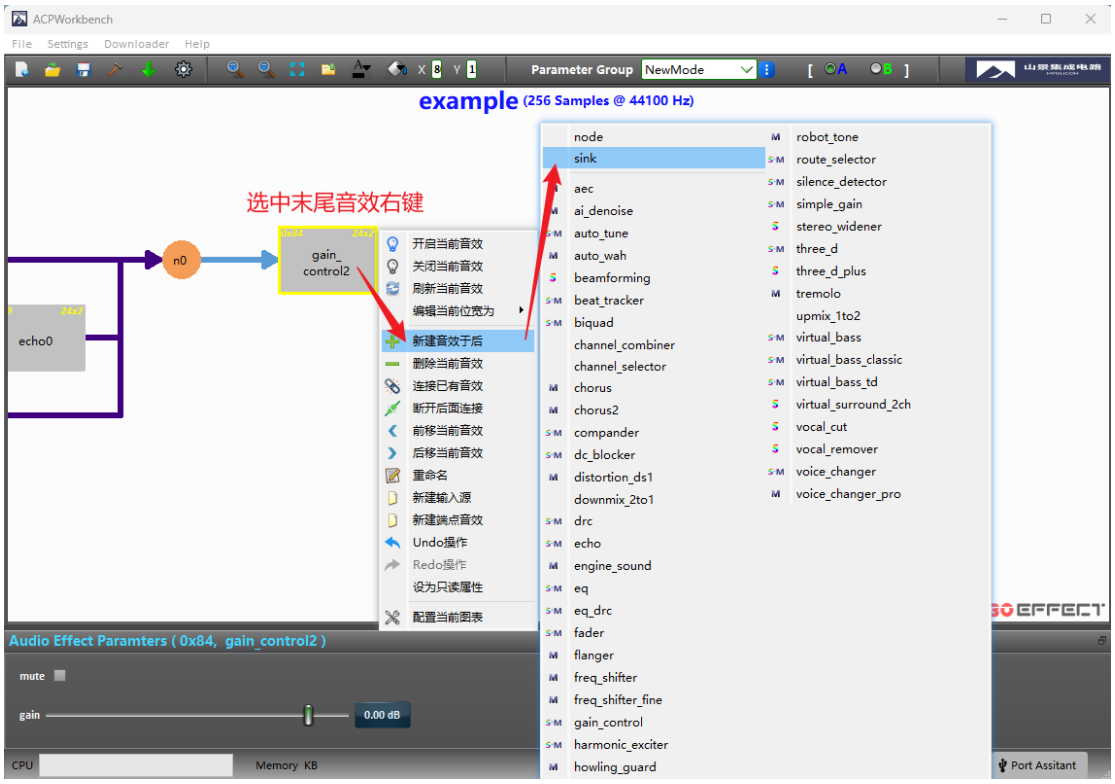
4.1.3. 新建 source 输入



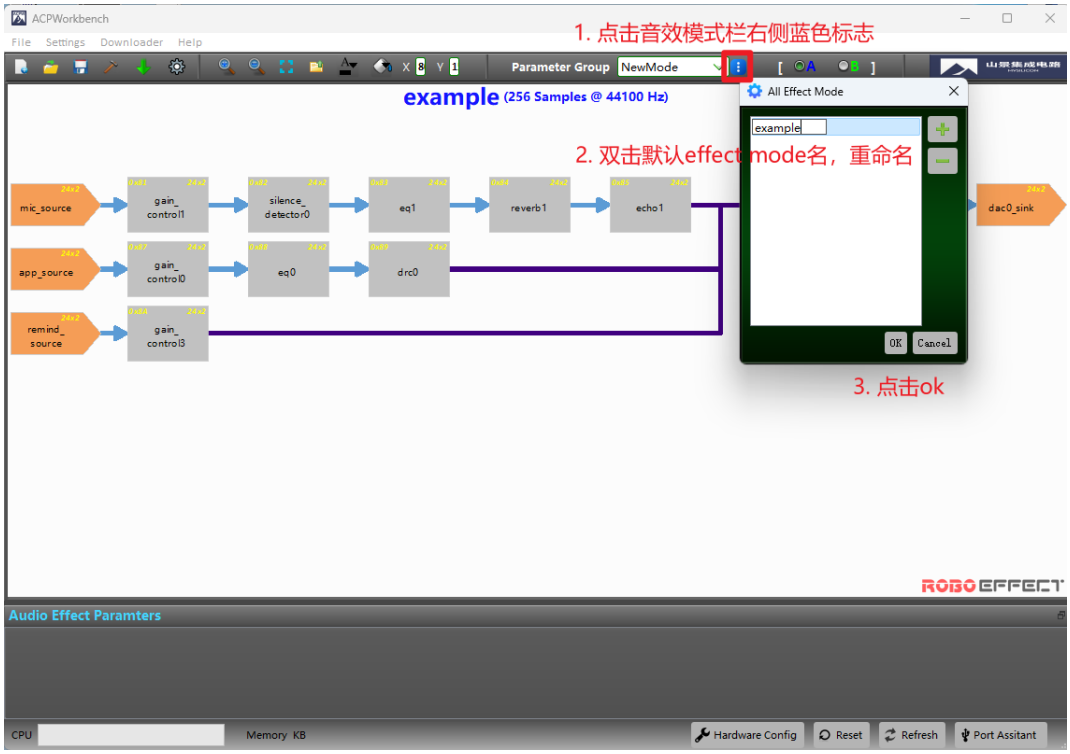
4.1.4. 增加音效



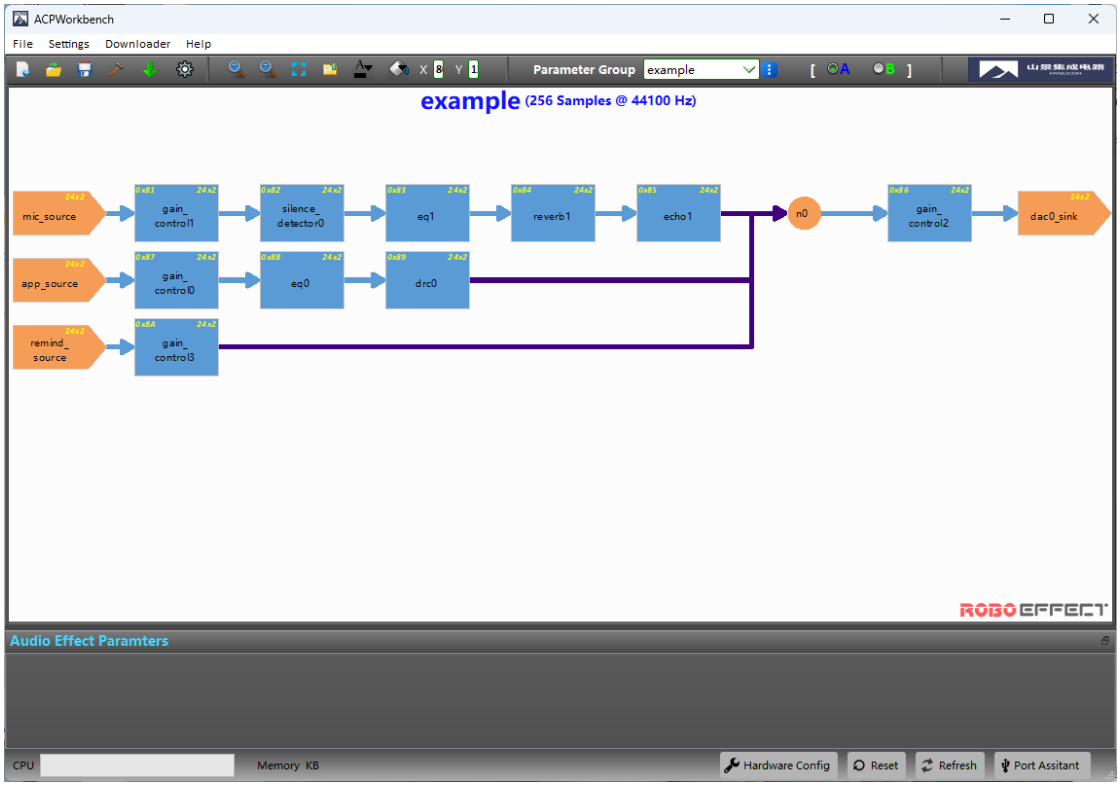
4.1.5. 新建 sink 输出



4.1.6. 设置音效模式名



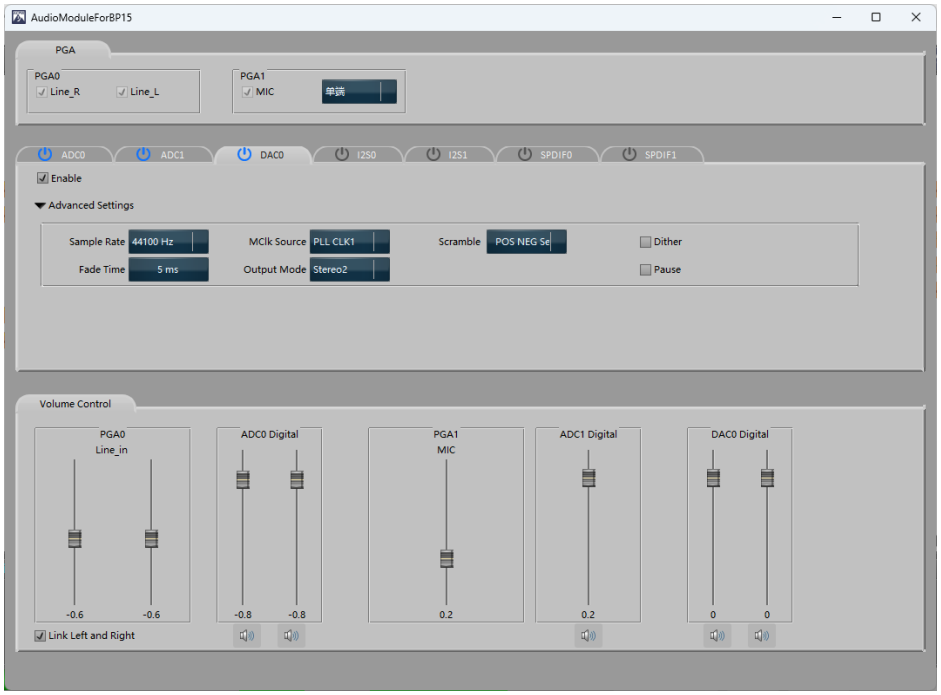
4.2. 导出音效



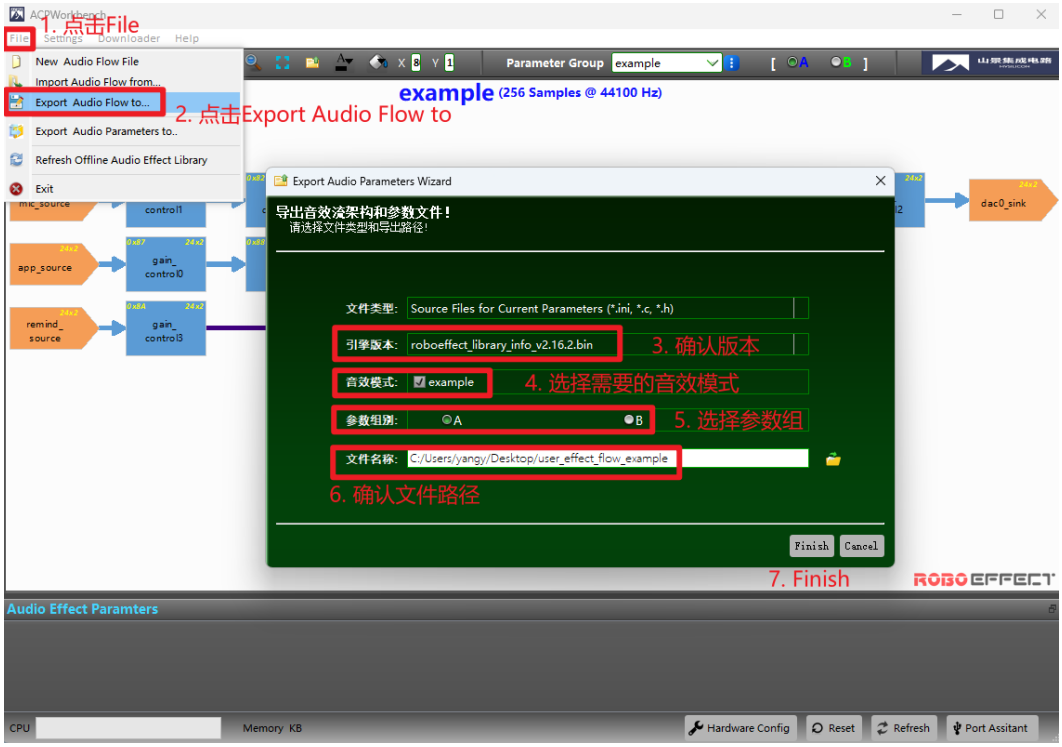
简单调整音效参数后，我们已经完成了一个基本的框图绘制。

4.2.1. 调整 hardware config

点击系统控制区的 Hardware Config 按钮，会弹出硬件控制界面，我们调整下默认的 DAC0 Digital gain 等各项参数。然后导出至 SDK。



4.2.2. 导出音效文件

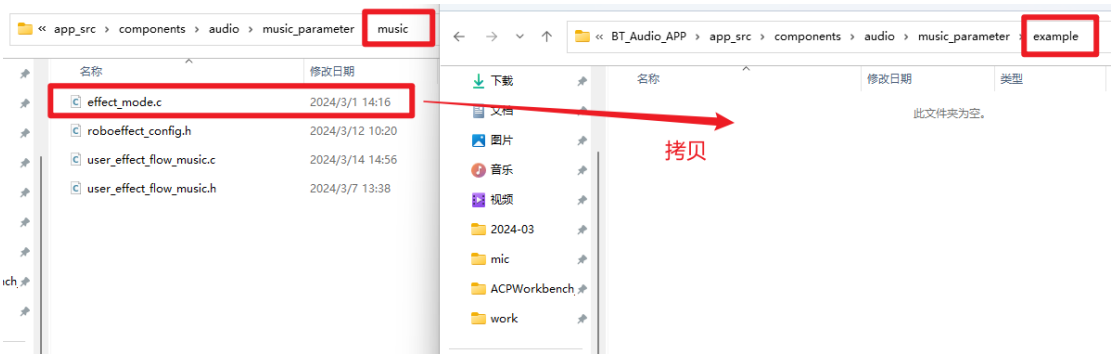


引擎版本可通过搜索 SDK 宏 **ROBOEFFECT_LIB_VER** 来确认。

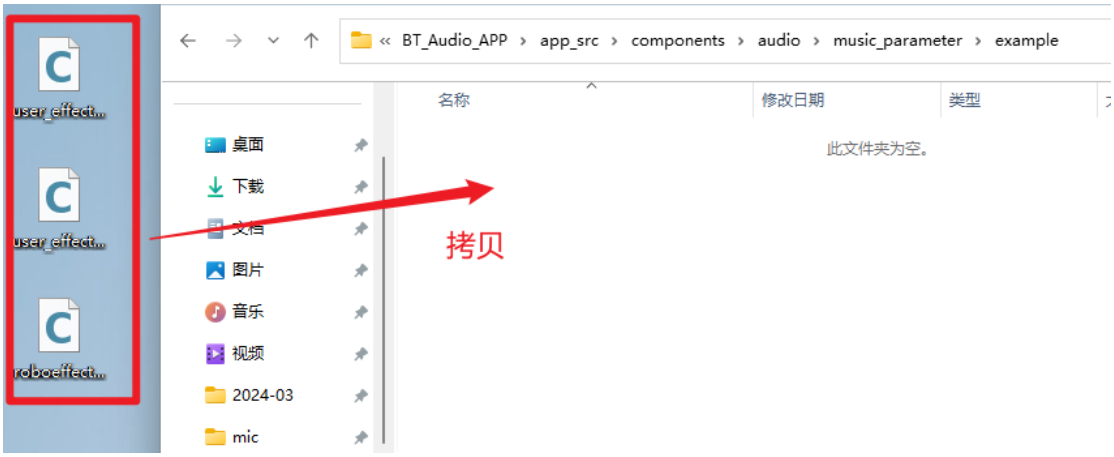
4.3. 新建音效路径及 effect_mode.c

1. 在音效并列目录下新建 example 文件夹；

2. 拷贝 music 路径下的 effect_mode.c 至新建 example 文件夹；



3. 拷贝导出的音效文件至新建 example 文件夹。



4.4. 修改 SDK 代码

1. 增加新音效枚举；

```
typedef enum _EFFECT_MODE
{
    EFFECT_MODE_DEFAULT = 0,
    /**********mic mode***************/
    EFFECT_MODE_MIC,
    /**********music mode**************/
    EFFECT_MODE_MUSIC,
    /**********example mode************/
    EFFECT_MODE_EXAMPLE,
    /**********bypass mode*************/
    EFFECT_MODE_BYPASS,
    /**********hfp mode****************/
    EFFECT_MODE_HFP_AEC,
    /**********karaoke mode************/
    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;
```

2. 修改 effect_mode.c 中 source 映射和 sink 映射；

```
#include "user_effect_flow_example.h"
#include "user_effect_parameter.h"

const AUDIOEFFECT_EFFECT_PARA example_effect_para =
{
    .user_effect_name = (uint8_t *)"example",
    .user_effect_list = (roboeffect_effect_list_info *)&user_effect_list_example,
    .user_effect_steps = (roboeffect_effect_steps_table *)&user_effect_steps_example,
    .user_effects_script = (uint8_t *)user_effects_script_example,
    .user_effect_parameters = (uint8_t *)user_effect_parameters_bypass_example,
    .user_module_parameters = (uint8_t *)user_module_parameters_bypass_example,
    .get_user_effects_script_len = get_user_effects_script_len_example,
};

const AUDIOEFFECT_SOURCE_SINK_NUM example_mode =
{
    //不要删除，source/sink默认值
    AUDIOEFFECT_SOURCE_SINK_DEFAULT_INIT,

    //ROBOEFFECT effect SOURCE映射
    .mic_source = EXAMPLE_SOURCE_MIC_SOURCE,
    .app_source = EXAMPLE_SOURCE_APP_SOURCE,

    //ROBOEFFECT effect SINK映射
    .dac0_sink = EXAMPLE_SINK_DAC0_SINK,
    .app_sink = EXAMPLE_SINK_APP_SINK,
};
```

3. EffectModeToggleMap 增加新音效；

```
56
57
58  //音效模式按键切换配置表，顺序切换
59  //包含音效参数节点，MSG消息处理
60  static const UserEffectModeValidConfig EffectModeToggleMap[] =
61  {
62      {EFFECT_MODE_Example,   EffectModeStateReady,   &example_mode,      &example_effect_para,       NULL,   NULL},
63  #ifdef CFG_AI_DENOISE_EN
64      {EFFECT_MODE_MICUSBAI,  EffectModeStateReady,   &micusbAI_mode, &micusbAI_effect_para,  NULL,   NULL},
65  #endif
66  #ifdef CFG_FUNC_EFFECT_BYPASS_EN
67      {EFFECT_MODE_BYPASS,    EffectModeStateReady,   &bypass_mode,   &bypass_effect_para,    NULL,   NULL},
68  #else
69      #ifdef CFG_FUNC_MIC_KARAOKE_EN
```

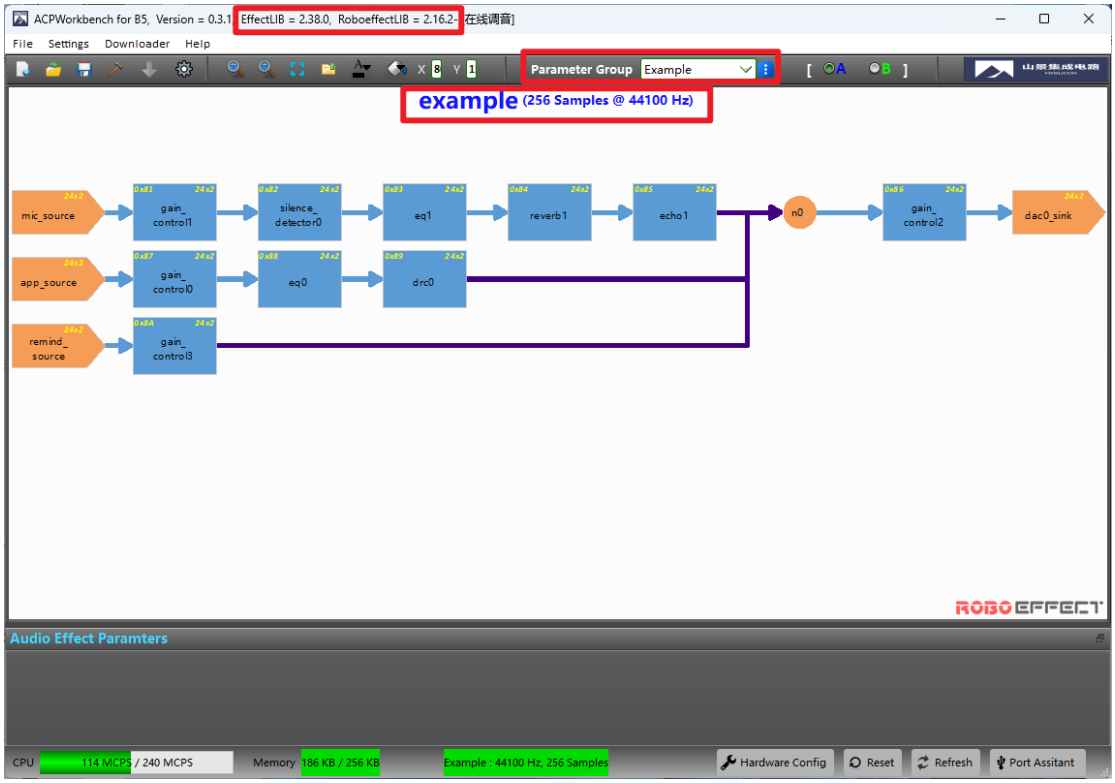
4. 修改默认音效为新增音效。

```
//各个模块默认参数设置函数
void DefaultParamgsInit(void)
{
    memset(&gCtrlVars,  0, sizeof(gCtrlVars));
    //for system control 0x01
    gCtrlVars.AutoRefresh = AutoRefresh_ALL_PARA;

    if(AudioCore.Audioeffect.context_memory)
    {
        memcpy(&gCtrlVars.HwCt, AudioCore.Audioeffect.user_module_parameters, sizeof(gCtrlVars.HwCt));
    }
    else
    {
        AUDIOEFFECT_EFFECT_PARA *para;
        if (mainAppCt.EffectMode  == 0)
        {
#ifdef CFG_FUNC_EFFECT_BYPASS_EN
            mainAppCt.EffectMode = EFFECT_MODE_BYPASS;
#else
#ifdef CFG_FUNC_MIC_KARAOKE_EN
            mainAppCt.EffectMode = EFFECT_MODE_HunXiang;
#else
            mainAppCt.EffectMode = EFFECT_MODE_EXAMPLE;
#endif
#endif
        }
        para = get_user_effect_parameters(mainAppCt.EffectMode);
        memcpy(&gCtrlVars.HwCt, para->user_module_parameters, sizeof(gCtrlVars.HwCt));
    }
```

4.5. 编译烧录确认

音效模式有断电记忆，因此请烧录时请选择全擦除。



5. SDK 音效控制

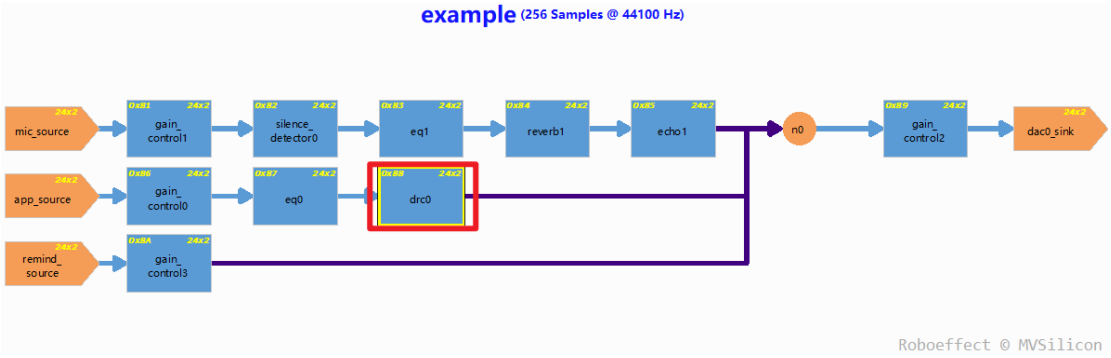
上一章节，我们演示了如何从 0 开始，新增一个音效模式导入到 SDK，接下来，我们来演示如何控制我们新增的音效。

5.1. 开始之前

在开始之前，我们首先需要明白在 V3 架构中，每个音效是以 0x81 这样的地址的形式存在，对于 SDK 来讲，SDK 并不知道具体的应用功能是需要控制哪个框图中的哪个音效，SDK 只是预设某个功能函数是对 EQ 的高低音进行控制或者 gain 大小控制等等，至于如何拿到正确的音效地址，需要通过 effect_mode.c 中的音效地址映射来告诉 SDK。

5.2. 音效开关

假设我们现在想要通过按键控制框图中标注的 DRC 的动态开关。



1. AUDIOEFFECT_EFFECT_CONTROL 增加枚举 DRC；

```
/**
 * @brief Audio effect adjust
 */
typedef enum _AUDIOEFFECT_EFFECT_CONTROL
{
    EQ_MODE_ADJUST = 0,
    MIC_BASS_ADJUST,
    MIC_TREB_ADJUST,
    MUSIC_BASS_ADJUST,
    MUSIC_TREB_ADJUST,
    MIC_VOLUME_ADJUST,
    MUSIC_VOLUME_ADJUST,
    ECHO_PARAM,
    MIC_SILENCE_DETECTOR_PARAM,
    MUSIC_SILENCE_DETECTOR_PARAM,
    _3D_ENABLE,
    DRC_CTRL,
    AUDIOEFFECT_EFFECT_CONTROL_MAX
}AUDIOEFFECT_EFFECT_CONTROL;
```

2. 音效地址映射 DRC_ADDR 指向框图中实际音效；

```
const uint8_t mic_effect_ctrl[AUDIOEFFECT_EFFECT_CONTROL_MAX] =
{
    [EQ_MODE_ADJUST] = MIC_mic_eq0_ADDR,
    [MUSIC_VOLUME_ADJUST] = MIC_gain_control0_ADDR,
    [MIC_VOLUME_ADJUST] = MIC_mic_gain_ADDR,
    [MIC_SILENCE_DETECTOR_PARAM] = MIC_silence_detector_mic_ADDR,
    [MUSIC_SILENCE_DETECTOR_PARAM] = MIC_silence_detector_music_ADDR,

    [DRC_CTRL] = MIC_mic_drc0_ADDR,
};
```

3. 新增 DRC 开关消息 id；

```
    MSG_MIC_VOLUP,
    MSG_MIC_VOLDOWN,
    MSG_MIC_EFFECT_UP,
    MSG_MIC_EFFECT_DW,
    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,
    MSG_PITCH_UP,              //变调加
    MSG_PITCH_DN,              //变调减
    MSG_VOCAL_CUT,
    MSG_MUTE,
    MSG_EQ,
    MSG_3D,
    MSG_VB,
    MSG_DRC_ONOFF,
    MSG_REMIND1,
    MSG_EFFECTMODE,
```

4. 新增消息处理逻辑；

```
refresh_addr = GetEffectControlIndex(DRC_CTRL);//获取地址
if(refresh_addr)
{
    bool enable= AudioEffect_effect_status_Get(refresh_addr);
    AudioEffect_effect_enable(refresh_addr, !enable);
}
```

5. 设置按键消息；

```
//adc2 key
{MSG_NONE,                  MSG_BT_HF_VOICE_RECOGNITION,
{MSG_NONE,                  MSG_FOLDER_PRE,
{MSG_NONE,                  MSG_FOLDER_NEXT,
{MSG_NONE,                  MSG_RTC_SET_TIME,
{MSG_NONE,                  MSG_RTC_SET_ALARM,
{MSG_NONE,                  MSG_RTC_UP,
{MSG_NONE,                  MSG_RTC_DOWN,
{MSG_NONE,                  MSG_BT_CONNECT_CTRL,
{MSG_NONE,                  MSG_BT_CONNECT_MODE,
{MSG_NONE,                  MSG_MUSIC_TREB_UP,
{MSG_NONE,                  MSG_DRC_ONOFF,
```

6. 编译烧录，测试 ok。

5.3. 音量控制

音量控制的实现相对简单，作为 SDK 的基本功能之一，只需在音效地址映射时选择想要控制的 gain_control 即可，无需再另外新增特别代码。注意用于音量控制的 gain_control 需要默认打开。

```
};

const uint8_t music_effect_ctrl[AUDIOEFFECT_EFFECT_CONTROL_MAX] =
{
    [EQ_MODE_ADJUST] = MUSIC_eq0_ADDR,
    [MUSIC_VOLUME_ADJUST] = MUSIC_gain_control0_ADDR,
    [MIC_VOLUME_ADJUST] = MUSIC_mic_gain_ADDR,
    [MIC_SILENCE_DETECTOR_PARAM] = MUSIC_silence_detector_mic_ADDR,
    [MUSIC_SILENCE_DETECTOR_PARAM] = MUSIC_silence_detector_music_ADDR,
};
```

5.4. EQ 控制

1. 开启宏 CFG_FUNC_MUSIC_EQ_MODE_EN；

```
#define CFG_FUNC_MUSIC_EQ_MODE_EN      //Music EQ模式功能配置
```

2. 音效地址映射匹配要调节的 EQ；

```
const uint8_t music_effect_ctrl[AUDIOEFFECT_EFFECT_CONTROL_MAX] =
{
    [EQ_MODE_ADJUST] = MUSIC_eq0_ADDR,
    [MUSIC_VOLUME_ADJUST] = MUSIC_gain_control0_ADDR,
    [MIC_VOLUME_ADJUST] = MUSIC_mic_gain_ADDR,
    [MIC_SILENCE_DETECTOR_PARAM] = MUSIC_silence_detector_mic_ADDR,
    [MUSIC_SILENCE_DETECTOR_PARAM] = MUSIC_silence_detector_music_ADDR,
};
```

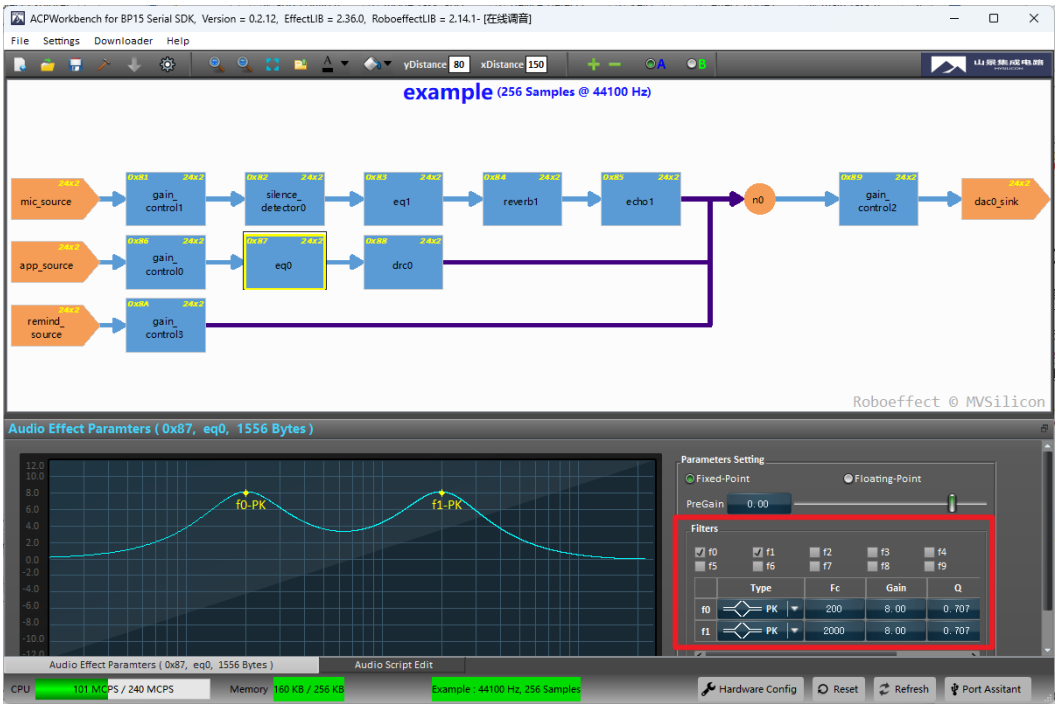
3. 按键测试和代码实现；

```
#if(defined(BT_SNIFF_ENABLE))
    {MSG_NONE,                  MSG_PLAY_PAUSE,                 MSG_BT_SNIFF,               MSG
#else
    {MSG_NONE,                  MSG_PLAY_PAUSE,                 MSG_DEEPSLEEP,              MSG
#endif
    {MSG_NONE,                  MSG_PRE,                        MSG_FB_START,               MSG
    {MSG_NONE,                  MSG_NEXT,                       MSG_FF_START,               MSG
    {MSG_NONE,                  MSG_MUSIC_VOLDOWN,              MSG_MUSIC_VOLDOWN,          MSG
    {MSG_NONE,                  MSG_MUSIC_VOLUP,                MSG_MUSIC_VOLUP,            MSG
    {MSG_NONE,                  MSG_EQ,                         MSG_3D,                     MSG
    {MSG_NONE,                  MSG_MUTE,                       MSG_VB,                     MSG
    {MSG_NONE,                  MSG_EFFECTMODE,                 MSG_VOCAL_CUT,              MSG
#if (BT_SOURCE_SUPPORT )
    {MSG_NONE,                  MSG_BT_SOURCE_DISCONNECT,/*MSG_REPEAT,*/                    MSG
    {MSG_NONE,                  MSG_BT_SOURCE_INQUIRY,/*MSG_REC,*/                          MSG_REC
```

```
uint8_t AudioCommonMsgProcess(uint16_t Msg)
{
    uint8_t     refresh_addr = 0;

    switch(Msg)
    {
#ifdef CFG_FUNC_MUSIC_EQ_MODE_EN
    case MSG_EQ:
        if(++EqMode >= 6)
            EqMode = 0;
        refresh_addr = AudioEqSync();
        break;
#endif
```

4. 监听声音同时上位机观察对应参数变化。



5.5. silence detector

silence detector 音效的功能在 SDK 中也有完善的应用可以参考，这里我们简单演示下打印出 silence detector 的值。

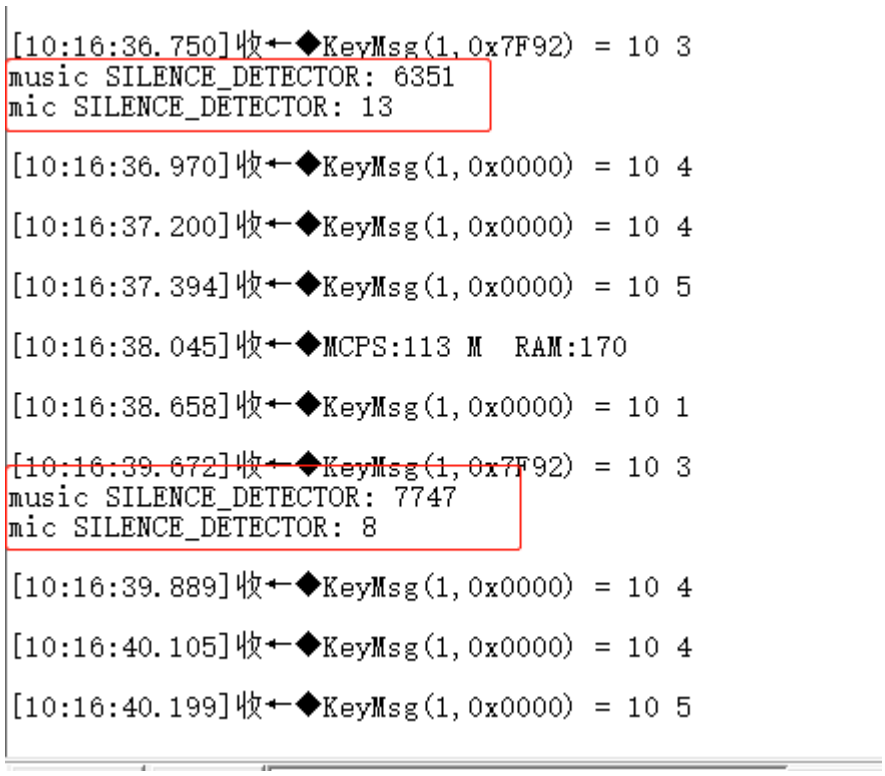
1. 匹配音效地址；

```
33
34 const uint8_t music_effect_ctrl[AUDIOEFFECT_EFFECT_CONTROL_MAX] =
35 {
36     [EQ_MODE_ADJUST] = MUSIC_eq0_ADDR,
37     [MUSIC_VOLUME_ADJUST] = MUSIC_gain_control0_ADDR,
38     [MIC_VOLUME_ADJUST] = MUSIC_mic_gain_ADDR,
39     [MIC_SILENCE_DETECTOR_PARAM] = MUSIC_silence_detector_mic_ADDR,
40     [MUSIC_SILENCE_DETECTOR_PARAM] = MUSIC_silence_detector_music_ADDR,
41 };
42
```

2. 增加一个自己的按键消息 MSG_TEST 进行打印；用 AudioEffectGetAllParameter 获取参数信息；

```
#endif
    case MSG_TEST:
        {
            SilenceDetectorUnit *Param;
            Param = AudioEffectGetAllParameter(MUSIC_SILENCE_DETECTOR_PARAM);
            DBG("music SILENCE_DETECTOR: %d\n",Param->level);
            Param = AudioEffectGetAllParameter(MIC_SILENCE_DETECTOR_PARAM);
            DBG("mic SILENCE_DETECTOR: %d\n",Param->level);
        }
        break;
```

3. 测试打印。

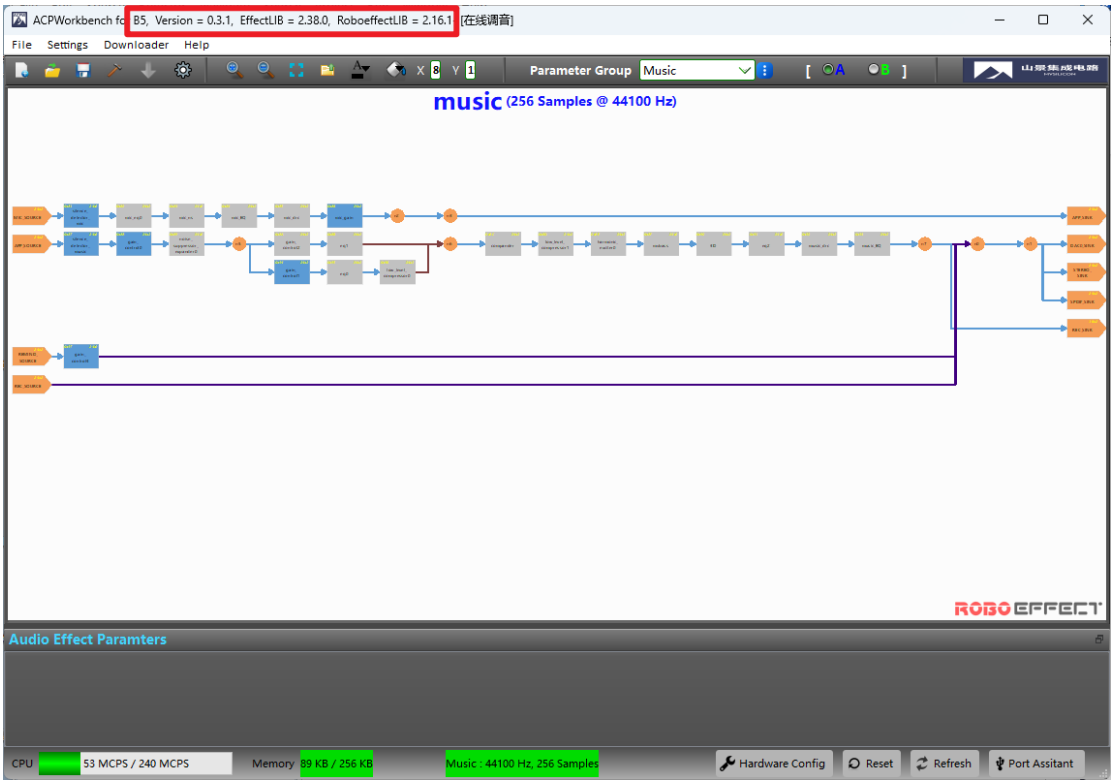


6. 音效库与 roboeffect 库的升级

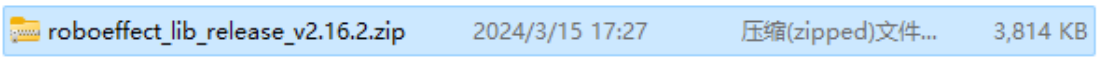
这一章节我们演示基于 SDK v0.3.1 版本从 roboeffect 库 v2.16.1 升级至 v2.16.2 的详细过程。V3 架构音效库与 roboeffect 库互相深度绑定，两者需要同步升级。

6.1. 环境准备

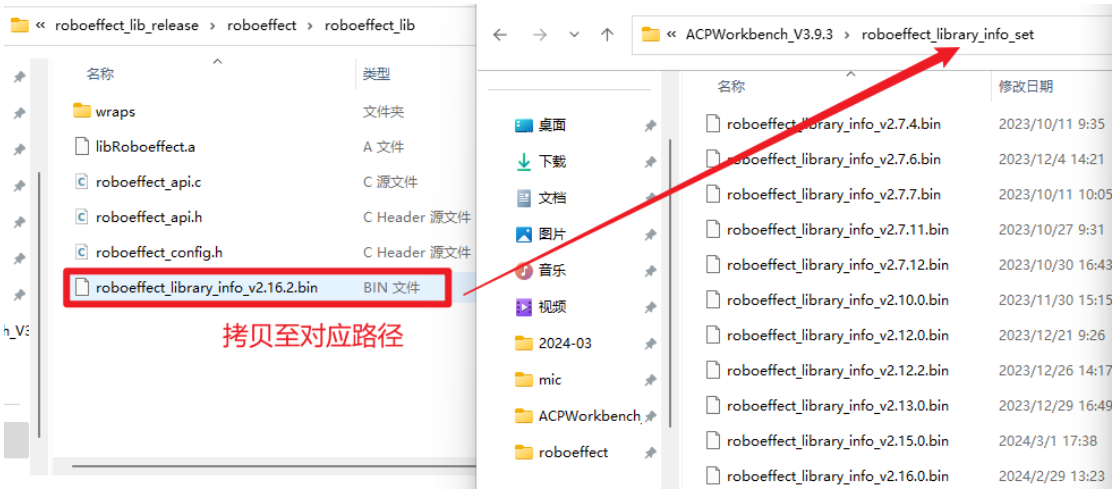
1. 一套完整的运行环境：SDK+开发板+ACPWorkBench（通常情况下，ACPWorkBench 使用手上最新版本即可）



2. 目标版本 roboeffect_lib_release 库压缩包



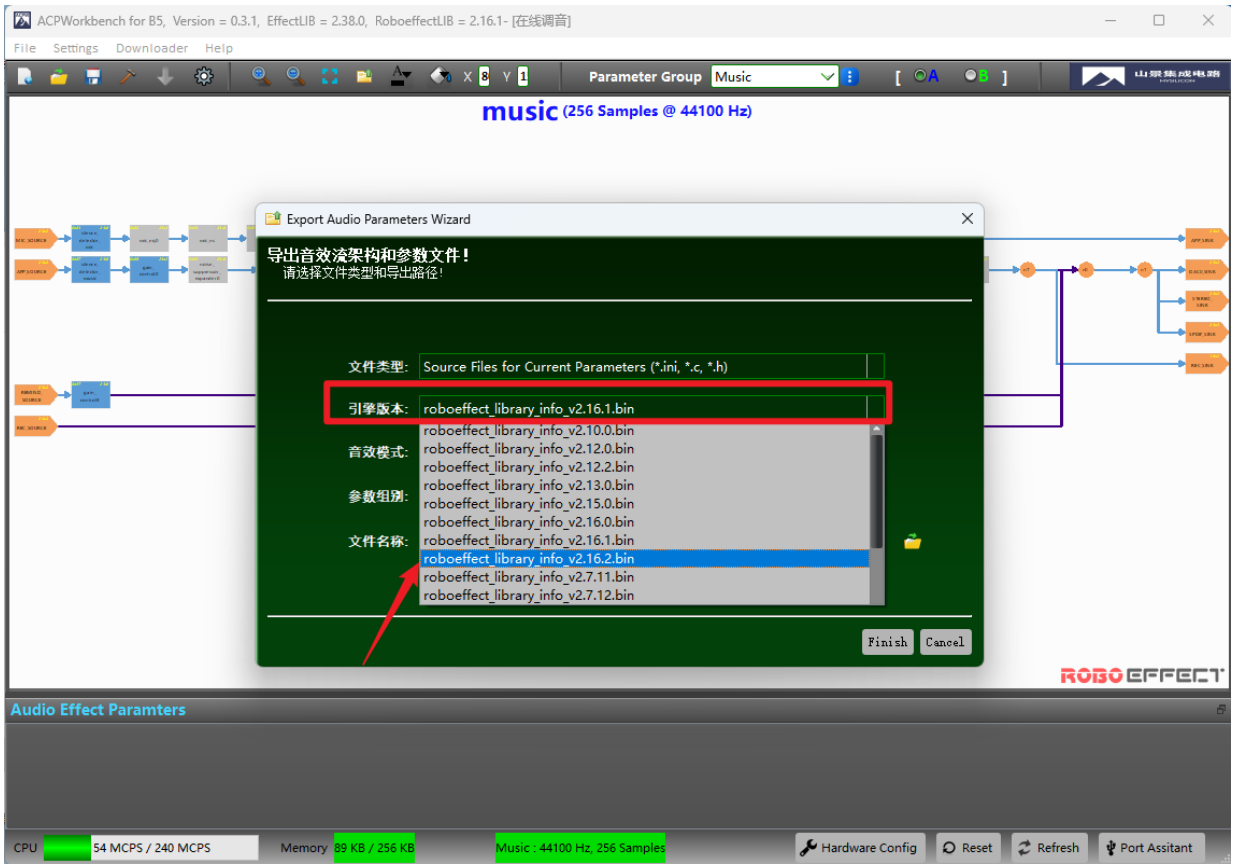
6.2. roboeffect_library_info_vX.X.X.bin



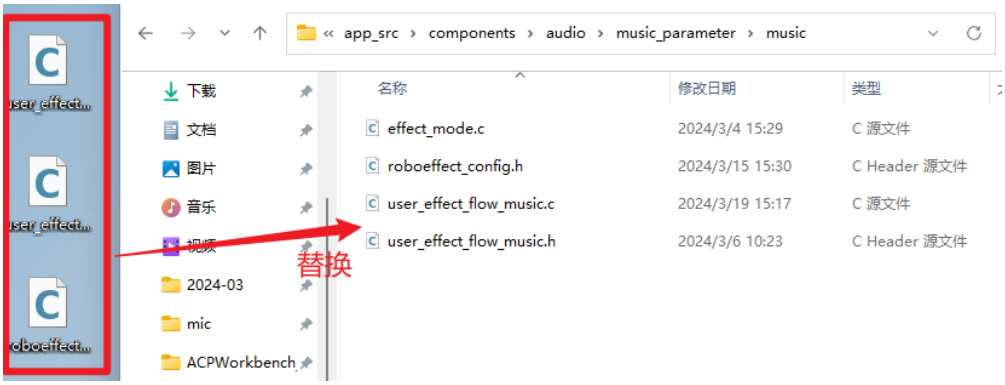
roboeffect_library_info_vX.X.X.bin 中包含了当前版本音效库的音效参数信息，在开始更新之前我们首先需要将其拷贝至调音工具的对应路径下。

6.3. 更新音效文件

1. 连接上位机
2. 基于目标版本导出音效 flow 文件

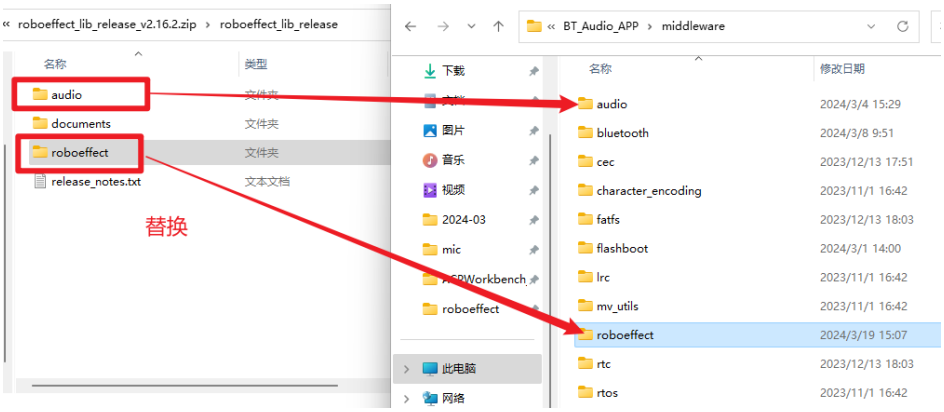


3. 替换音效文件

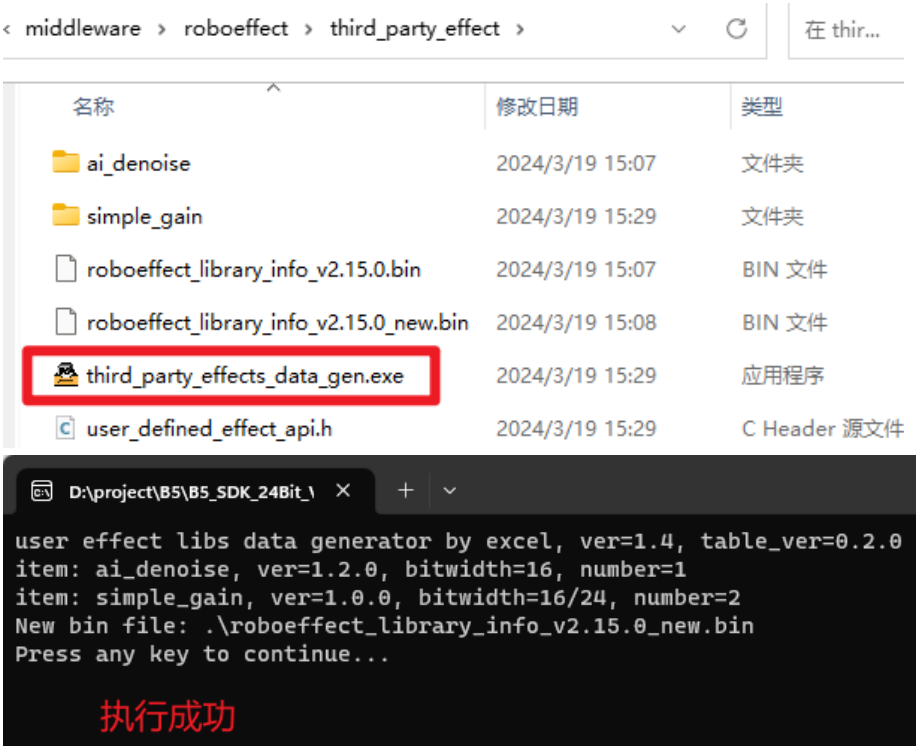


6.4. 更新库

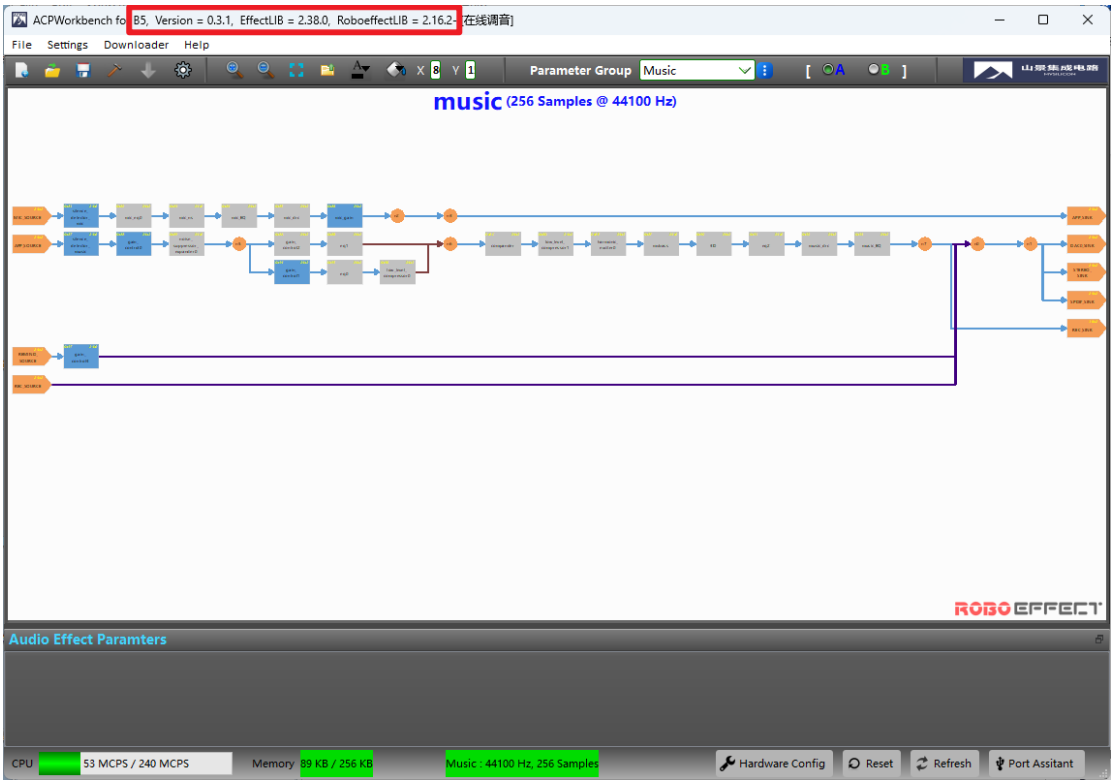
1. 替换音效库+roboeffect 库



2. 双击执行 third_party_effects_data_gen 脚本



3. 编译烧录确认版本是否升级成功


ACPWorkbench fo
B5, Version = 0.3.1, EffectLIB = 2.38.0, RoboeffectLIB = 2.16.2-
在线调音]
File
Settings
Downloader
Help
Parameter Group
Music
music (256 Samples @ 44100 Hz)
ROBOEFFECT
Audio Effect Paramters
CPU
53 MCPS / 240 MCPS
Memory
89 KB / 256 KB
Music : 44100 Hz, 256 Samples
Hardware Config
Reset
Refresh
Port Assitant

7. 定制开发说明

7.1. 建议开发流程

1. 确定产品形态，确定输入输出及音频通路；
2. 软件工程师编写代码，完成输入输出相关 IO 流程；
3. 调音工程师在上位机修改音效，调音；
4. 软件工程师最后进行代码层面定制化音效控制功能完善。

上述流程中，调音工程师仅需在上位机修改框图导出相关音效文件至 SDK 工程编译烧录，无需手动修改任何代码。

代码层面的修改全部由软件工程师来完成，同时软件工程师也需要在上位机修改和导出音效文件，确认输入输出通道数据的正确性。

7.2. 删减 source/sink

参考 4.1 小节，先在上位机删掉对应 source/sink，然后导出音效文件至 SDK，此时直接编译会报错，删除对应的映射就可以。SDK 第一行代码会将 effect_mode.c 的 source/sink 定义为 ERROR 状态。AudioCore 中数据获取输出的逻辑代码是否要删掉，由软件工程师自行把控。

```
14
15 const AUDIOEFFECT_SOURCE_SINK_NUM music_mode =
16 {
17     //不要删除，source/sink默认值
18     AUDIOEFFECT_SOURCE_SINK_DEFAULT_INIT,
19
20     //ROBOEFFECT effect SOURCE映射
21     .mic_source = MUSIC_SOURCE_MIC_SOURCE,
22     .app_source = MUSIC_SOURCE_APP_SOURCE,
23     .remind_source = MUSIC_SOURCE_REMIND_SOURCE,
24     .rec_source = MUSIC_SOURCE_REC_SOURCE,
25
26     //ROBOEFFECT effect SINK映射
27     .dac0_sink = MUSIC_SINK_DAC0_SINK,
28     .app_sink = MUSIC_SINK_APP_SINK,
29     .stereo_sink = MUSIC_SINK_STEREO_SINK,
30     .rec_sink = MUSIC_SINK_REC_SINK,
31     .spdif_sink = MUSIC_SINK_SPDIF_SINK,
32 };
33
```

7.3. 增加 source/sink

如果 AudioCore 中有完整的数据输入输出代码，增加 source/sink 的流程与上面删减类似，在上位机新增相关通道，然后需要将 ERROR 修改为正确的通道。

如果是新增一个 AudioCore 中也不存在的通道，则需要先从 AudioCore 开始，

AudioCore 的定制修改与 V3 架构无关，这里不做过多赘述。AudioCore 部分修改完成后，后续操作应当与上述流程一致。

7.4. 绘制框图音效通路

参考前面 4.1 小节音效操作部分，只修改音效通路部分，**不要修改输入或者输出通道**，绘制完成后可直接导入 SDK 编译，无需手动修改任何代码。

7.5. 删减音效控制

在 effect_mode.c 中，有 SDK 默认的一些音效控制，如不需要可直接删掉相应音效地址映射。

```
const uint8_t music_effect_ctrl[AUDIOEFFECT_EFFECT_CONTROL_MAX] =
{
    [EQ_MODE_ADJUST] = MUSIC_eq0_ADDR,
    [MUSIC_VOLUME_ADJUST] = MUSIC_gain_control0_ADDR,
    [MIC_VOLUME_ADJUST] = MUSIC_mic_gain_ADDR,
    [MIC_SILENCE_DETECTOR_PARAM] = MUSIC_silence_detector_mic_ADDR,
    [MUSIC_SILENCE_DETECTOR_PARAM] = MUSIC_silence_detector_music_ADDR,
};
```

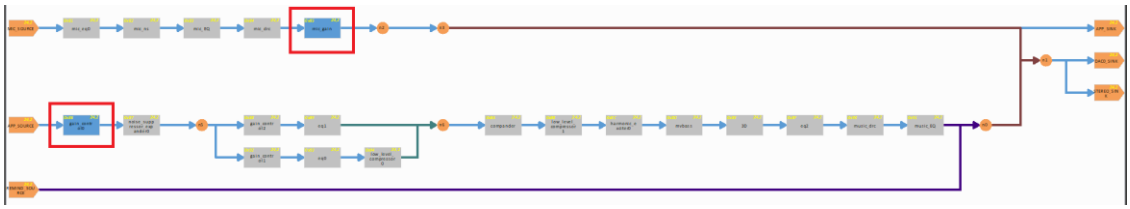
7.6. 增加音效控制

参考第 5 章节修改。

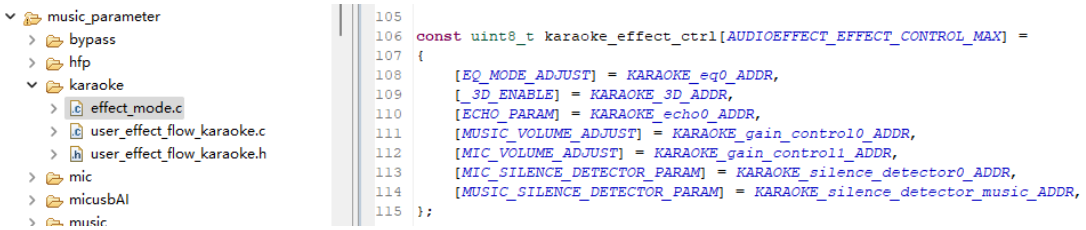
8. 注意事项和常见问题

8.1. 音量控制

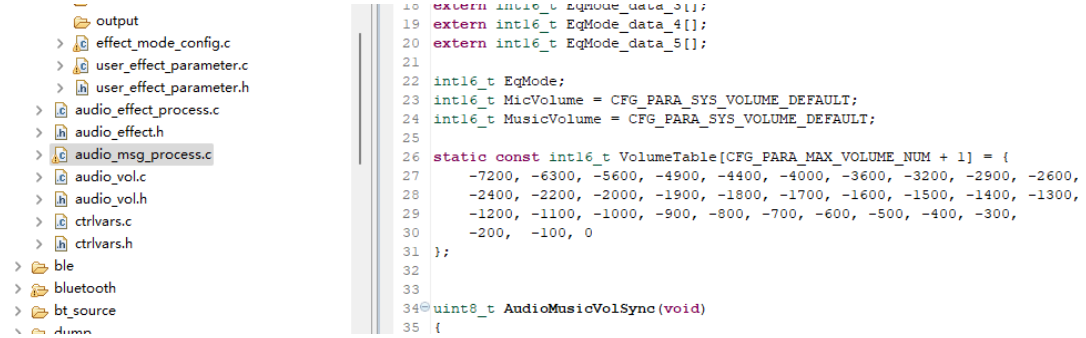
1. 音量控制依赖于音效框图中的 gain control 音效；



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，在 audio_msg_process.c 中可修改如下地方

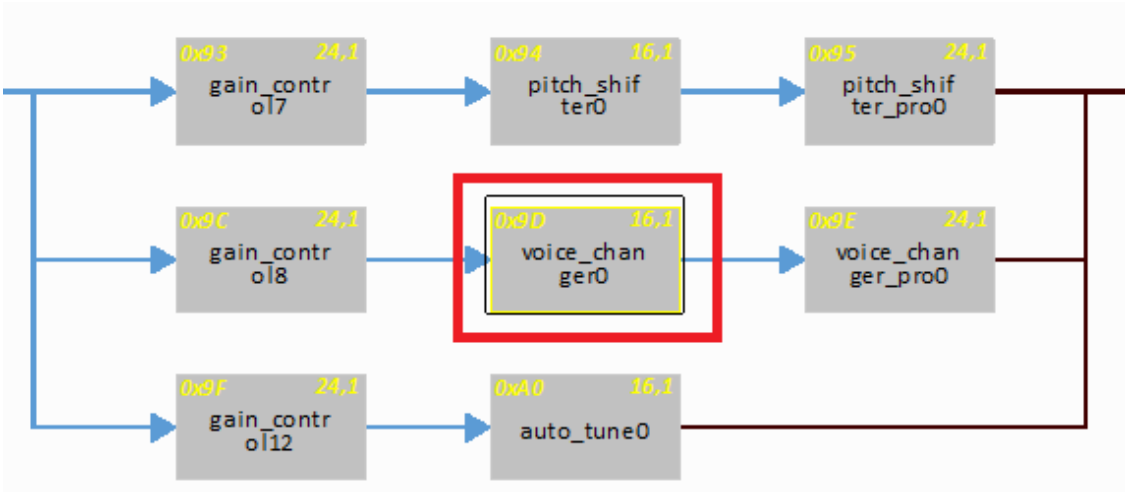


8.2. frame size 和 sample rate 的修改

在修改系统 frame size 时，需要修改 user_effect_flow_xxx.c 中 user_effect_list_xxx 中的对应参数。修改系统 sample rate 同理，需注意部分硬件（如 I2S）可能也会有 sample rate 的参数，对于该部分还需修改 app_config.h 中的对应宏。

8.3. 动态帧长切换

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，SDK 会自动重置音效引擎并切换系统和音效的帧长为 512，再次关闭 voice_changer，系统帧长会切换回默认定义大小。



8.4. 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。

8.5. roboeffect 的内存管理

Roboeffect 的内存主要由**输入输出 buffer + 控制逻辑内存 + 开启音效内存**组成。

- **输入输出 buffer** = (frame size * 4 * bit_width) * (frame buffer num +1);
- **控制逻辑内存** = 340 + 104 * (node + effect num) + sum(all effect params) * 4;
- **开启音效内存** = sum(all effect context size);

注意：只要框图中存在 24bit 音效，bit_width = 2。frame buffer num 由上位机优化给出，根据框图实际 memory 使用情况给出，frame buffer num <= (source + sink num)。

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. 确认不会使用到的音效在框图中全部删除；
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。

8.6. 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*/
```

8.7. 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_mode.c 中需要将其赋为 **AUDIOCORE_SOURCE_SINK_ERROR**，让 SDK 能够正确的去处理。如果 SDK 不使用对应的通道，建议把对应宏也一起关闭。

在第一行默认加上 AUDIOEFFECT_SOURCE_SINK_DEFAULT_INIT 就行，后面在添加自己的配置。

```
const AUDIOEFFECT_SOURCE_SINK_NUM karaoke_mode =
{
    //不要删除，source/sink默认值
    AUDIOEFFECT_SOURCE_SINK_DEFAULT_INIT,

    //ROBOEFFECT effect SOURCE映射
    .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,
    .i2s_mix2_source = KARAOKE_SOURCE_I2S_MIX2_SOURCE,
    .linein_mix_source = KARAOKE_SOURCE_LINEIN_MIX_SOURCE,


    //ROBOEFFECT effect SINK映射
    .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,
};
```

8.8. Roboeffect 错误号

枚举	值	说明	措施
ROBOEFFECT_EFFECT_NOT_EXISTED	-256	音效图中出现了不存在的音效 ID，或者音效代码已经被裁剪	使用调音软件重新导出符合前 Roboeffect 库的音效图及参数
ROBOEFFECT_EFFECT_PARAMS_NOT_FOUND	-255	没有找到相关地址的音效参数	使用调音软件重新导出符合当前 Roboeffect 库的音效图及参数
ROBOEFFECT_INSUFFICIENT_MEMORY	-254	所打开的音效没有足够的内存	重新估算内存使用，增大 Roboeffect 的上下文内存空间
ROBOEFFECT_EFFECT_INIT_FAILED	-253	音效初始化错误	检查音效参数
ROBOEFFECT_ILLEGAL_OPERATION	-252	错误操作，如关闭节点音效，等	避免类似操作
ROBOEFFECT_EFFECT_LIB_NOT_MATCH_1	-251	音效参数和当前的 Roboeffect 库不匹配	使用最新的调音软件导出参数，或者进行参数版本转换
ROBOEFFECT_EFFECT_LIB_NOT_MATCH_2	-250	Roboeffect 库和音效库不匹配	不要单独替换 audio library
ROBOEFFECT_ADDRESS_NOT_EXISTED	-249	roboeffect_estimate_effect_size 返回的错误，地址无法找到	
ROBOEFFECT_PARAMS_ERROR	-248	用户自定义音效相关错误	
ROBOEFFECT_FRAME_SIZE_ERROR	-247	帧长错误	使用音效所匹配的帧长
ROBOEFFECT_MEMORY_SIZE_QUERY_ERROR	-246	内存用量查询错误，可能是错误的输入参数导致	
ROBOEFFECT_EFFECT_VER_NOT_MATCH_ERROR	-245	参数数据中某一个音效的版本号与当前运行的 Roboeffect 库中不匹配	
ROBOEFFECT_LIB_VER_NOT_MATCH_ERROR	-244	参数数据中的 Roboeffect 版本号与当前运行的库不匹配	
ROBOEFFECT_3RD_PARTY_LIB_NOT_MATCH_ERROR	-243	参数数据中的第三方音效版本号与当前运行的库不匹配	