V3 架构应用指导说明

V1.3

版本记录：

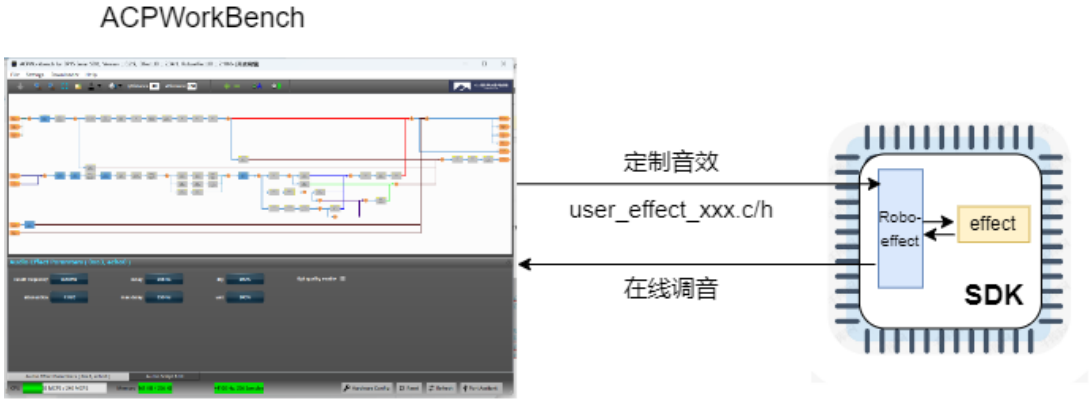
版本号	日期	作者	备注
V1.0	2023-10-18	Yangyu	初版
V1.1	2023-11-21	Yangyu	架构调整更新
V1.2	2023-12-15	Yangyu	格式及内容优化
V1.3	2024-01-29	Yangyu	内容更新

目录

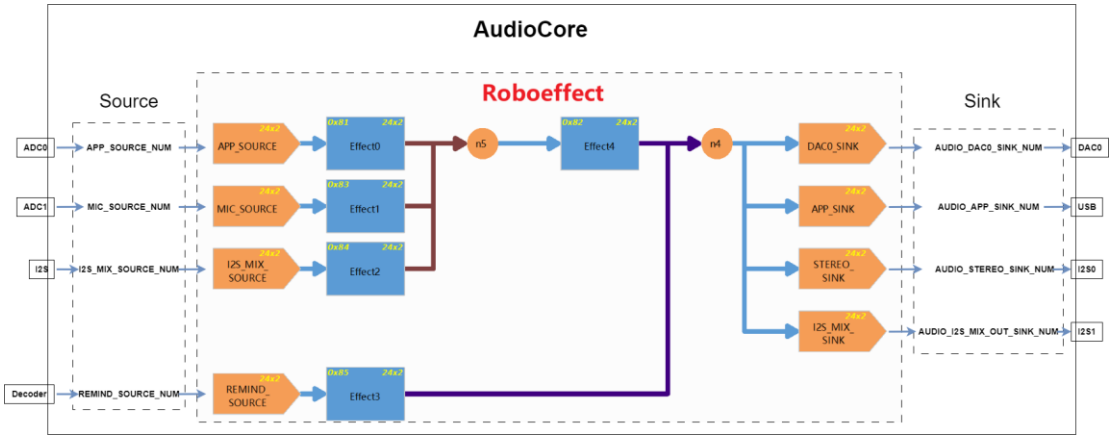
1. Roboeffect 介绍

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。

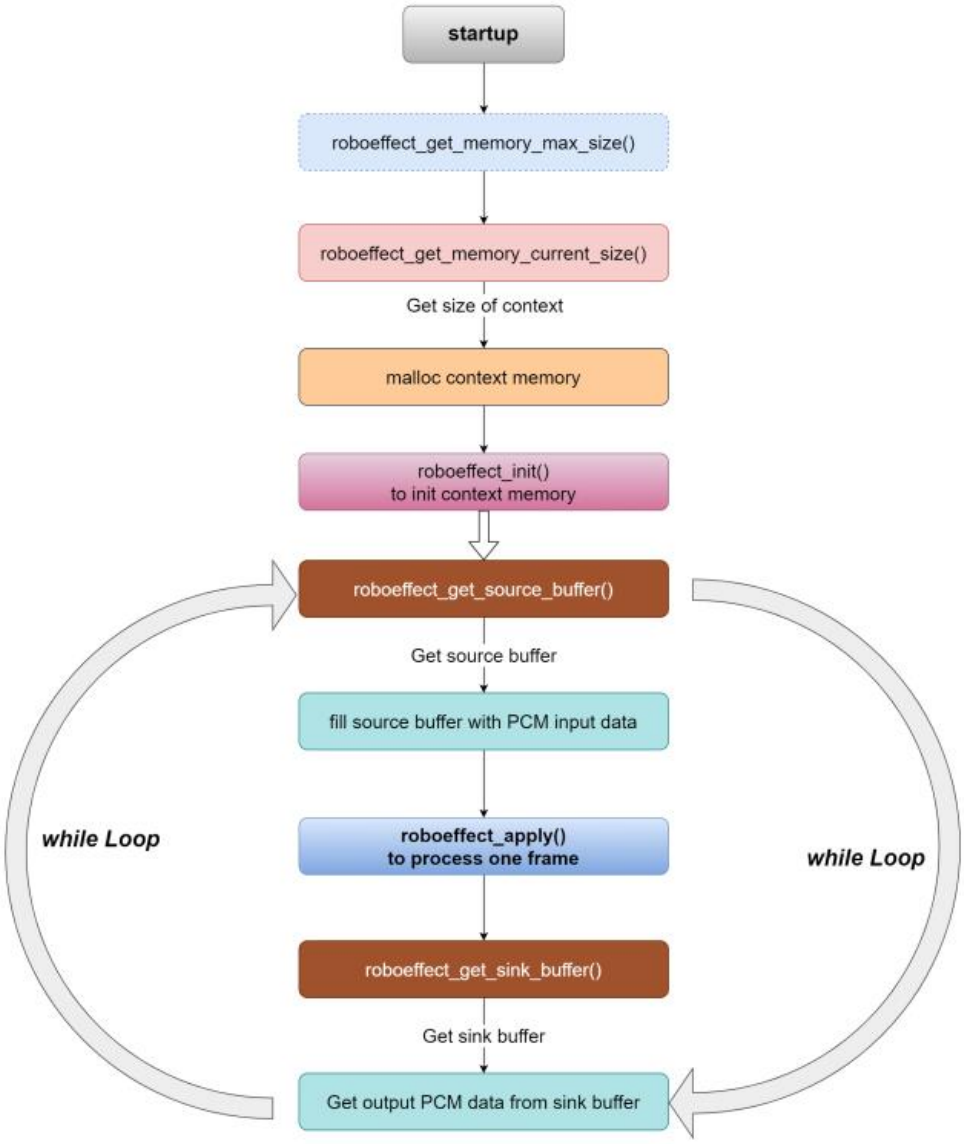


1.1. Roboeffect 与 SDK 的关系



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

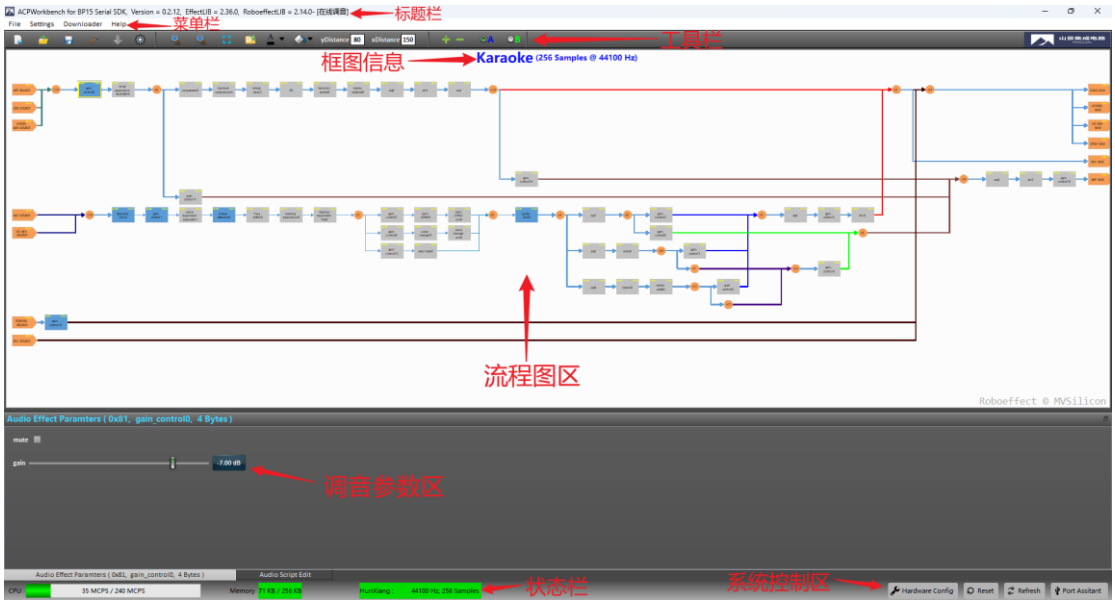
Roboeffect 提供丰富的 API，使外部 SDK 可灵活调用操作整个引擎库。

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_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

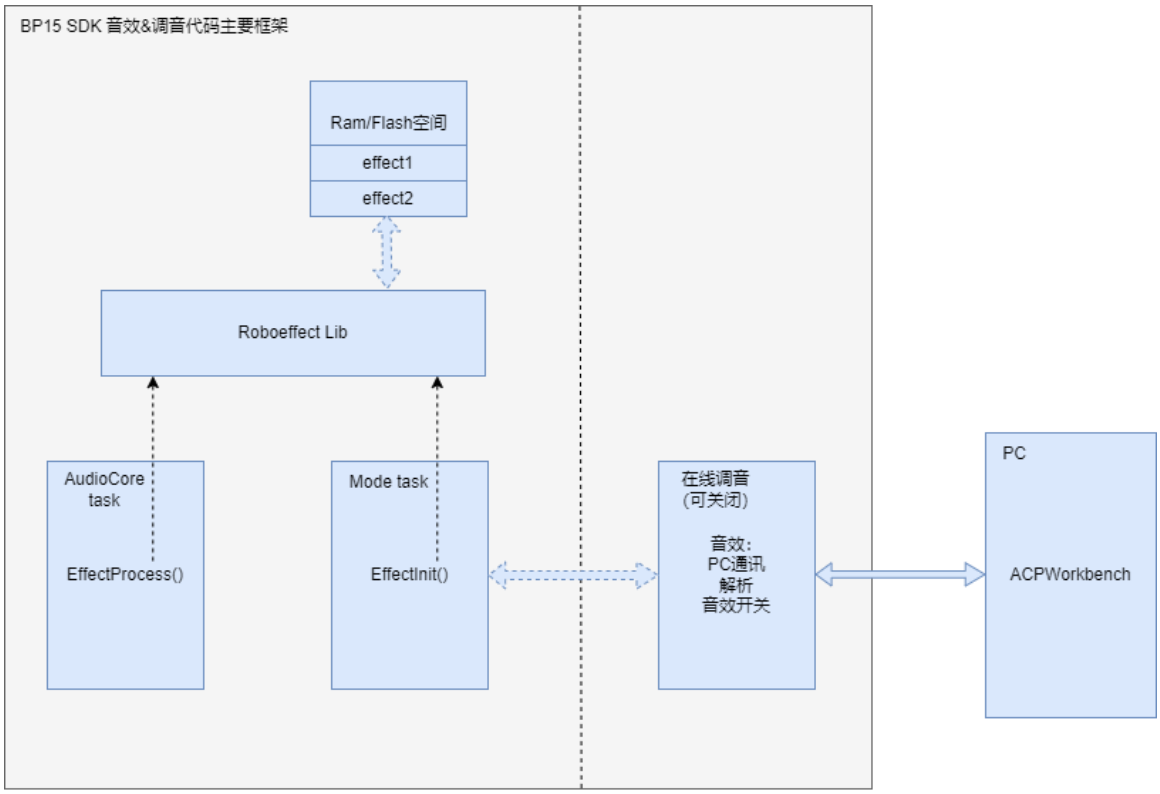
可视化调音工具 ACPWorkbench 是一款可以实时绘制音效流， 实时调音的工具， 相比 ACPWorkbench V2 版本， 该版本从视觉和功能上有了直观的改变。 无论是熟悉山景 SDK 的用户还是刚刚接触的新用户， 都能受益于其直观的操作和快速的音效流定制。需注意 ACPWorkbench V3 版本不兼容 V2 版本。



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。



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。

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_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

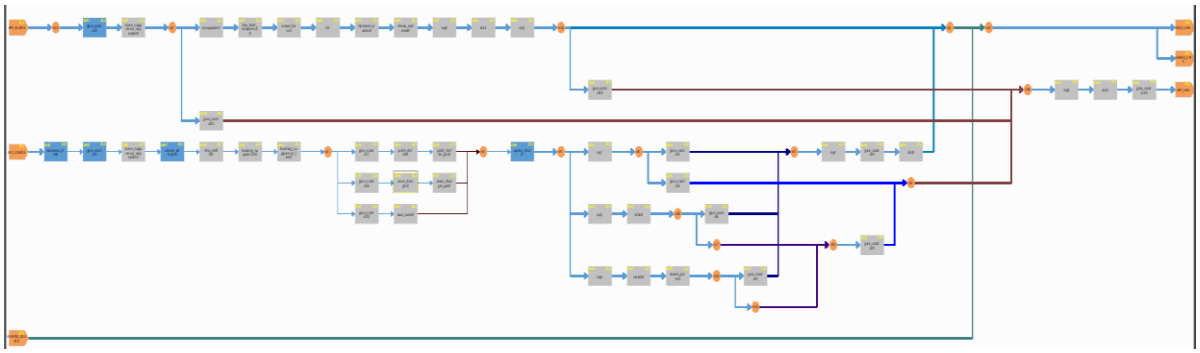
3.3. 以 effect_mode 为核心

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。

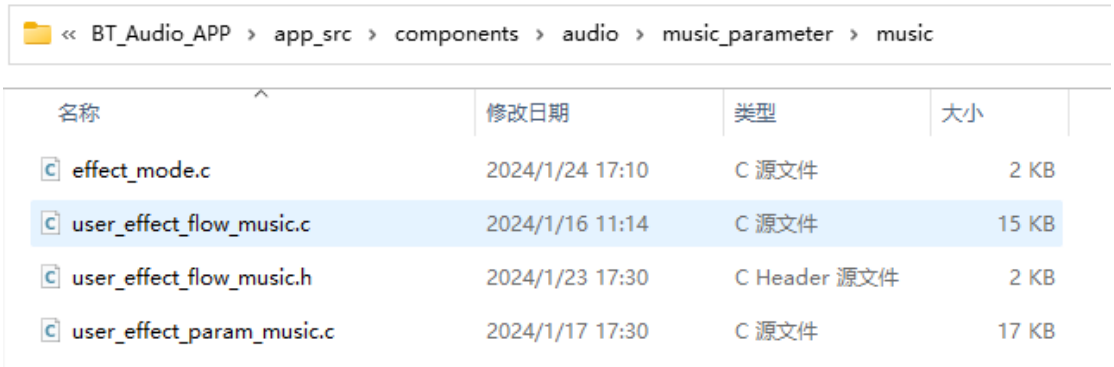
```
typedef enum _EFFECT_MODE
{
    EFFECT_MODE_DEFAULT = 0,
    /**********mic mode***************/
    EFFECT_MODE_MIC,
    /**********music mode*************/
    EFFECT_MODE_MUSIC,
    /**********bypass mode************/
    EFFECT_MODE_BYPASS,
    /**********hfp mode***************/
    EFFECT_MODE_HFP_AEC,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

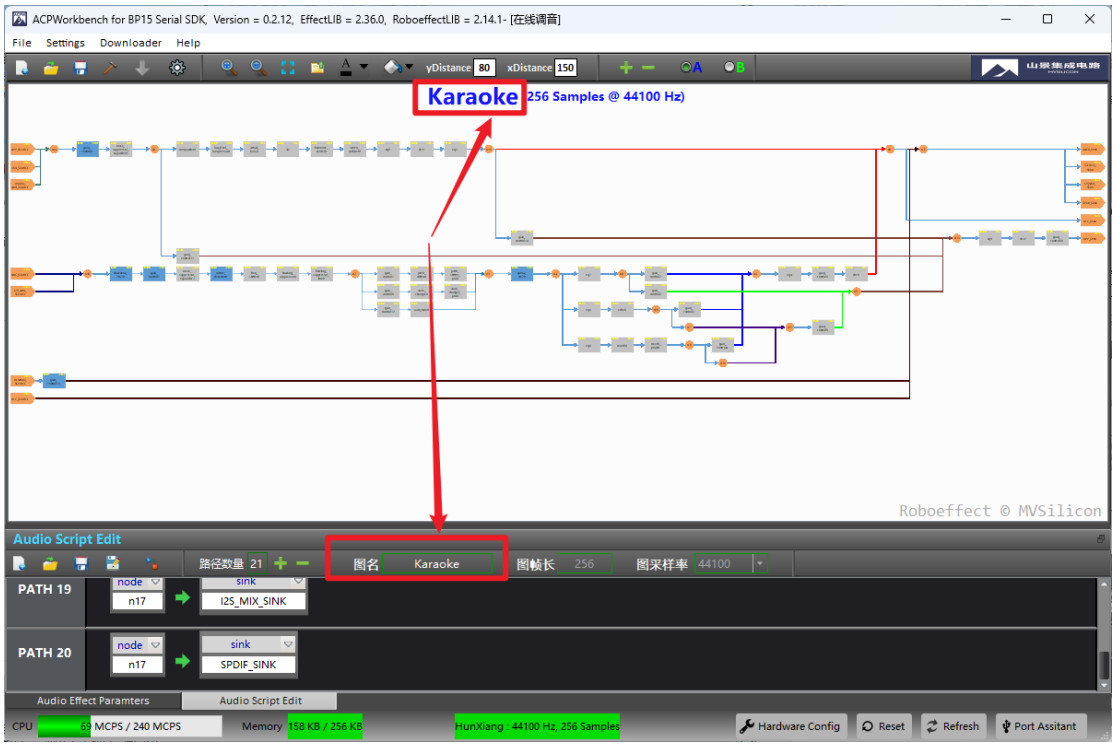


如图，目前上位机导出的音效相关文件全部放在./app_src_components/audio/music_parameter/目录下。

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。

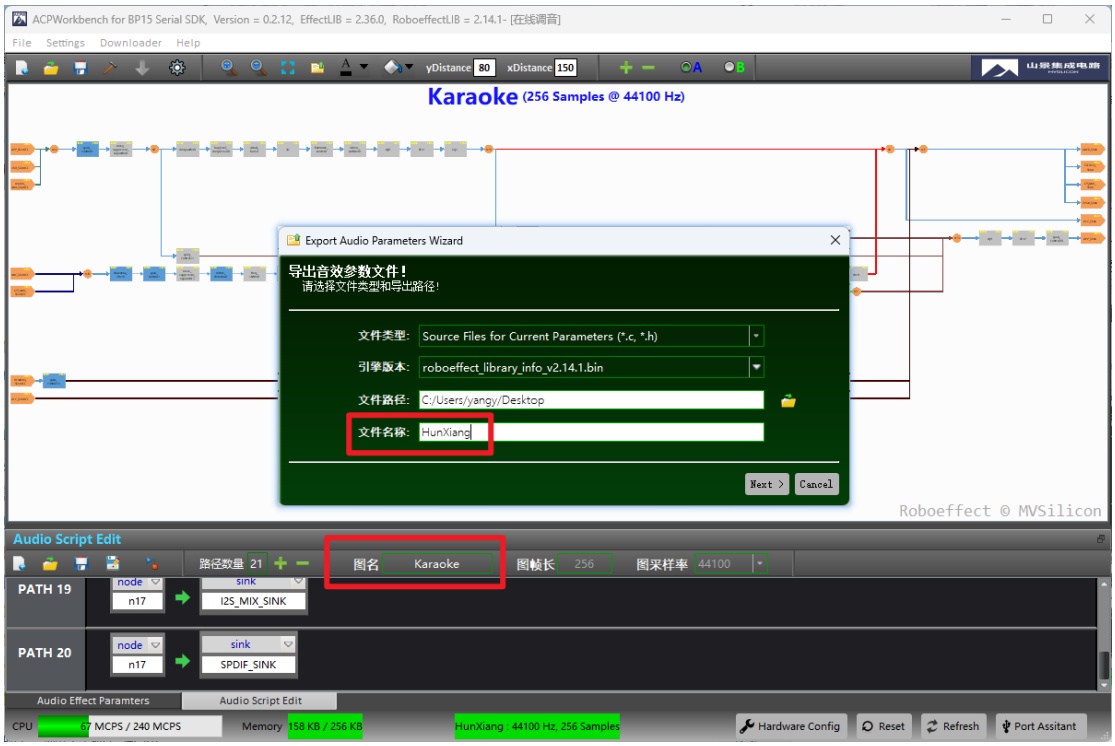
以 karaoke 模式为例，打开 karaoke 模式后连接调音工具，即可调音工具中看到“Karaoke“字样，表示当前框图名是 Karaoke，在导出的 karaoke flow 文件中，所有结构的命名都是以 KARAOKE 为前缀。




```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,
```

3.4.2. 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。




```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*/
```

3.4.3. 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。

```
    //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。

3.5. 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。

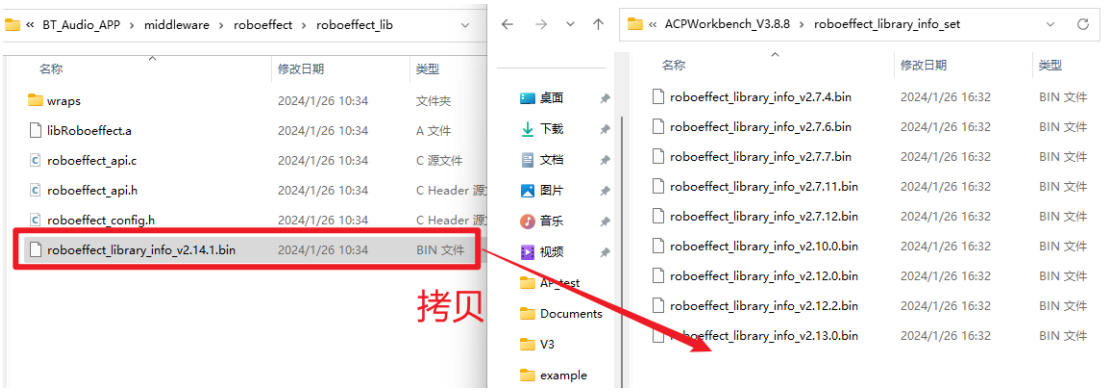
4. 定制音效

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。

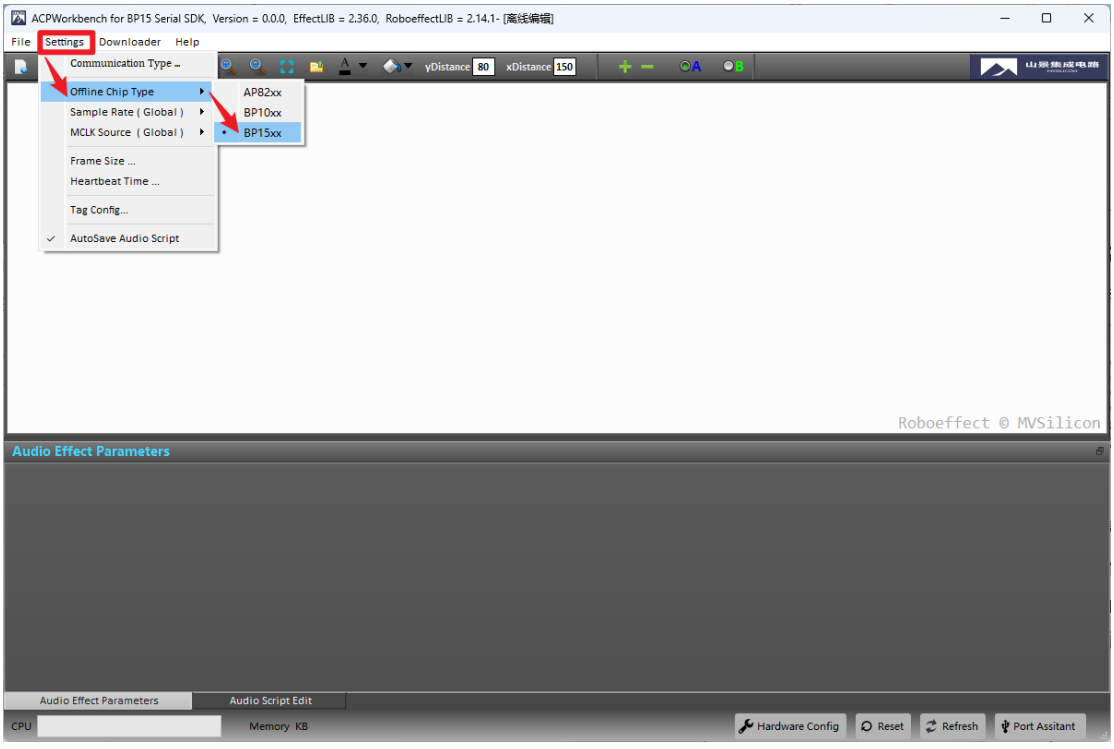
4.1. 上位机绘制框图

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

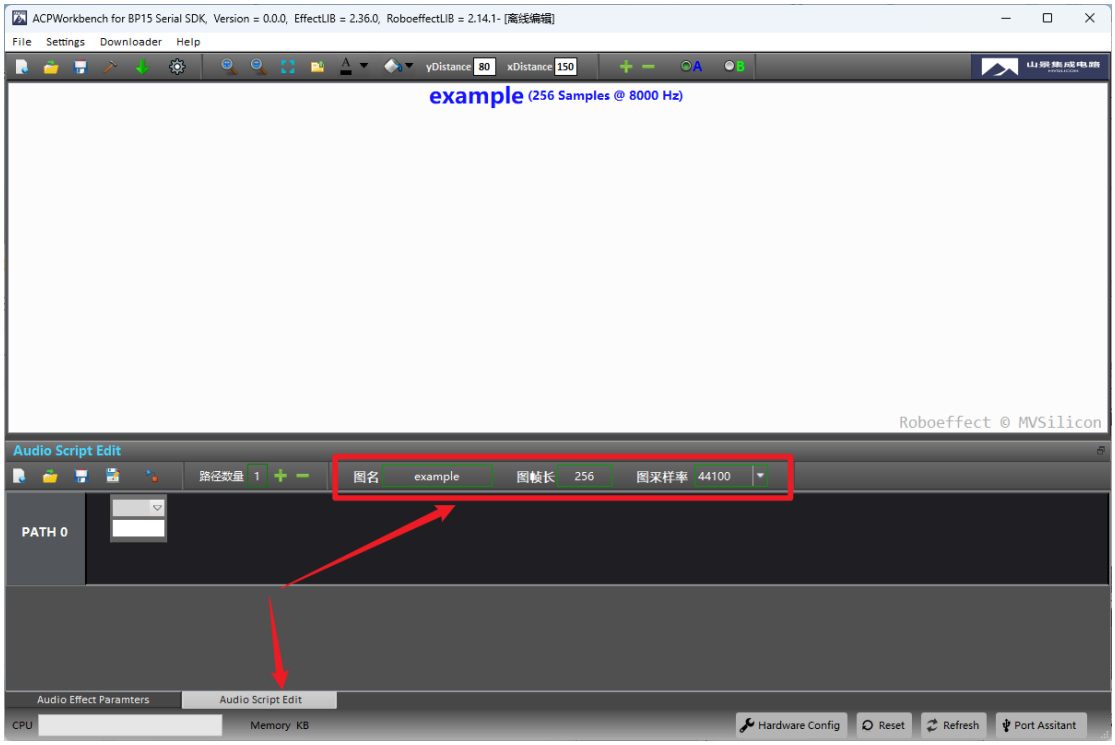


2. 打开调音工具选择正确的芯片型号



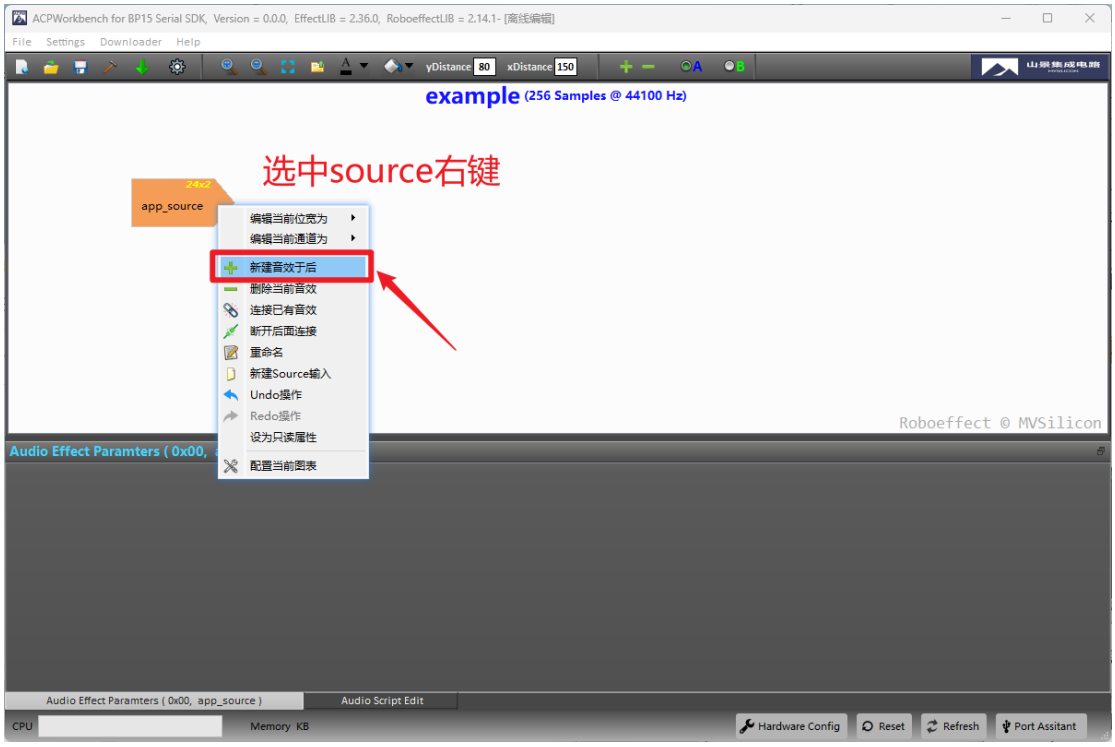
4.1.2. 修改图名称

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。

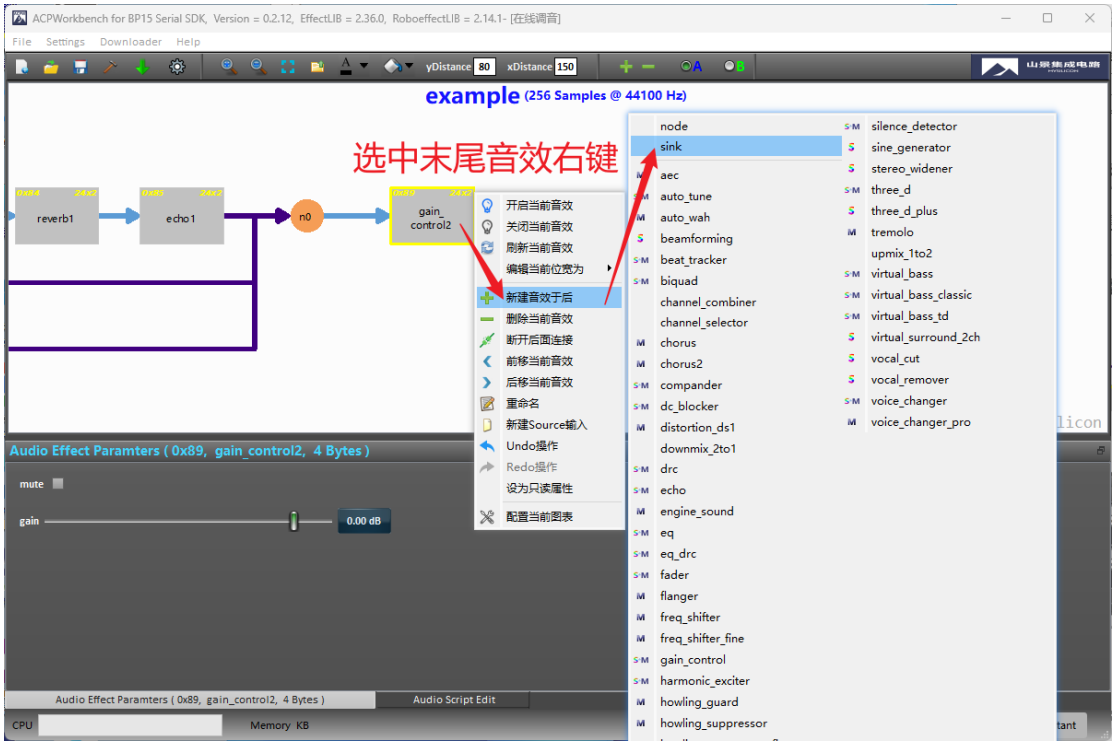


4.1.3. 新建 source 输入

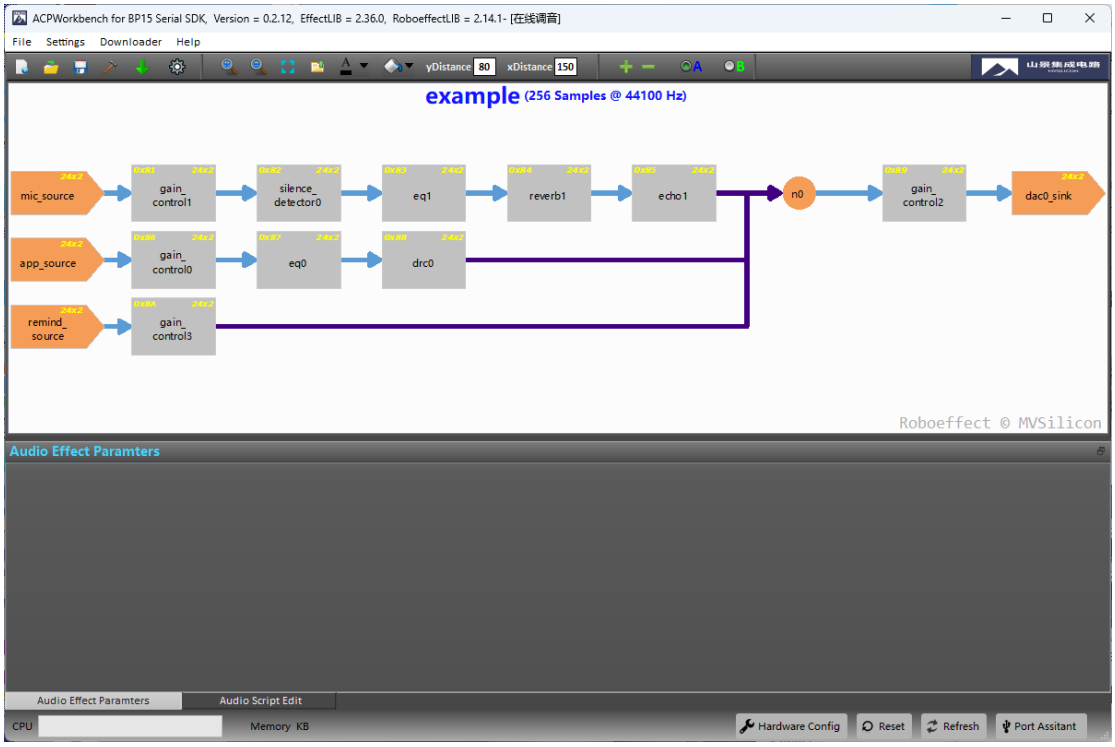
4.1.4. 增加音效



4.1.5. 新建 sink 输出

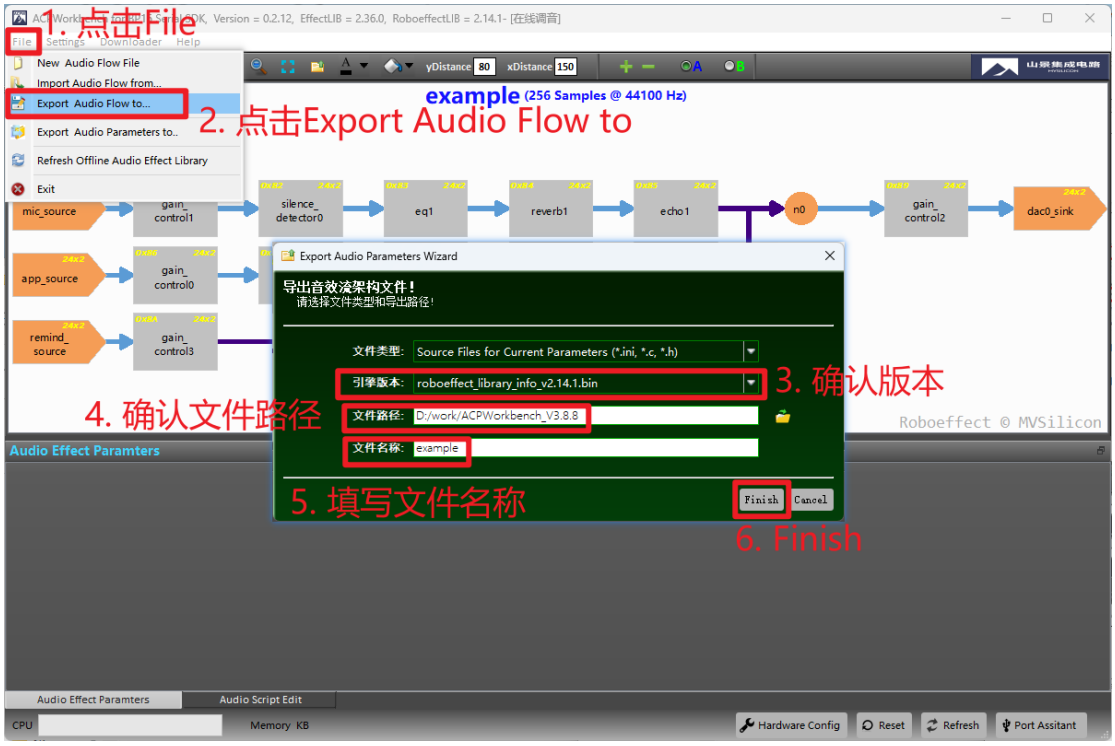


4.2. 导出音效



如图，我们已经完成了一个简单的框图绘制，接下来我们开始将其导出至 SDK。

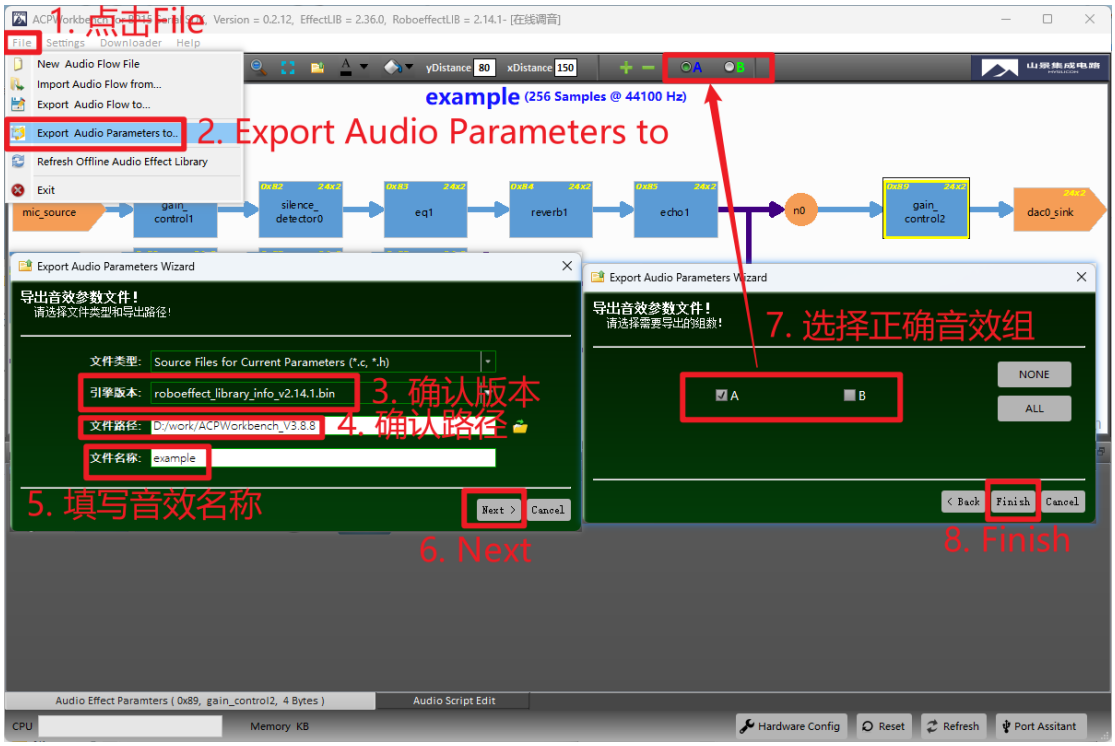
4.2.1. 导出音效 flow



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

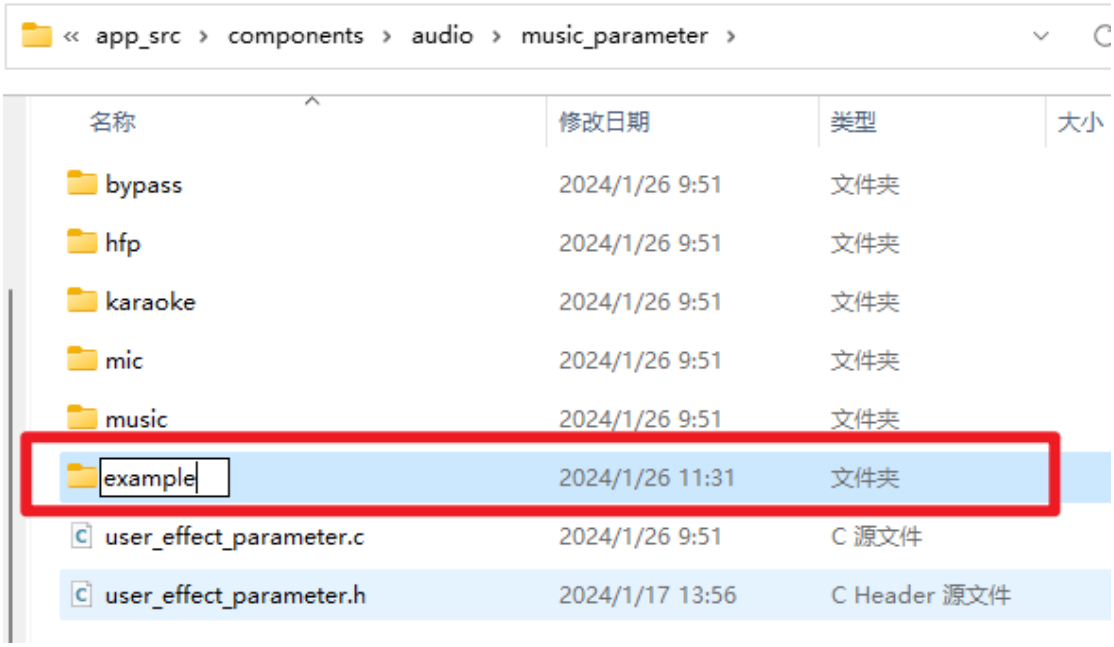
4.2.2. 导出音效参数

简单调整音效参数后，开始导出。

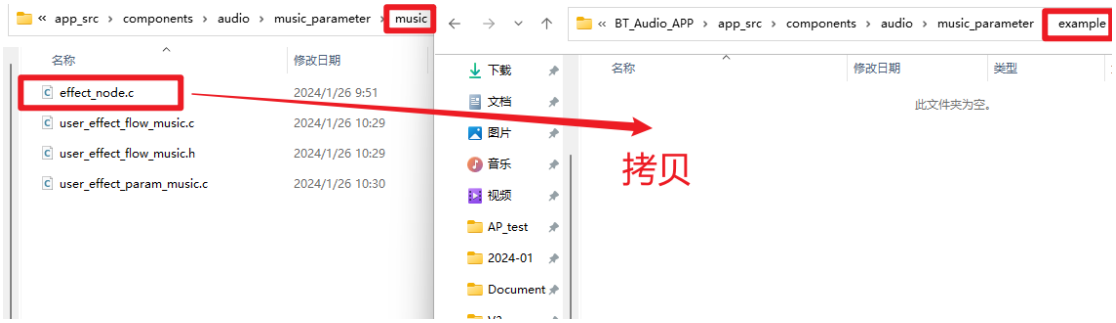


4.3. 新建音效路径及 effect_mode.c

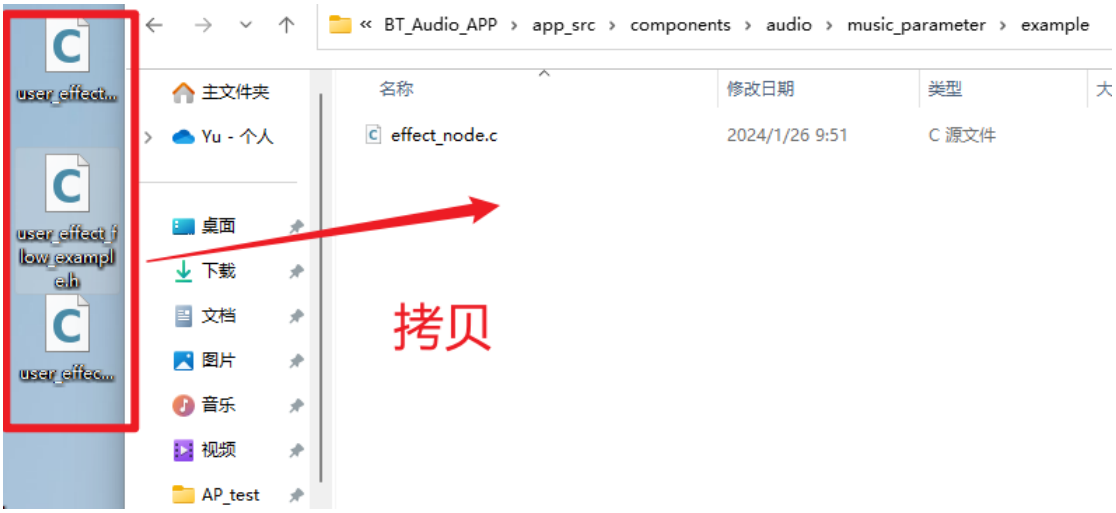
1. 在音效并列目录下新建 example 文件夹；



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 中结构名；

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

extern uint32_t get_user_effects_script_len_example(void);
extern const unsigned char user_effect_parameters_example_example[];
extern const unsigned char user_module_parameters_example_example[];

static const AUDIOEFFECT_EFFECT_PARA 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_example_example,
        .user_module_parameters = (uint8_t *)user_module_parameters_example_example,
        .get_user_effects_script_len = get_user_effects_script_len_example,
    }
};

const AUDIOEFFECT_EFFECT_PARA_TABLE example_mode =
{
    //ROBOEFFECT effect ID 通过这个ID来搜索匹配
    .effect_id    = EFFECT_MODE_EXAMPLE ,
    //该框图下面有1个音效
    .effect_id_count = 1,
```

3. 修改音效地址映射、source 映射和 sink 映射；

```
const AUDIOEFFECT_EFFECT_PARA_TABLE example_mode =
{
    //ROBOEFFECT effect ID 通过这个ID来搜索匹配
    .effect_id    = EFFECT_MODE_EXAMPLE ,
    //该框图下面有1个音效
    .effect_id_count = 1,

    //ROBOEFFECT effect 音效地址映射
    .effect_addr =
    {
        .APP_SOURCE_GAIN_ADDR = EXAMPLE_gain_control0_ADDR,
        .MIC_SOURCE_GAIN_ADDR = EXAMPLE_gain_control1_ADDR,
        .REMIND_SOURCE_GAIN_ADDR = EXAMPLE_gain_control3_ADDR,
    },

    //ROBOEFFECT effect SOURCE映射
    .audioeffect_source =
    {
        .mic_source = EXAMPLE_SOURCE_mic_source,
        .app_source = EXAMPLE_SOURCE_app_source,
        .remind_source = EXAMPLE_SOURCE_remind_source,
        .rec_source = AUDIOCORE_SOURCE_SINK_ERROR,
        .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 = EXAMPLE_SINK_dac0_sink,
        .app_sink = AUDIOCORE_SOURCE_SINK_ERROR,
        .stereo_sink = AUDIOCORE_SOURCE_SINK_ERROR,
        .rec_sink = AUDIOCORE_SOURCE_SINK_ERROR,
        .i2s_mix_sink = AUDIOCORE_SOURCE_SINK_ERROR,
        .spdif_sink = AUDIOCORE_SOURCE_SINK_ERROR,
    },
```

注意：框图中没有的source或sink需要置为ERROR状态

4. effect_para_table 增加新音效；

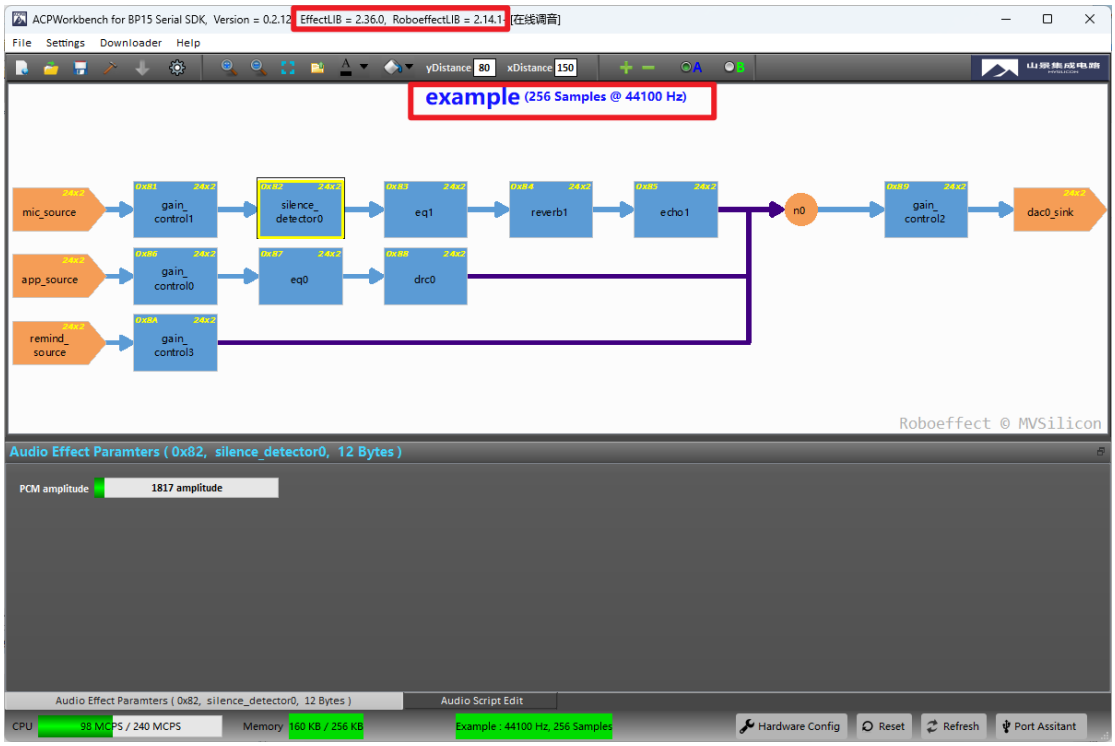
```
#ifdef CFG_FUNC_EFFECT_BYPASS_EN
extern const AUDIOEFFECT_EFFECT_PARA_TABLE bypass_mode;
#else
#ifdef CFG_FUNC_MIC_KARAOKE_EN
extern const AUDIOEFFECT_EFFECT_PARA_TABLE karaoke_mode;
#else
extern const AUDIOEFFECT_EFFECT_PARA_TABLE mic_mode;
extern const AUDIOEFFECT_EFFECT_PARA_TABLE music_mode;
extern const AUDIOEFFECT_EFFECT_PARA_TABLE example_mode;
#endif
#endif
#if defined(CFG_APP_BT_MODE_EN) && (BT_HFP_SUPPORT == ENABLE)
extern const AUDIOEFFECT_EFFECT_PARA_TABLE hfp_mode;
#endif

static const AUDIOEFFECT_EFFECT_PARA_TABLE *effect_para_table[] =
{
#ifdef CFG_FUNC_EFFECT_BYPASS_EN
    &bypass_mode,
#else
#ifdef CFG_FUNC_MIC_KARAOKE_EN
    &karaoke_mode,
#else
    &mic_mode,
    &music_mode,
    &example_mode,
#endif
#endif
#if defined(CFG_APP_BT_MODE_EN) && (BT_HFP_SUPPORT == ENABLE)
    &hfp_mode,
#endif
    NULL,
};
```

5. 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

音效模式由断电记忆，因此请烧录时请选择全擦除。



5. SDK 音效控制

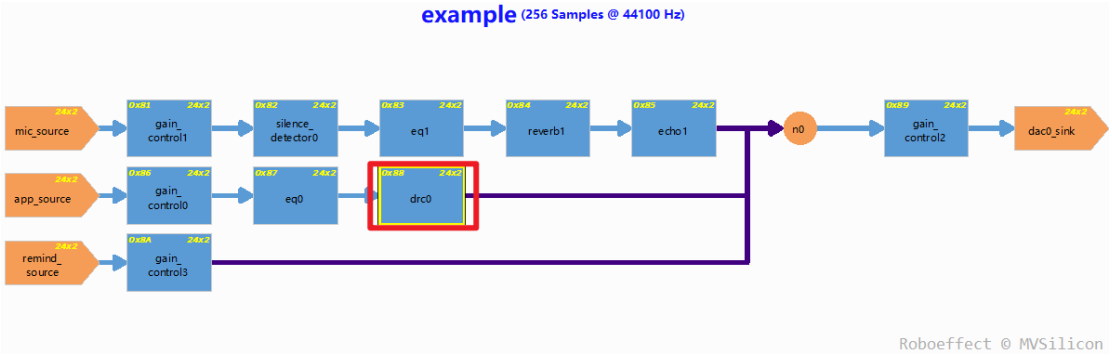
上一章节，我们演示了如何从 0 开始，新增一个音效模式导入到 SDK，接下来，我们来演示如何控制我们新增的音效。

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_NUM 增加枚举 DRC；

```
/**
 * @brief Audio effect num(SDK defined)
 */
typedef enum _AUDIOEFFECT_EFFECT_NUM
{
    MUSIC_EQ = 0,
    MIC_EQ,
    REVERB,
    REVERBPLATE,
    ECHO,
    VOCAL_CUT,
    SILENCE_DETECTOR,
    VOICE_CHANGER,
    APP_SOURCE_GAIN,
    I2S_MIX_SOURCE_GAIN,
    USB_SOURCE_GAIN,
    LINEIN_MIX_SOURCE_GAIN,
    REMIND_SOURCE_GAIN,
    MIC_SOURCE_GAIN,
    REC_SOURCE_GAIN,
    DAC0_SINK_GAIN,
    APP_SINK_GAIN,
    STEREO_SINK_GAIN,
    I2S_MIX_SINK_GAIN,
    REC_SINK_GAIN,
    DRC,
} AUDIOEFFECT_EFFECT_NUM;
```

2. AUDIOEFFECT_EFFECT_ADDR 新增 DRC_ADDR；

```
/**
 * @brief Audio effect addr
 */
typedef struct _AUDIOEFFECT_EFFECT_ADDR
{
    uint8_t MUSIC_EQ_ADDR;
    uint8_t MIC_EQ_ADDR;
    uint8_t REVERB_ADDR;
    uint8_t REVERBPLATE_ADDR;
    uint8_t ECHO_ADDR;
    uint8_t VOCAL_CUT_ADDR;
    uint8_t SILENCE_DETECTOR_ADDR;
    uint8_t VOICE_CHANGER_ADDR;
    uint8_t APP_SOURCE_GAIN_ADDR;
    uint8_t REMIND_SOURCE_GAIN_ADDR;
    uint8_t MIC_SOURCE_GAIN_ADDR;
    uint8_t REC_SOURCE_GAIN_ADDR;
    uint8_t DAC0_SINK_GAIN_ADDR;
    uint8_t APP_SINK_GAIN_ADDR;
    uint8_t STEREO_SINK_GAIN_ADDR;
    uint8_t REC_SINK_GAIN_ADDR;
    uint8_t DRC_ADDR;
} AUDIOEFFECT_EFFECT_ADDR;
```

3. get_audioeffect_addr()函数新增 DRC 相关功能代码；

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

    switch(effect_name)
    {
        case DRC:
            addr = param->effect_addr.DRC_ADDR;
            break;
        case MUSIC_EQ:
            addr = param->effect_addr.MUSIC_EQ_ADDR;
            break;
```

4. 音效地址映射 DRC_ADDR 指向框图中实际音效；

```
//ROBOEFFECT effect 音效地址映射
.effect_addr =
{
    .APP_SOURCE_GAIN_ADDR = EXAMPLE_gain_control0_ADDR,
    .MIC_SOURCE_GAIN_ADDR = EXAMPLE_gain_control1_ADDR,
    .REMIND_SOURCE_GAIN_ADDR = EXAMPLE_gain_control3_ADDR,
    .DRC_ADDR = EXAMPLE_drc0_ADDR,
},
```

5. 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,
```

6. 新增消息处理逻辑；

```
    case MSG_VB:
        APP_DBG("MSG_VB\n");
        #if CFG_AUDIO_EFFECT_MUSIC_VIRTUAL_BASS_EN
        gCtrlVars.music_vb_unit.vb_en = !gCtrlVars.music_vb_unit.vb_en;
        #endif
        #ifdef CFG_FUNC_DISPLAY_EN
        msgSend.msgId = MSG_DISPLAY_SERVICE_VB;
        MessageSend(GetSysModeMsgHandle(), &msgSend);
        #endif
        break;
    case MSG_DRC_ONOFF:
        APP_DBG("MSG_DRC_ONOFF\n");
        AudioEffect_effect_enable(DRC, !AudioEffect_effect_status_Get(DRC));
        break;
    case MSG_EFFECTMODE:
def CFG_FUNC_EFFECT_BYPASS_EN
        if(GetSystemMode() == ModeBtHfPlay  //蓝牙通话模式下，不支持音效模式切换
f CFG_FUNC_RECORDER_EN
        || SoftFlagGet(SoftFlagRecording)    //录音中，不能切换音效模式
```

7. 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,
```

8. 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。

```
//ROBOEFFECT effect 音效地址映射
.effect_addr =
{
    .APP_SOURCE_GAIN_ADDR = EXAMPLE_gain_control0_ADDR,
    .MIC_SOURCE_GAIN_ADDR = EXAMPLE_gain_control1_ADDR,
    .REMIND_SOURCE_GAIN_ADDR = EXAMPLE_gain_control3_ADDR,
    .DRC_ADDR = EXAMPLE_drc0_ADDR,
},
```

5.4. EQ 控制

EQ 高低音调节作为 Karaoke 的基本功能之一，我们以 Music EQ 为例，演示如何在新增音效下实现高低音调节。

1. 开启宏 CFG_FUNC_MUSIC_TREB_BASS_EN；

```
#define CFG_FUNC_MUSIC_TREB_BASS_EN     //Music高低音调节功能配置
```

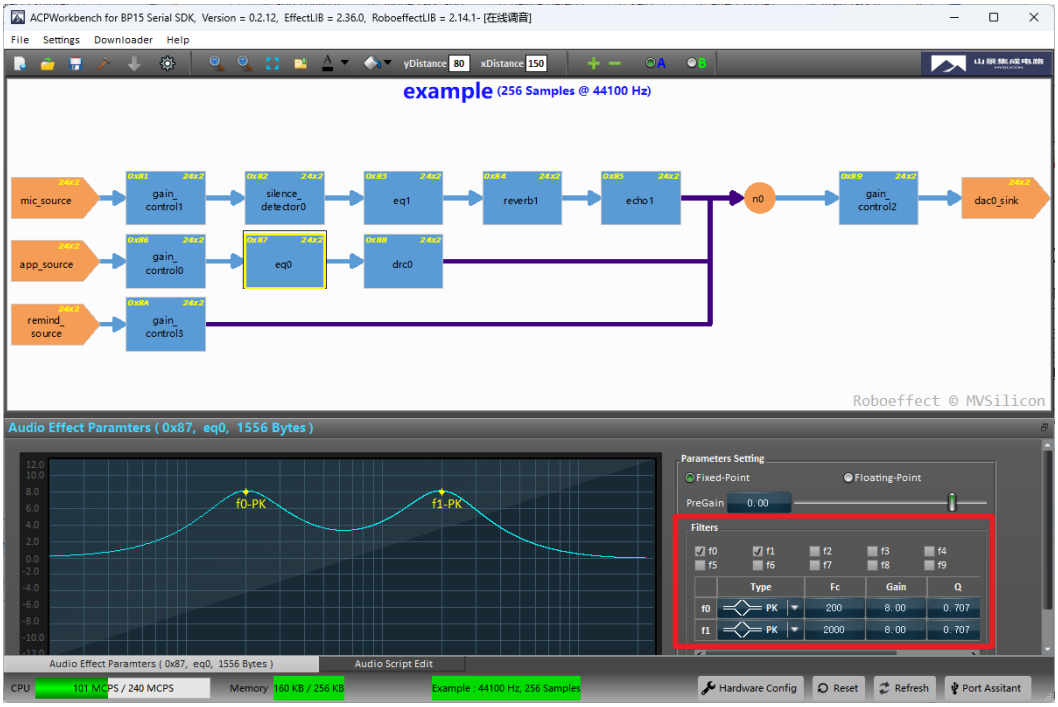
2. 音效地址映射匹配要调节的 EQ；

```
//ROBOEFFECT effect 音效地址映射
.effect_addr =
{
    .APP_SOURCE_GAIN_ADDR = EXAMPLE_gain_control0_ADDR,
    .MIC_SOURCE_GAIN_ADDR = EXAMPLE_gain_control1_ADDR,
    .REMIND_SOURCE_GAIN_ADDR = EXAMPLE_gain_control3_ADDR,
    .DRC_ADDR = EXAMPLE_drc0_ADDR,
    .MUSIC_EQ_ADDR = EXAMPLE_eq0_ADDR,
},
```

3. 按键测试如下四个按键消息；

```
//adc2 key
{MSG_NONE,      MSG_BT_HF_VOICE_RECOGNITION,    MSG_BT_HF_CALL_REJECT,      M
{MSG_NONE,      MSG_FOLDER_PRE,                 MSG_BROWSE,                 M
{MSG_NONE,      MSG_FOLDER_NEXT,                MSG_REC_FILE_DEL,           M
{MSG_NONE,      MSG_RTC_SET_TIME,               MSG_RTC_DISP_TIME,          M
{MSG_NONE,      MSG_RTC_SET_ALARM,              MSG_MIC_FIRST,              M
{MSG_NONE,      MSG_RTC_UP,                     MSG_NONE,                   M
{MSG_NONE,      MSG_RTC_DOWN,                   MSG_BT_HF_TRANS_CHANGED,    M
{MSG_NONE,      MSG_BT_CONNECT_CTRL,            MSG_BT_ENTER_DUT_MODE,      M
{MSG_NONE,      MSG_BT_CONNECT_MODE,            MSG_UPDATE,                 M
{MSG_NONE,      MSG_MUSIC_TREB_UP,              MSG_MUSIC_TREB_DW,          M
{MSG_NONE,      MSG_MUSIC_BASS_UP,              MSG_MUSIC_BASS_DW,          M
```

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。

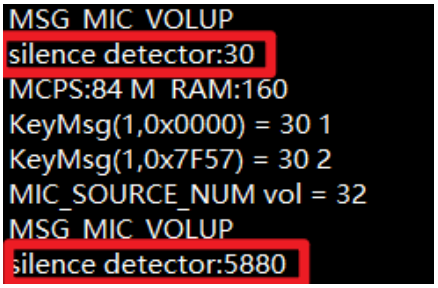
1. 匹配音效地址；

```
//ROBOEFFECT effect 音效地址映射
.effect_addr =
{
    .APP_SOURCE_GAIN_ADDR = EXAMPLE_gain_control0_ADDR,
    .MIC_SOURCE_GAIN_ADDR = EXAMPLE_gain_control1_ADDR,
    .REMIND_SOURCE_GAIN_ADDR = EXAMPLE_gain_control3_ADDR,
    .DRC_ADDR = EXAMPLE_drc0_ADDR,
    .MUSIC_EQ_ADDR = EXAMPLE_eq0_ADDR,
    .SILENCE_DETECTOR_ADDR = EXAMPLE_silence_detector0_ADDR,
},
```

2. 在按键消息 MSG_MIC_VOLUP 处添加打印；

```
case MSG_MIC_VOLUP:
    AudioMicVolUp();
    APP_DBG("MSG_MIC_VOLUP\n");
    APP_DBG("silence detector:%d\n", AudioEffect_SilenceDetector_Get(SILENCE_DETECTOR));
    msgSend.msgId = MSG_DISPLAY_SERVICE_MIC_VOL;
```

3. 测试打印。

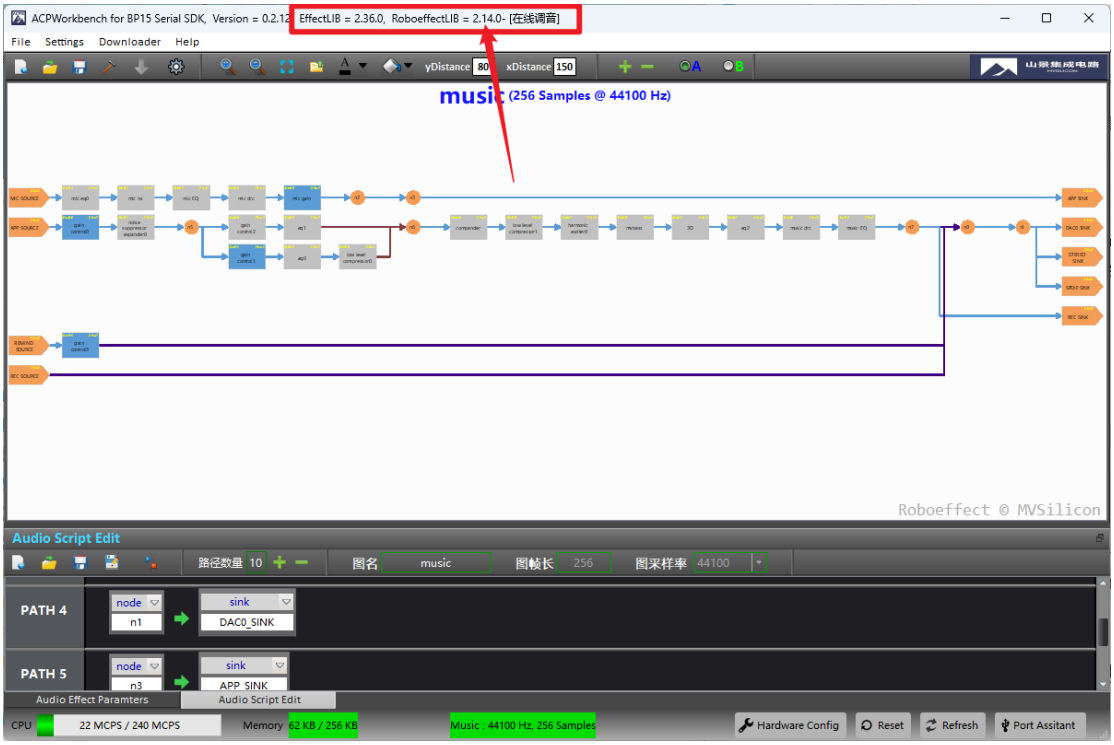


6. 音效库与 roboeffect 库的升级

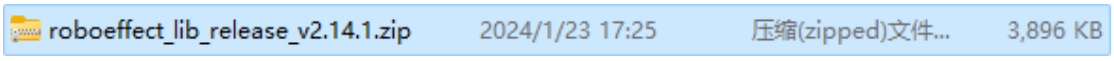
这一章节我们演示基于 SDK v0.2.12 版本从 roboeffect 库 v2.14.0 升级至 v2.14.1 的详细过程。V3 架构音效库与 roboeffect 库互相深度绑定，两者需要同步升级。

6.1. 环境准备

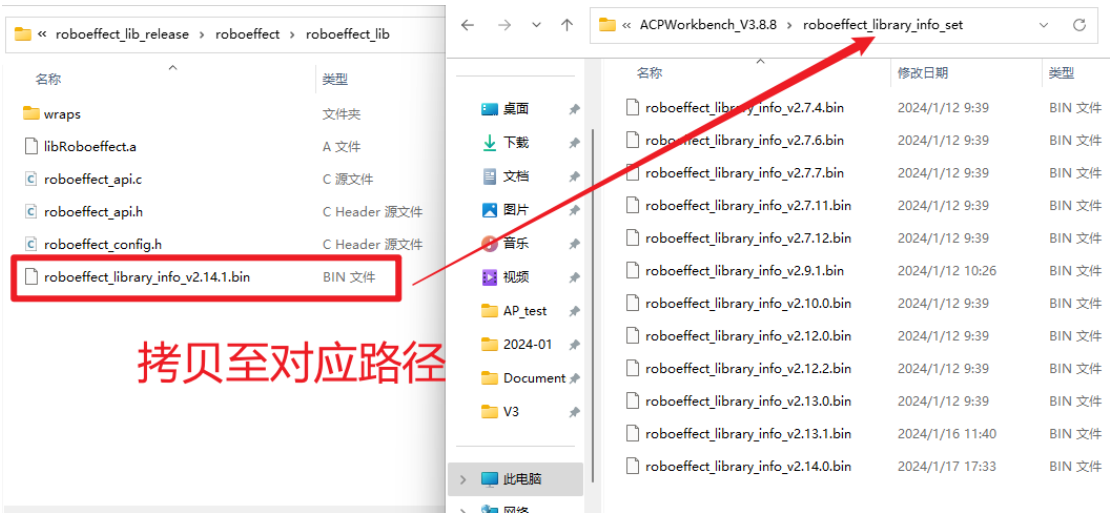
1. 一套完整的运行环境：SDK+开发板+ACPWorkBench（通常情况下，ACPWorkBench 使用手上最新版本即可）



2. 目标版本 roboeffect_lib_release 库压缩包



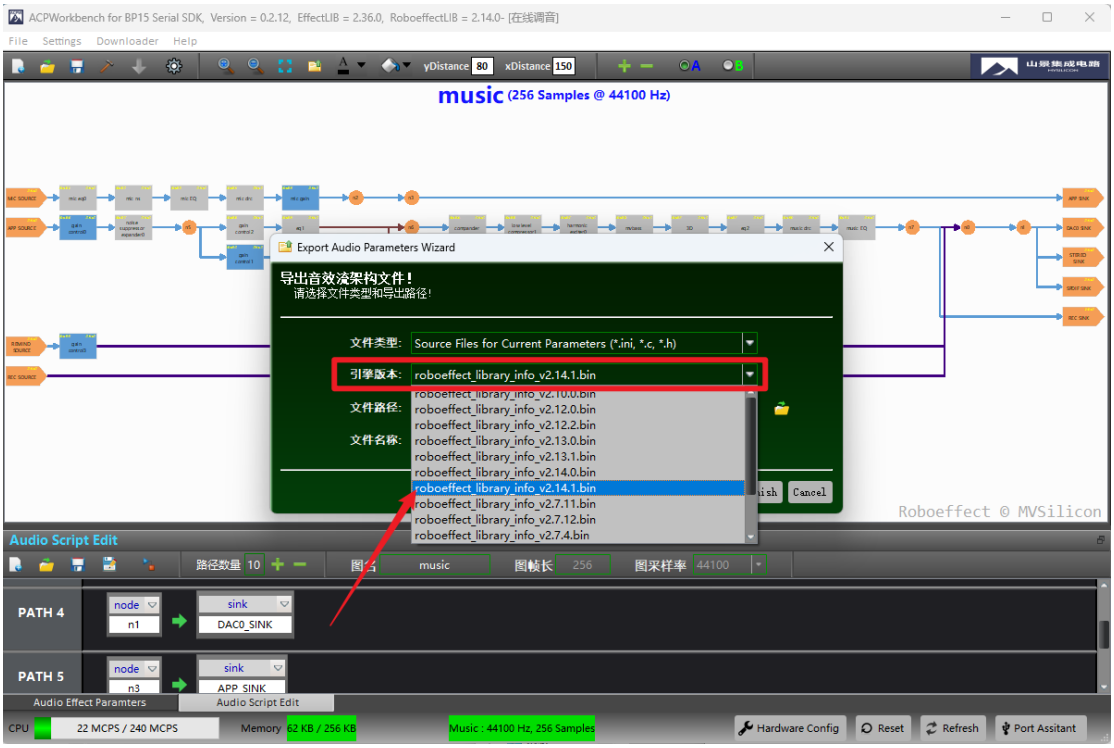
6.2. roboeffect_library_info_vX.X.X.bin



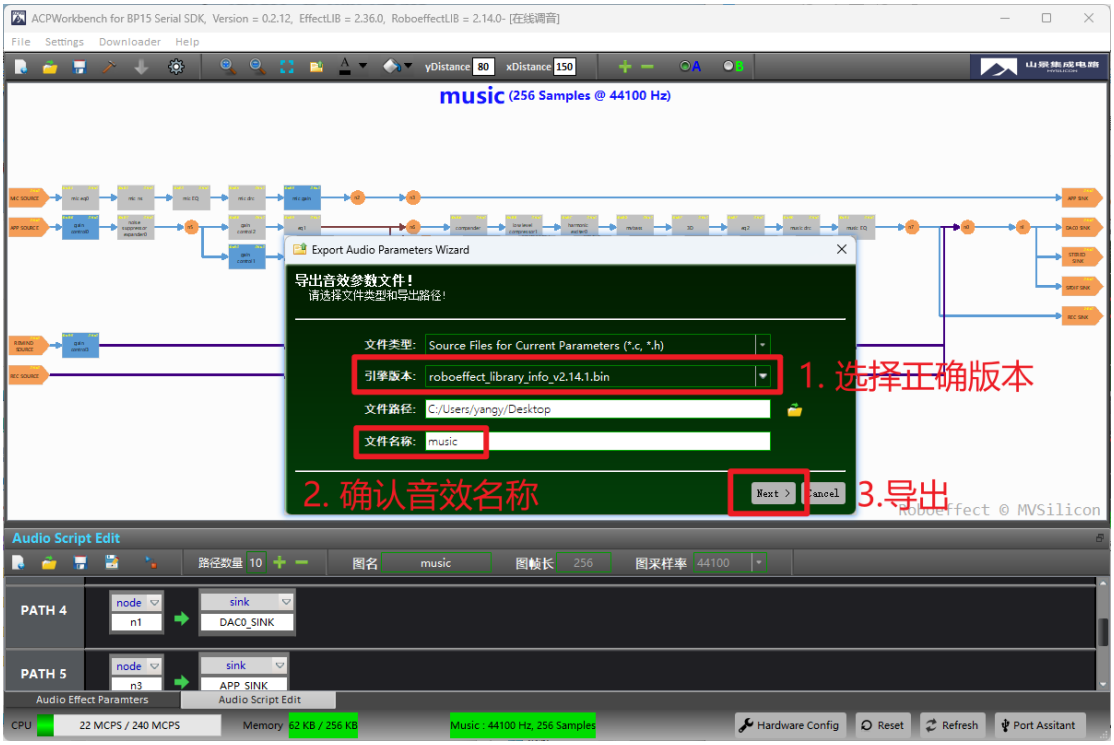
roboeffect_library_info_vX.X.X.bin 中包含了当前版本音效库的音效参数信息，在开始更新之前我们首先需要将其拷贝至调音工具的对应路径下。

6.3. 更新音效文件

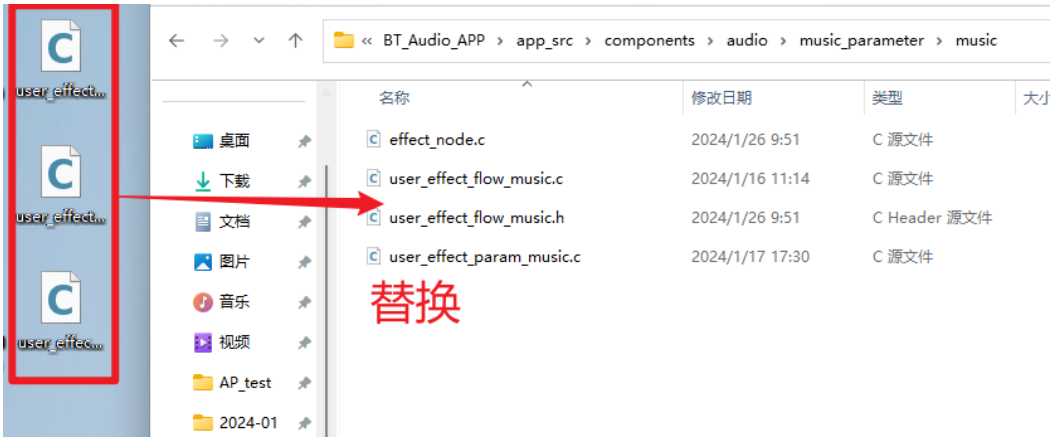
1. 连接上位机
2. 基于目标版本导出音效 flow 文件



3. 基于目标版本导出音效参数文件

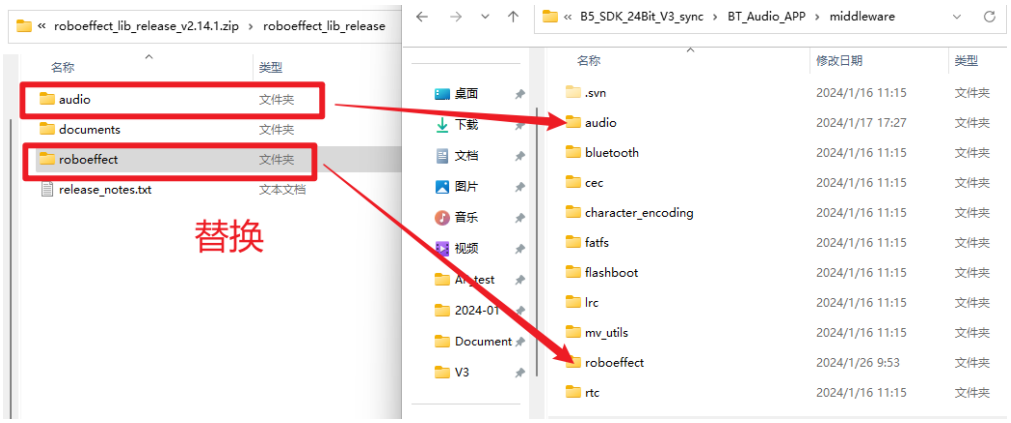


4. 替换音效文件

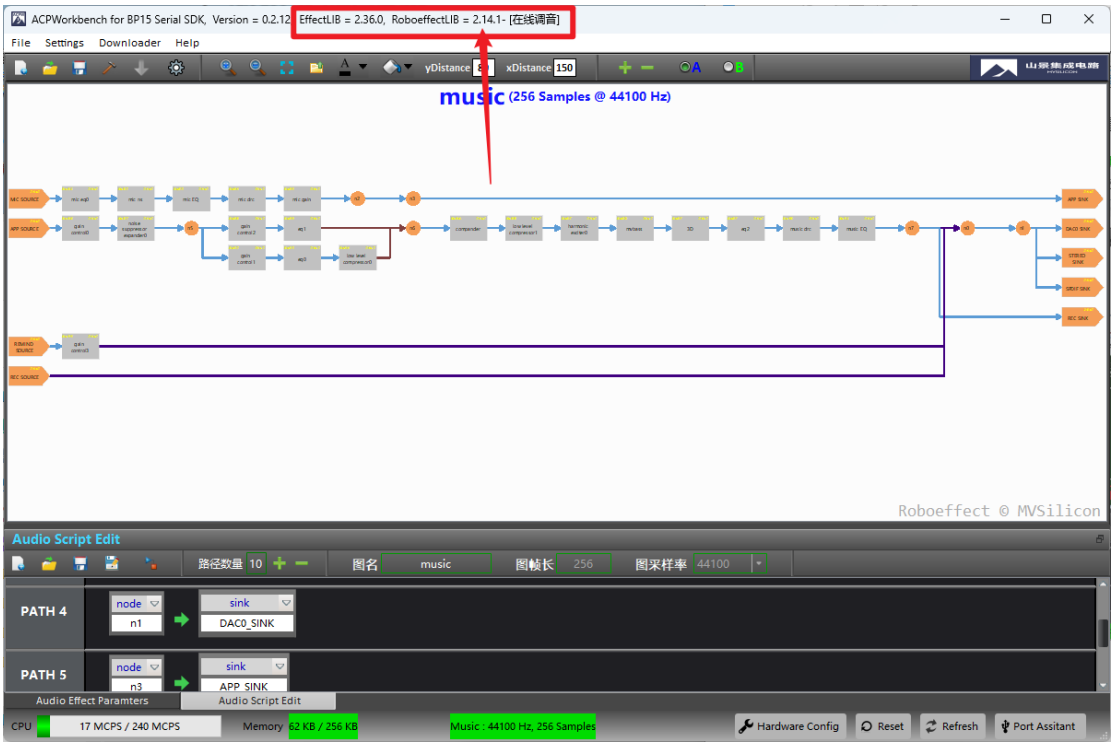


6.4. 更新库

1. 替换音效库+roboeffect 库



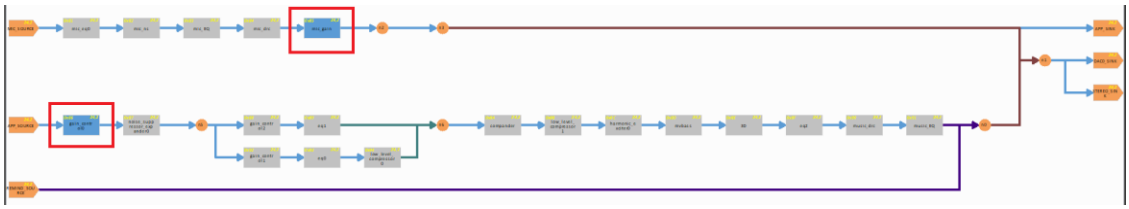
2. 编译烧录确认版本是否升级成功



7. 注意事项和常见问题

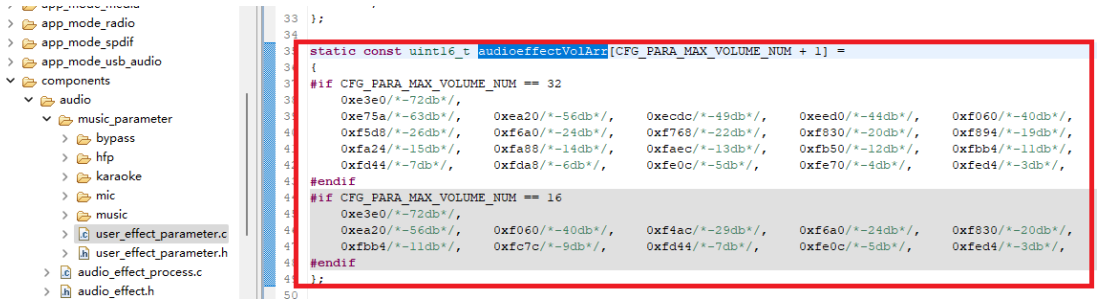
7.1. 音量控制

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



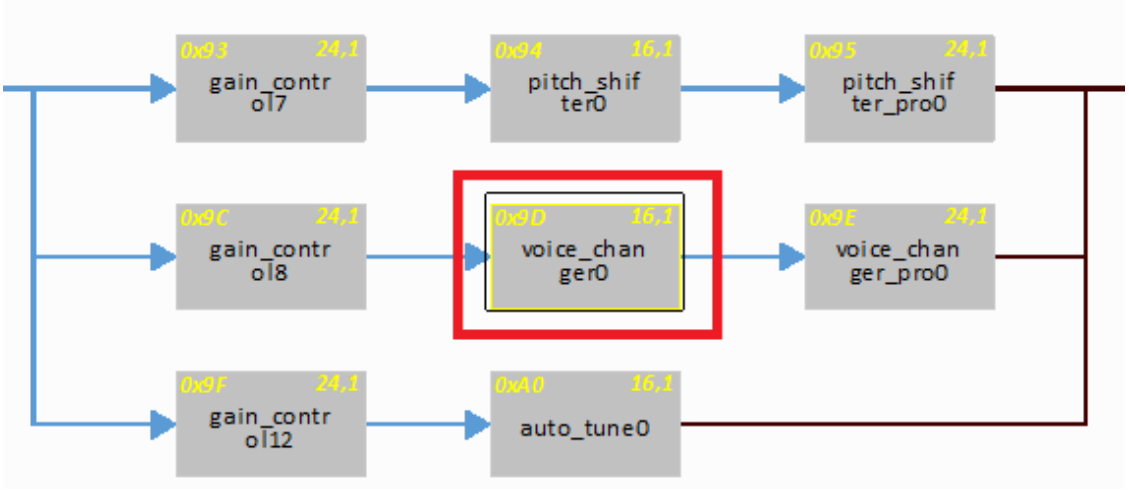
7.2. frame size 和 sample rate 的修改

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 则需修改 app_config.h 中的系统宏，如音效框图与系统 sample rate 不一致，音效默认会跟随系统帧长设置，保证实际运行一致。

7.3. 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，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。



7.4. 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。

7.5. 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

7.6. 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*/
```

7.7. 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。

```
//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,
},
```