BP15 V3架构使用说明

BP15 V3架构使用说明
1. Roboeffect介绍
1.1 roboeffect文件说明
1.2 roboeffect工作流程
1.3 roboeffect API介绍
2. ACPWorkBench V3.x.x版本介绍
3. SDK音效架构设计
3.1 Roboeffect音效文件
3.1.1 音效flow文件
3.1.2 音效参数文件
3.2 Roboeffect Init
3.2.1 选择正确的框图和音效参数
3.2.2 计算需要的内存大小并尝试申请
3.2.3 roboeffect_init()初始化roboeffect引擎
3.2.4 初始化上位机交互模块
3.3 Source & Sink Init
3.4 Effect Process
3.5 在线调音
4. 快速定制音效
4.1 音效宏的选择
4.2 定制框图
4.2.1 增加/删除音效
4.2.2 新增/删掉输入输出源
4.3 定制音效参数
5. 注意事项和常见问题
5.1 音量控制
5.2 帧长的切换
5.3 调音文件的导入导出
5.3.1 音效flow文件
5.3.2 音效参数文件
5.3 调音工具与USB debug工具的冲突
5.4 frame size和sample rate修改

1. Roboeffect介绍

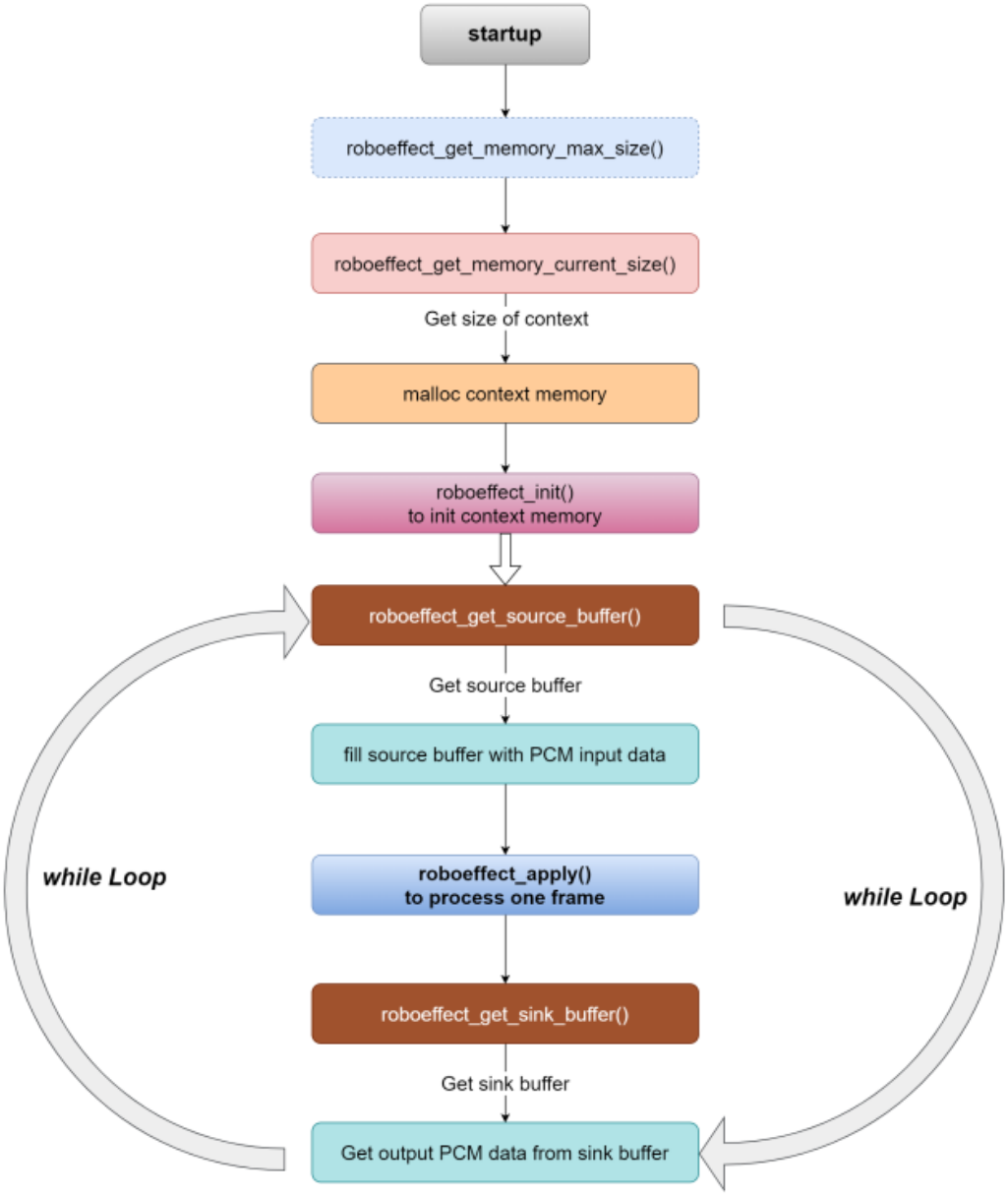
Roboeffect引擎是V3版本提出的新模型，提供所见即所得的可视化图形能力，只需简单的操作即可完成复杂的音效定制化开发。

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，以及若干结构体的定义
|   +--- roboeffect_config.h        #音效宏开关和音效接口声明
|   +--- user_defined_effect_api.h  #外部自定义有关roboeffect的api函数声明
+--- 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  #外部自定义有关roboeffect的api函数实现
```

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 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

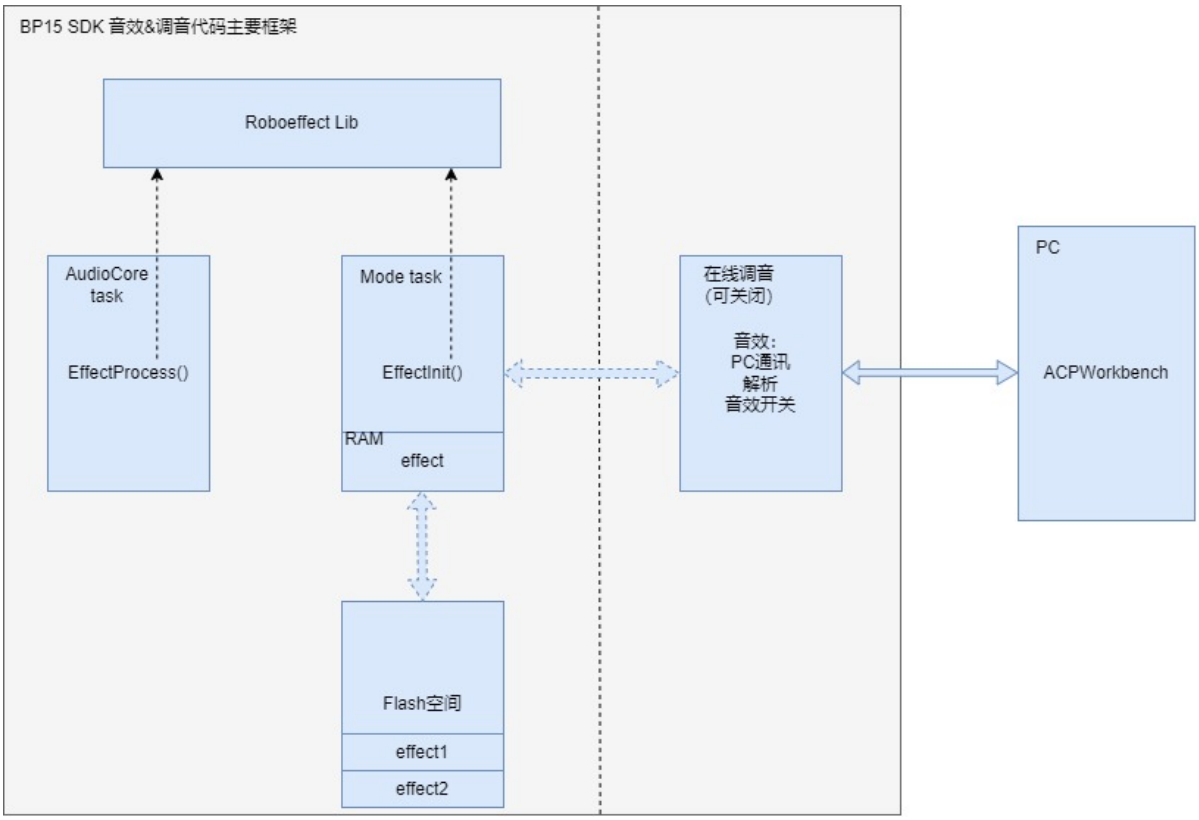
2. ACPWorkBench V3.x.x版本介绍

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

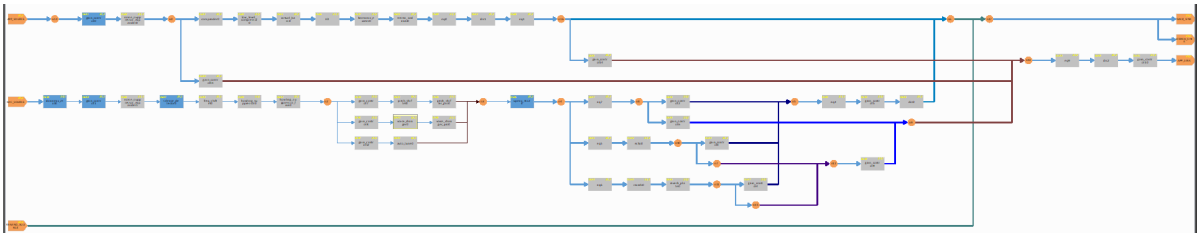
BP15 SDK 音效和调音的软件设计架构如下图所示 。



BP15 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

BP15 SDK的一个音效框图在调音工具的展示如下：



SDK的音效flow由音效框图决定，根据该图会产生如下C和H头文件：

```
./app_src/components/audio/music_parameter
+--- inc
|   +--- user_effect_flow_music.h    #若干结构定义声明
+--- src
|   +--- user_effect_flow_music.c    #音效框图描述以及若干结构定义
|   +--- user_effect_param_music.c   #音效参数和硬件配置参数
```

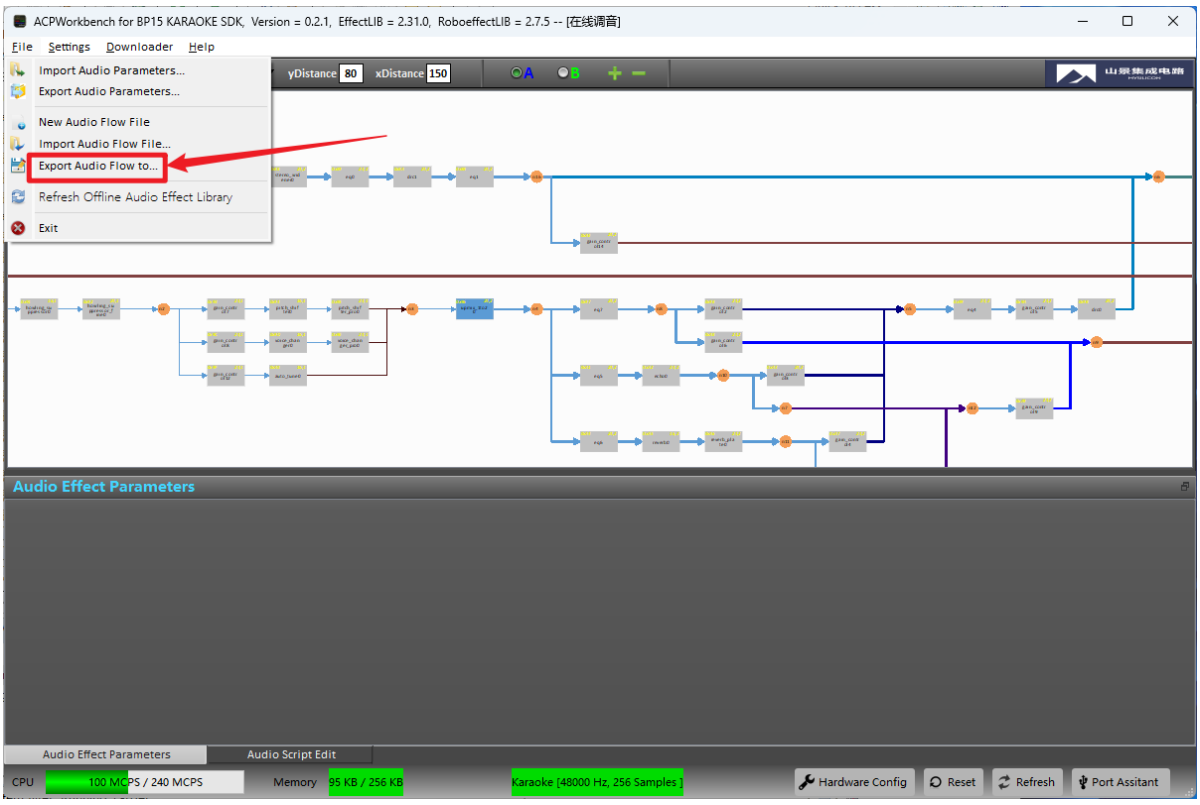
BP15 SDK通过音效功能宏来控制音效，便于用户开关宏来调试音效，量产时关闭部分宏来节省代码和内存的使用，具体见下表。

宏定义	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

SDK中的音效和调音相关文件如下表。

音效文件和目录	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_defined_api.c/user_defined_api.h	SDK自定义若干调用roboeffect库功能的函数

3.1.1 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。

```
 /***********user_effect_flow_music.h/************/
//输入输出定义
```

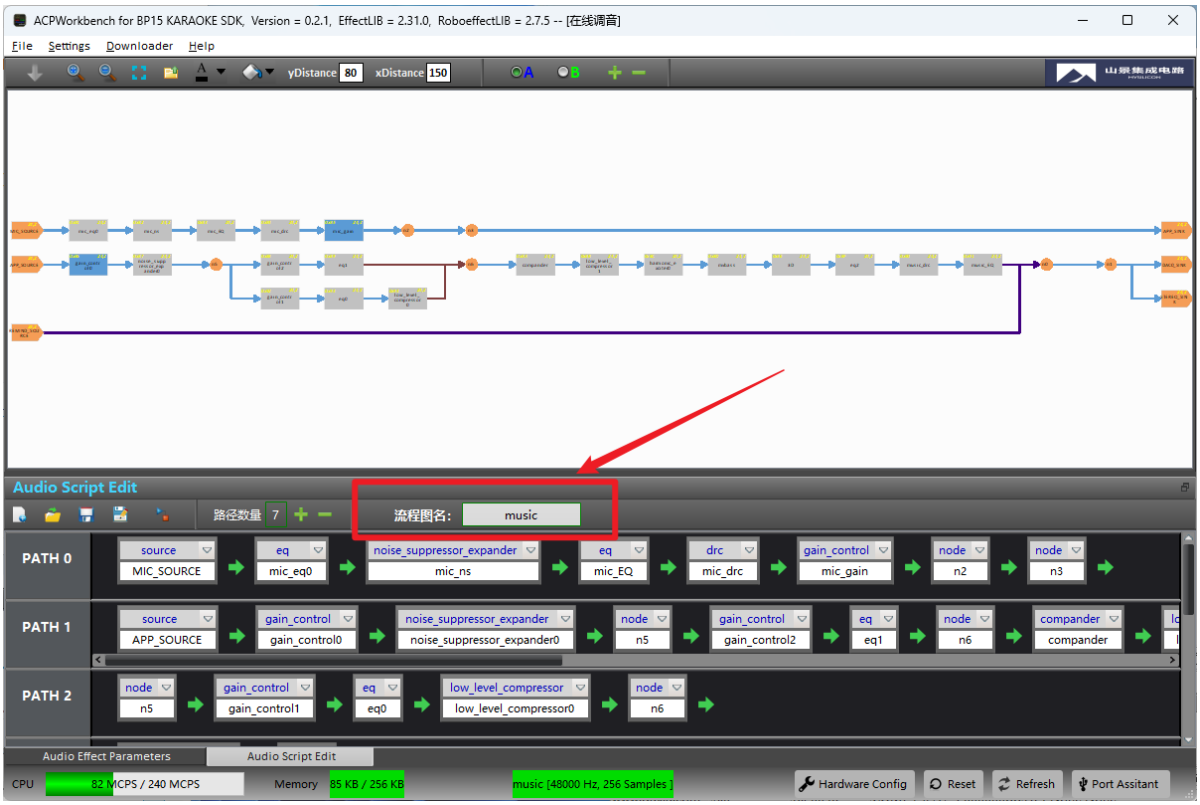
```
typedef enum _MUSIC_roboeffect_io_enum
{
    xxxxxx,
    xxxxxx,
    ...
} MUSIC_roboeffect_io_enum;

//框图使用的音效列表
typedef enum _MUSIC_roboeffect_effect_list_enum{
    xxxxxx,
    xxxxxx,
    ...
} MUSIC_roboeffect_effect_list_enum;
```

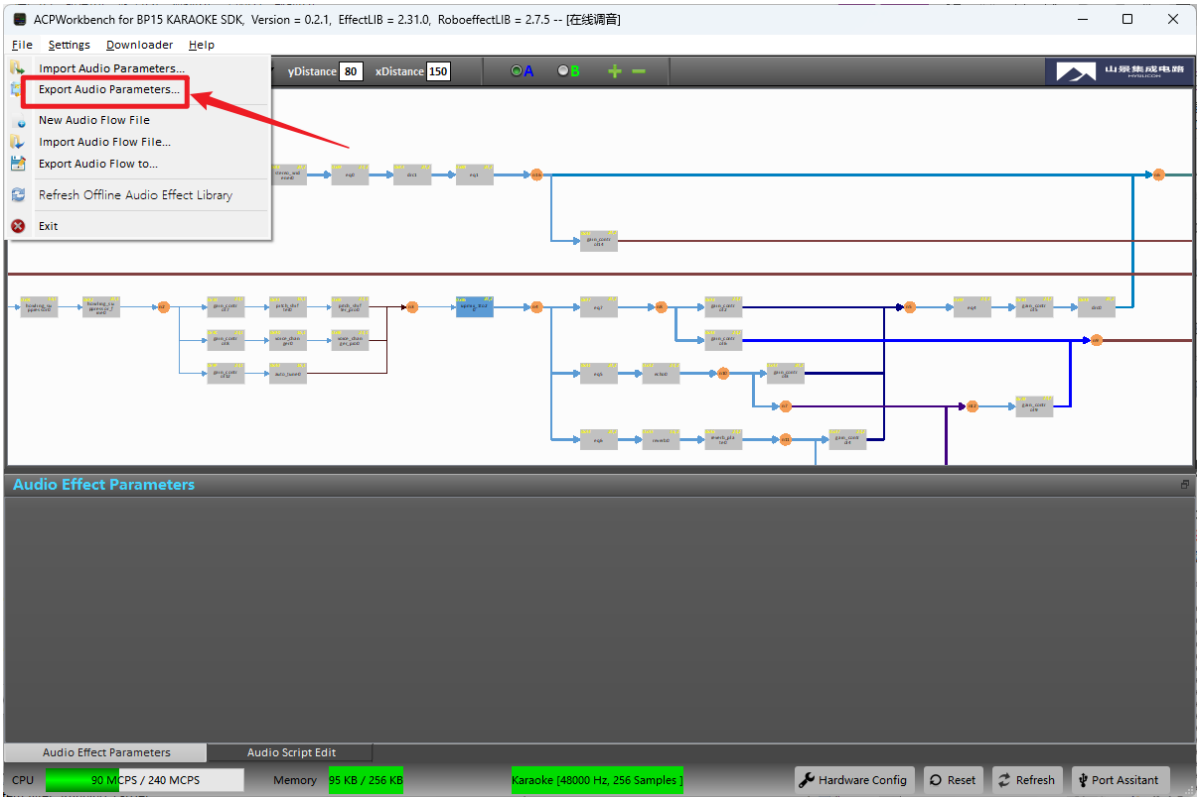
```
 /***********user_effect_flow_music.c/************/
//音效框图加密描述
const unsigned char user_effects_script_music[] = {
    xxxx......
}
//音效细节描述
static const roboeffect_exec_effect_info user_effects_music[] = {
    {XXX   , XXX   , XXX   , XXX},  //mic_eq0
    ...
};

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，上述信息应全部由调音工具导出，不建议手动修改。



3.1.2 音效参数文件

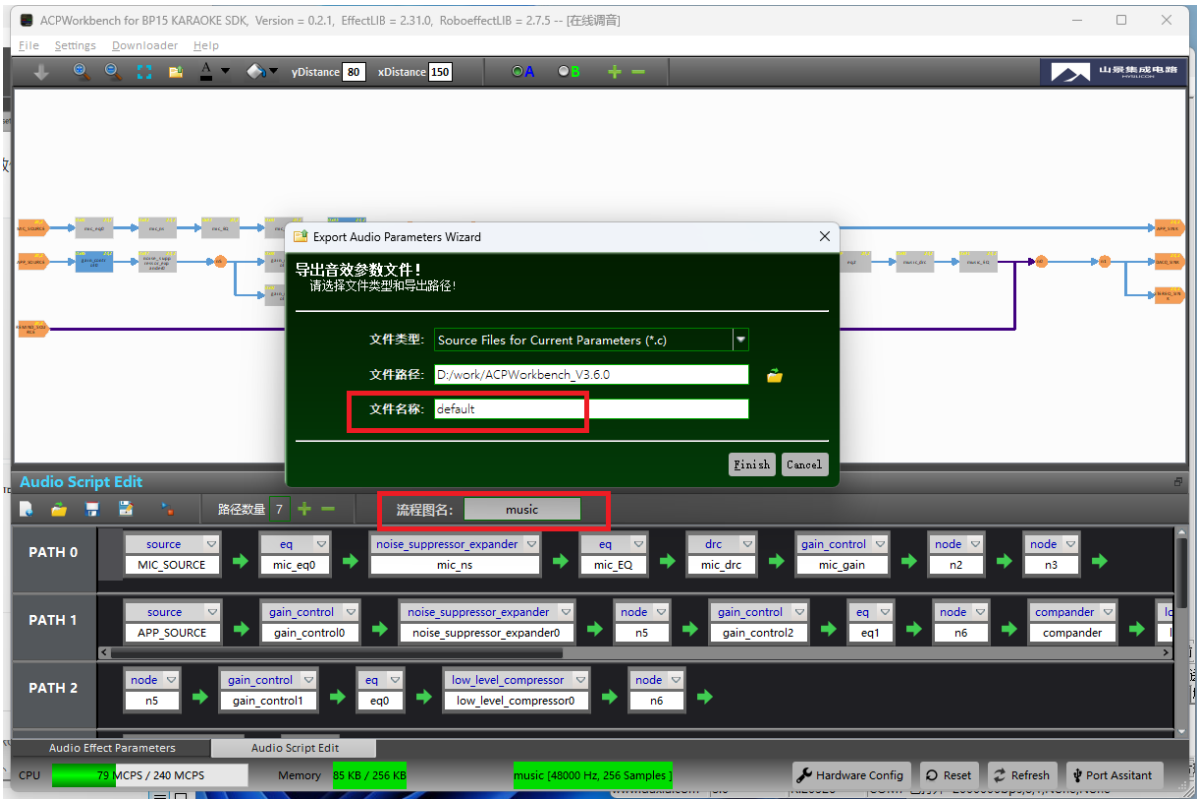


音效参数文件（user_effect_param_xxx.c）由调音工具导出，主要包含调音完成的音效参数信息。一个音效框图可以有多组不同的音效参数。

```
//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。



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 flow_chart_mode;
    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;
```

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。

```
  if(mainAppCt.EffectMode == EFFECT_MODE_MIC)
  {
      memcpy(&local_effect_list, &user_effect_list_mic, sizeof(roboeffect_effect_list_info));
      AudioCore.Roboeffect.user_effect_list = (roboeffect_effect_list_info *)&local_effect_list;
      AudioCore.Roboeffect.user_effect_steps = (roboeffect_effect_steps_table *)&user_effect_steps_mic;
      AudioCore.Roboeffect.user_effects_script = (uint8_t *)user_effects_script_mic;
      AudioCore.Roboeffect.user_effects_script_len = (uint16_t)get_user_effects_script_len_mic();
      AudioCore.Roboeffect.user_effect_parameters = osPortMalloc(
              get_user_effect_parameters_len((uint8_t *)user_effect_parameters_mic_mic) * sizeof(uint8_t));
      memcpy(AudioCore.Roboeffect.user_effect_parameters, (uint8_t *)user_effect_parameters_mic_mic,
              get_user_effect_parameters_len((uint8_t *)user_effect_parameters_mic_mic) * sizeof(uint8_t));
      AudioCore.Roboeffect.user_module_parameters = (uint8_t *)user_module_parameters_mic_mic;
      AudioCore.Roboeffect.flow_chart_mode = 0;
      AudioCore.Roboeffect.effect_count = MIC_COUNT_ADDR - 1;
      DBG("EFFECT_MODE Mic\n");
  }
  else if(mainAppCt.EffectMode == EFFECT_MODE_MUSIC)
  {
      memcpy(&local_effect_list, &user_effect_list_music, sizeof(roboeffect_effect_list_info));
      AudioCore.Roboeffect.user_effect_list = (roboeffect_effect_list_info *)&local_effect_list;
      AudioCore.Roboeffect.user_effect_steps = (roboeffect_effect_steps_table *)&user_effect_steps_music;
      AudioCore.Roboeffect.user_effects_script = (uint8_t *)user_effects_script_music;
      AudioCore.Roboeffect.user_effects_script_len = (uint16_t)get_user_effects_script_len_music();
      AudioCore.Roboeffect.user_effect_parameters = osPortMalloc(
              get_user_effect_parameters_len((uint8_t *)user_effect_parameters_music_music) * sizeof(uint8_t));
      memcpy(AudioCore.Roboeffect.user_effect_parameters, (uint8_t *)user_effect_parameters_music_music,
              get_user_effect_parameters_len((uint8_t *)user_effect_parameters_music_music) * sizeof(uint8_t));
      AudioCore.Roboeffect.user_module_parameters = (uint8_t *)user_module_parameters_music_music;
      AudioCore.Roboeffect.flow_chart_mode = 1;
      AudioCore.Roboeffect.effect_count = MUSIC_COUNT_ADDR - 1;
      DBG("EFFECT_MODE Music\n");
  }
```

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;
}
```

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。

```
//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, buf, 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

BP15 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BP15 Karaoke为例。

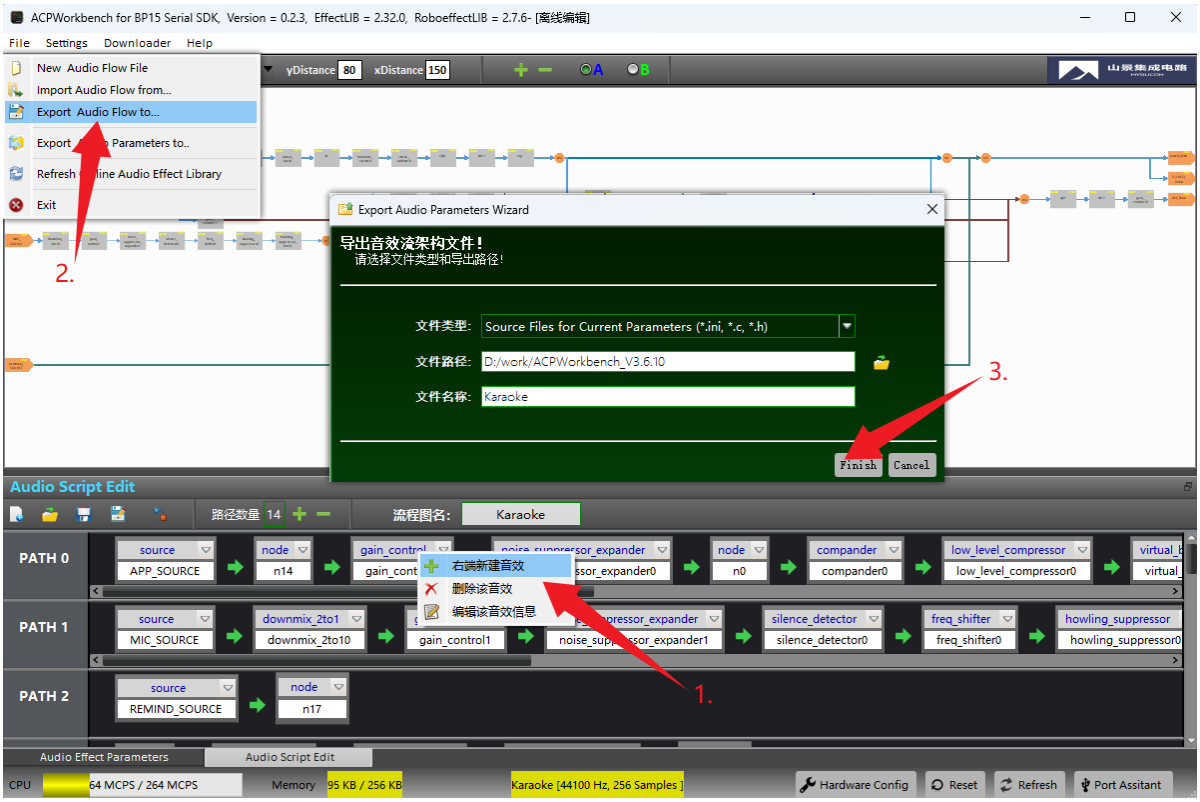
4.1 音效宏的选择

BP15 SDK 中对于各种音效用宏进行了管理，当某些音效确定不会使用时，将 roboeffect_config.h 文件中对应音效的宏配置为“0”，这样这部分代码以及相关的音效库函数均不会被包含到 SDK 代码中来，可以减少代码量。

4.2 定制框图

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

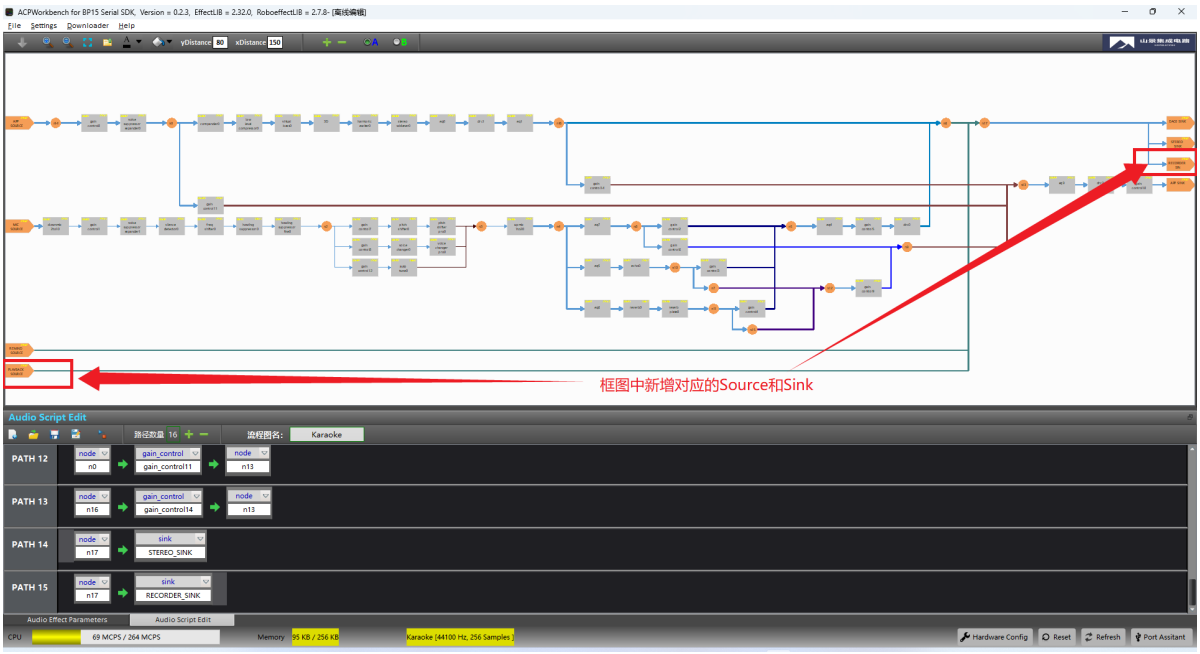
4.2.1 增加/删除音效



将生成.c和.h文件替换至SDK目录(./app_src/components/audio/music_parameter)下的**.inc**和**.src**中，导入对应文件到SDK后，请参考**4.3小节**的流程继续修改或添加音效。

4.2.2 新增/删掉输入输出源

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



然后按照**4.2.1小节**的流程导出框图文件到SDK对应目录下。

注意：框图中source和sink改动之后一定要更新**user_defined_effect_api.c/.h**如下部分代码。

user_defined_effect_api.h : Working Copy

```
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[];
```

user_defined_effect_api.c : Working Copy

```
const ROBOEFFECT_EFFECT_ADDR effect_addr[] = {
  //{music_EQ ,mic_EQ ,REVERB  ,ECHO  ,silence  ,APP_source  ,remind  ,mic  ,DAC0  ,APP_sink  ,stereo},
    {0x0      ,0x0    ,0x0     ,0x0   ,0x0      ,0x86        ,0x0     ,0x85 ,0x0   ,0x0       ,0x0},//mic
    {0x0      ,0x0    ,0x0     ,0x0   ,0x0      ,0x86        ,0x0     ,0x85 ,0x0   ,0x0       ,0x0},//music
    {0x0      ,0x0    ,0x0     ,0x0   ,0x0      ,0x86        ,0x0     ,0x85 ,0x0   ,0x0       ,0x0},//HFP
    {0x0      ,0x0    ,0x0     ,0x0   ,0x8F     ,0x81        ,0x0     ,0x8D ,0x0   ,0xAC      ,0x0},//HUNXIANG
    {0x0      ,0x0    ,0x0     ,0x0   ,0x8F     ,0x81        ,0x0     ,0x8D ,0x0   ,0xAC      ,0x0},//DIANYIN
    {0x0      ,0x0    ,0x0     ,0x0   ,0x8F     ,0x81        ,0x0     ,0x8D ,0x0   ,0xAC      ,0x0},//MOYIN
    {0x0      ,0x0    ,0x0     ,0x0   ,0x8F     ,0x81        ,0x0     ,0x8D ,0x0   ,0xAC      ,0x0},//HANMAI
    {0x0      ,0x0    ,0x0     ,0x0   ,0x8F     ,0x81        ,0x0     ,0x8D ,0x0   ,0xAC      ,0x0},//NANBIANNV
    {0x0      ,0x0    ,0x0     ,0x0   ,0x8F     ,0x81        ,0x0     ,0x8D ,0x0   ,0xAC      ,0x0},//NVBIANNAN
    {0x0      ,0x0    ,0x0     ,0x0   ,0x8F     ,0x81        ,0x0     ,0x8D ,0x0   ,0xAC      ,0x0},//WAWAYIN
};
const ROBOEFFECT_SOURCE_NUM roboeffect_source[] = {
  //{mic_source,      app_source,      remind_source,      playback_source},
    {MIC_SOURCE_MIC_SOURCE, MIC_SOURCE_APP_SOURCE, MIC_SOURCE_REMIND_SOURCE, MIC_SOURCE_PLAYBACK_SOURCE}, //mic
    {MUSIC_SOURCE_MIC_SOURCE, MUSIC_SOURCE_APP_SOURCE, MUSIC_SOURCE_REMIND_SOURCE, MUSIC_SOURCE_PLAYBACK_SOURCE}, //music
    {HFP_SOURCE_MIC_SOURCE, HFP_SOURCE_APP_SOURCE, HFP_SOURCE_REMIND_SOURCE, HFP_SOURCE_APP_SOURCE}, //HFP
    {KARAOKE_SOURCE_MIC_SOURCE, KARAOKE_SOURCE_APP_SOURCE, KARAOKE_SOURCE_REMIND_SOURCE, KARAOKE_SOURCE_PLAYBACK_SOURCE}, //HUNXIANG
    {KARAOKE_SOURCE_MIC_SOURCE, KARAOKE_SOURCE_APP_SOURCE, KARAOKE_SOURCE_REMIND_SOURCE, KARAOKE_SOURCE_PLAYBACK_SOURCE}, //DIANYIN
    {KARAOKE_SOURCE_MIC_SOURCE, KARAOKE_SOURCE_APP_SOURCE, KARAOKE_SOURCE_REMIND_SOURCE, KARAOKE_SOURCE_PLAYBACK_SOURCE}, //MOYIN
    {KARAOKE_SOURCE_MIC_SOURCE, KARAOKE_SOURCE_APP_SOURCE, KARAOKE_SOURCE_REMIND_SOURCE, KARAOKE_SOURCE_PLAYBACK_SOURCE}, //HANMAI
    {KARAOKE_SOURCE_MIC_SOURCE, KARAOKE_SOURCE_APP_SOURCE, KARAOKE_SOURCE_REMIND_SOURCE, KARAOKE_SOURCE_PLAYBACK_SOURCE}, //NANBIANNV
    {KARAOKE_SOURCE_MIC_SOURCE, KARAOKE_SOURCE_APP_SOURCE, KARAOKE_SOURCE_REMIND_SOURCE, KARAOKE_SOURCE_PLAYBACK_SOURCE}, //NVBIANNAN
    {KARAOKE_SOURCE_MIC_SOURCE, KARAOKE_SOURCE_APP_SOURCE, KARAOKE_SOURCE_REMIND_SOURCE, KARAOKE_SOURCE_PLAYBACK_SOURCE}, //WAWAYIN
};
const ROBOEFFECT_SINK_NUM roboeffect_sink[] = {
  //{dac0_sink,      app_sink,      stereo_sink,      recorder_sink},
    {MIC_SINK_DAC0_SINK, MIC_SINK_APP_SINK, MIC_SINK_STEREO_SINK, MIC_SINK_RECORDER_SINK}, //mic
    {MUSIC_SINK_DAC0_SINK, MUSIC_SINK_APP_SINK, MUSIC_SINK_STEREO_SINK, MUSIC_SINK_RECORDER_SINK}, //music
    {HFP_SINK_DAC0_SINK, HFP_SINK_APP_SINK, HFP_SINK_STEREO_SINK, HFP_SINK_APP_SINK}, //HFP
    {KARAOKE_SINK_DAC0_SINK, KARAOKE_SINK_APP_SINK, KARAOKE_SINK_STEREO_SINK, KARAOKE_SINK_RECORDER_SINK}, //HUNXIANG
    {KARAOKE_SINK_DAC0_SINK, KARAOKE_SINK_APP_SINK, KARAOKE_SINK_STEREO_SINK, KARAOKE_SINK_RECORDER_SINK}, //DIANYIN
    {KARAOKE_SINK_DAC0_SINK, KARAOKE_SINK_APP_SINK, KARAOKE_SINK_STEREO_SINK, KARAOKE_SINK_RECORDER_SINK}, //MOYIN
    {KARAOKE_SINK_DAC0_SINK, KARAOKE_SINK_APP_SINK, KARAOKE_SINK_STEREO_SINK, KARAOKE_SINK_RECORDER_SINK}, //HANMAI
    {KARAOKE_SINK_DAC0_SINK, KARAOKE_SINK_APP_SINK, KARAOKE_SINK_STEREO_SINK, KARAOKE_SINK_RECORDER_SINK}, //NANBIANNV
    {KARAOKE_SINK_DAC0_SINK, KARAOKE_SINK_APP_SINK, KARAOKE_SINK_STEREO_SINK, KARAOKE_SINK_RECORDER_SINK}, //NVBIANNAN
    {KARAOKE_SINK_DAC0_SINK, KARAOKE_SINK_APP_SINK, KARAOKE_SINK_STEREO_SINK, KARAOKE_SINK_RECORDER_SINK}, //WAWAYIN
};
```

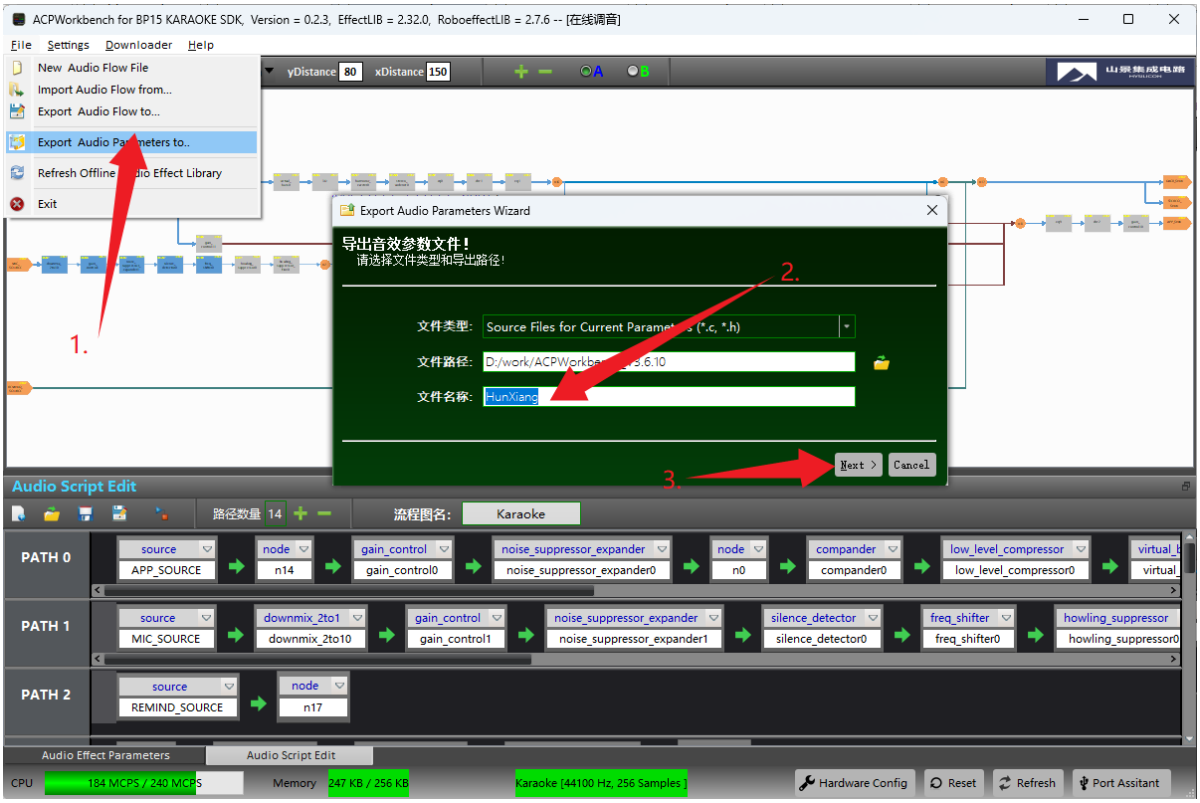
user_defined_effect_api.c : Working Copy

```
665 uint8_t AudioCoreSourceToRoboeffect(int8_t source)
666 {
667
668     switch (source) {
669         case MIC_SOURCE_NUM:
670             return roboeffect_source[AudioCore.Roboeffect.flow_chart_mode].mic_source;
671         case APP_SOURCE_NUM:
672             return roboeffect_source[AudioCore.Roboeffect.flow_chart_mode].app_source;
673         case REMIND_SOURCE_NUM:
674             return roboeffect_source[AudioCore.Roboeffect.flow_chart_mode].remind_source;
675         case PLAYBACK_SOURCE_NUM:
676             return roboeffect_source[AudioCore.Roboeffect.flow_chart_mode].playback_source;
677         default:
678             //handle error
679             return roboeffect_source[AudioCore.Roboeffect.flow_chart_mode].app_source;
680     }
681     return 0;
682 }
683
684 uint8_t AudioCoreSinkToRoboeffect(int8_t sink)
685 {
686     switch (sink) {
687         case AUDIO_DAC0_SINK_NUM:
688             return roboeffect_sink[AudioCore.Roboeffect.flow_chart_mode].dac0_sink;
689 #if (defined(CFG_APP_BT_MODE_EN) && (BT_HFP_SUPPORT == ENABLE)) || defined(CFG_APP_USB_AUDIO_MODE_EN)
690         case AUDIO_APP_SINK_NUM:
691             return roboeffect_sink[AudioCore.Roboeffect.flow_chart_mode].app_sink;
692 #endif
693 #if defined(CFG_RES_AUDIO_I2SOUT_EN)
694         case AUDIO_STEREO_SINK_NUM:
695             return roboeffect_sink[AudioCore.Roboeffect.flow_chart_mode].stereo_sink;
696 #endif
697         case AUDIO_RECORDER_SINK_NUM:
698             return roboeffect_sink[AudioCore.Roboeffect.flow_chart_mode].app_sink;
699         default:
700             //handle error
701             return roboeffect_sink[AudioCore.Roboeffect.flow_chart_mode].app_sink;
702     }
703     return 0;
704 }
705
```

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：



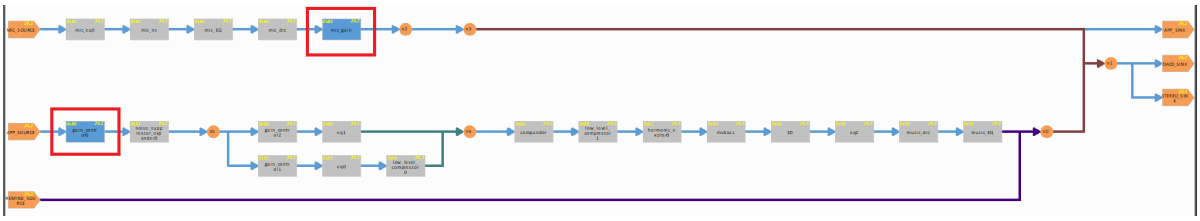
将生成文件替换至SDK目录(./app_src/components/audio/music_parameter/src)重新编译烧录即可。如果是新增音效，导入音效参数文件到SDK后，需参考混响的流程修改SDK如下部分代码。

1. ctrlvars.h 中共用体**EFFECT_MODE**中新增SDK音效名；
2. mode_task_api.c 中的**RoboeffectInit()**增加新的音效初始化逻辑；
3. user_defined_effect_api.h 中共用体**ROBOEFFECT_EFFECT_MODE**新增引擎音效名；
4. user_defined_effect_api.c 中更新结构**effect_addr、roboeffect_source和roboeffect_sink。**

5. 注意事项和常见问题

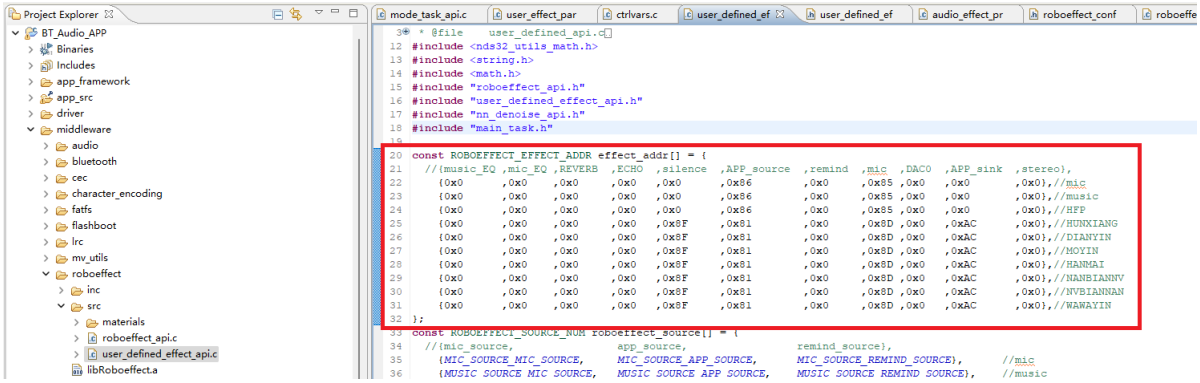
5.1 音量控制

2.1 音量控制依赖于音效框图中的gain control音效；



2.2 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；

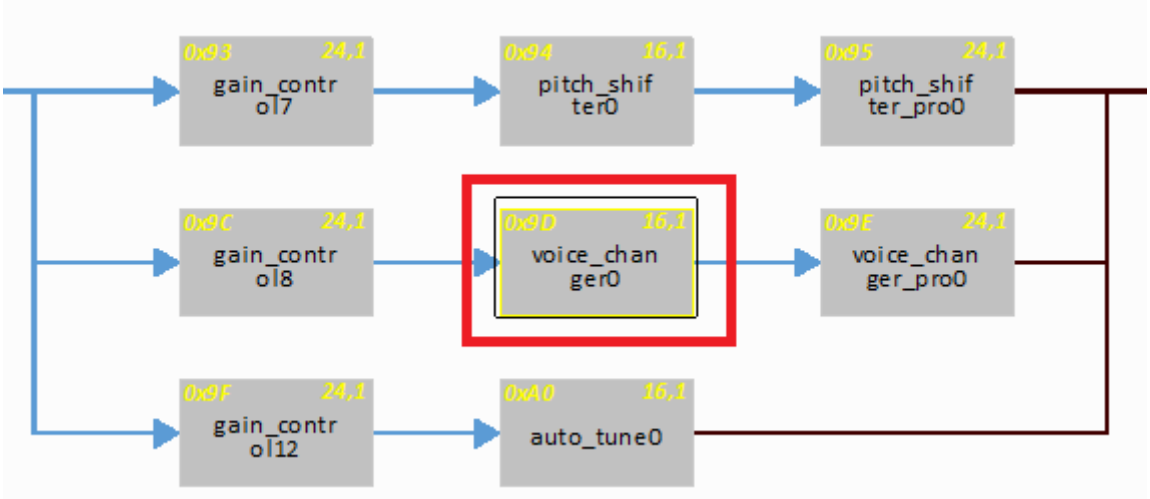
2.3 修改框图或者新加框图，需在代码中如下位置更新音量控制gain地址，否则会导致音量控制不生效甚至死机；



2.4 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。

5.2 帧长的切换

1. 通常情况下，帧长的大小由宏CFG_PARA_SAMPLES_PER_FRAME决定。
2. 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，系统帧长会切换回宏定义大小。

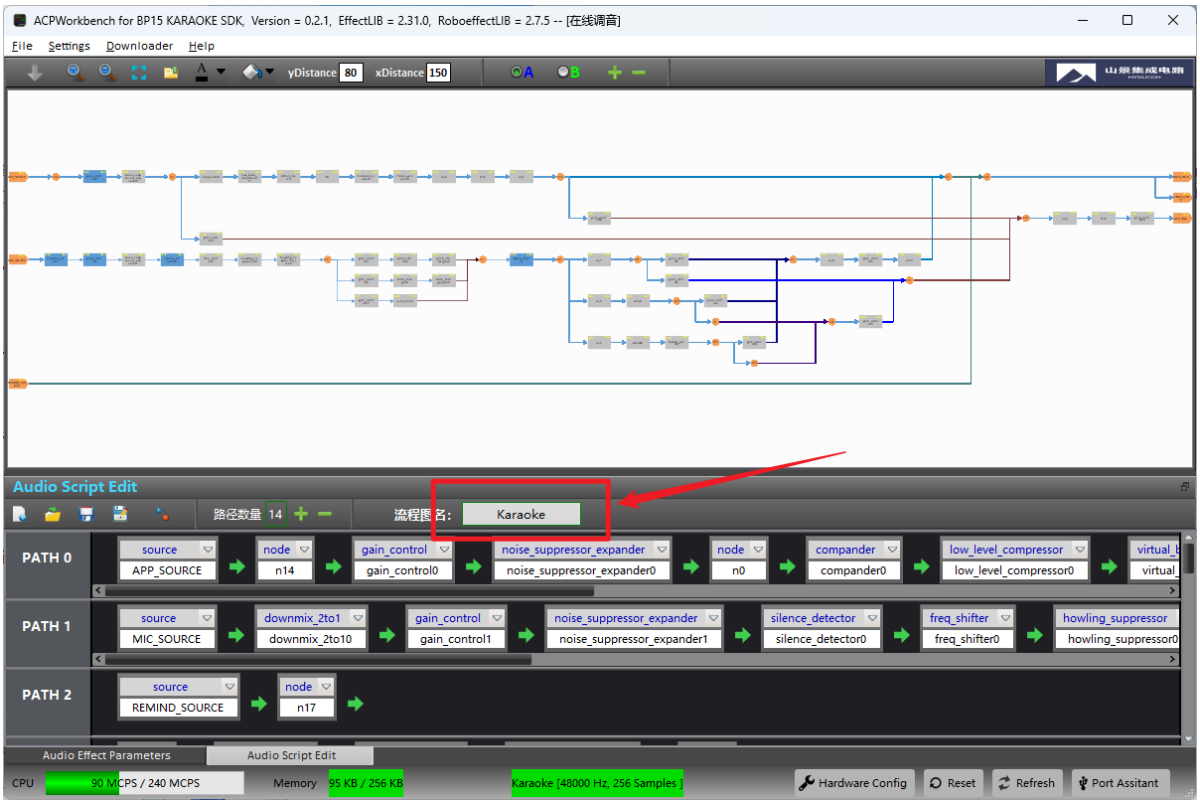


5.3 调音文件的导入导出

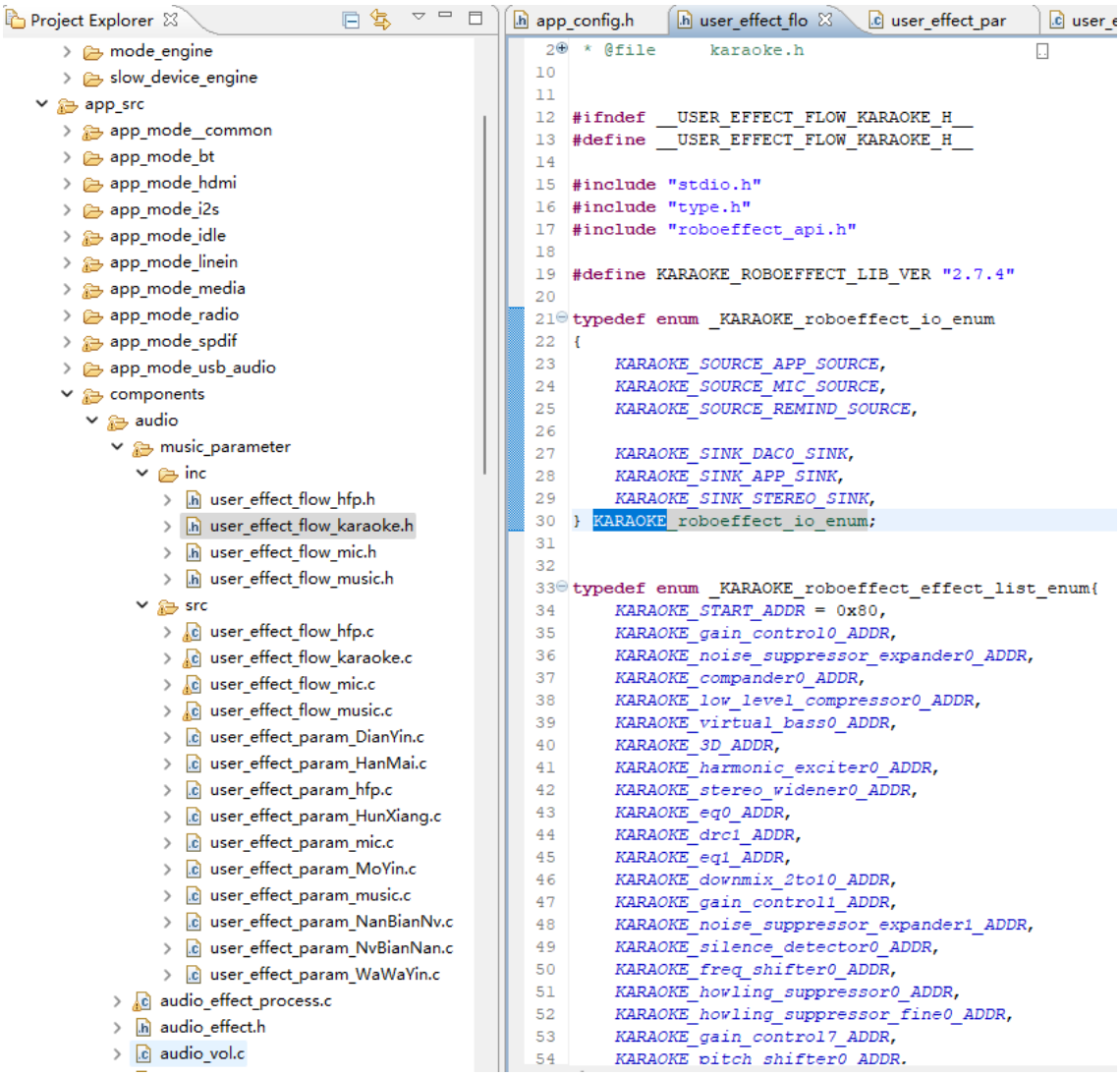
调音文件主要分为音效flow和音效参数两种，需要注意的是音效参数是跟一些flow深度绑定的，在使用调音工具导出的时候一定要明确导出的音效参数对应的音效flow是哪一个。

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。

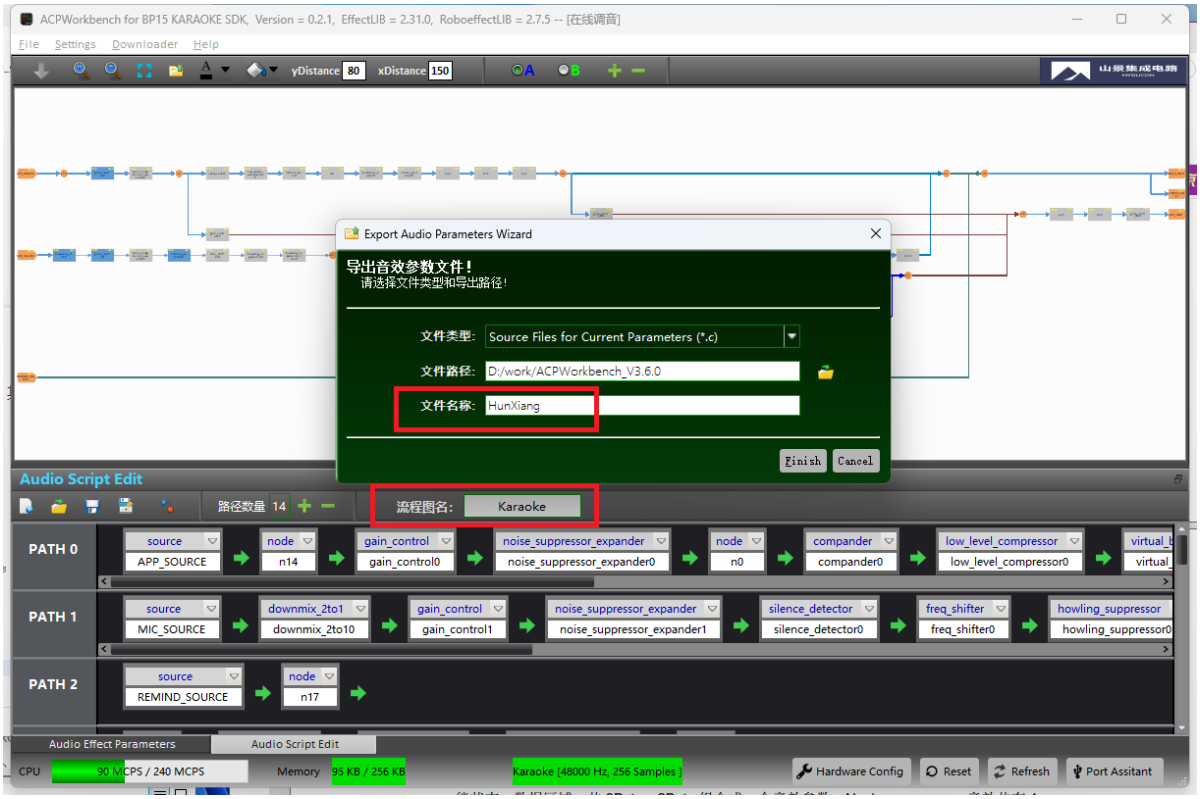


音效flow文件导出的命名为user_effect_flow_xxx.c/h，可以看到在导出的karaoke flow中，所有结构的命名都是以**KARAOKE**为前缀。



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。



```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*/
```

5.3 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。

5.4 frame size和sample rate修改

在修改系统frame size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。sample rate同理。