# Audiobox\_v2 设计文档

### 适用产品

|  |
| --- |
| 类别 |
| 软件版本 |
| 芯片型号 |

### 修订记录

|  |
| --- |
| 修订说明 |
| 初版 |
| v2.1 |

### 术语解释

|  |
| --- |
| 术语 |
| QSDK |
| Audiobox |
| API |
| AEC |
| AEC1.0 |
| AEC2.0 |
| Eventhub |

|  |
| --- |
| ## 1. Audiobox 基础 Audiobox 是QSDK的音频子系统，提供音频的录音和放音功能支持，同时支持静音(Mute)控制、音量(Volume)控制、音频流格式(Format)控制、回声消除(AEC)控制等音频通道参数管理功能。另外还提供 abctrl 工具用来测试 Audiobox 各项功能 |
| 此文档针对Audiobox子系统的开发人员，对Audiobox的内部机制进行了简要介绍 |
|  |

## 2. 基本功能架构

### 2.1 基本需求

初版的 Audiobox 版本功能比较单一，通过音频通道只能获取到PCM格式音频，录音和放音通道的参数也必须保持一致，而且实际上只能支持一路录音，无法满足复杂应用场景的开发需求。因此，我们以新的框架重构了 Audiobox 模块。本文后续描述中，将以audiobox\_v1表示初版，以audiobox\_v2表示新版本。audiobox\_v2保持完全向前兼容，原有的上层应用程序，无需任何修改即可从audiobox\_v1升级到audiobox\_v2

audiobox\_v2除了基本的录音和放音需求外，还需要满足如下设计要求： - 集成多种codec，可以申请带编解码的通道 - 录音通道和放音通道格式可以不同，通道参数可以和设备参数不一致，可以支持多个通道同时录音放音，并且各个通道可以采用不同参数 - 相同参数的录音通道可共享编码处理，以节省开销 - 支持AEC功能，支持基于DSP的alango算法和基于ARM的speex算法，支持AEC1.0和可配置延时的AEC2.0 - 申请录音通道时可以指定从最老帧还是最新帧开始获取音频流 - 可以申请4声道(包含两个录音声道和两个放音声道)格式的录音通道

要升级到最新的audiobox\_v2，system、kernel和buildroot都需要同步更新。system部分除Audiobox模块版本升级外，还删除了对vcp7g模块的依赖，并调整了hlibdsp和hlibunitrace模块；buildroot/package中增加了speexdsp支持；kernel中infotm驱动优化了ceva-dsp和sound模块

|  |
| --- |
| ### 2.2 基于Eventhub进程实现的API接口 Audiobox提供了一整套API接口给其他进程调用，这些API调用基于进程间消息队列机制实现，主要函数如下，具体请查看audiobox.h头文件 ```cpp int audio\_start\_service(void); int audio\_stop\_service(void); int audio\_get\_format(const char *dev, audio\_fmt\_t* fmt); int audio\_set\_format(const char *dev, audio\_fmt\_t* fmt); int audio\_get\_master\_volume(const char *dev, ...);//ignore origional parameter: int dir int audio\_set\_master\_volume(const char* dev, int volume, ...);//ignore origional parameter: int dir int audio\_get\_channel\_ex(const char *dev, audio\_chn\_fmt\_t* chn\_fmt, int flag); int audio\_enhance\_mode(void); int audio\_get\_channel(const char *dev, audio\_fmt\_t* fmt, int flag); int audio\_put\_channel(int handle); int audio\_get\_mute(int handle); int audio\_set\_mute(int handle, int mute); int audio\_get\_volume(int handle); int audio\_set\_volume(int handle, int volume); int audio\_get\_frame(int handle, struct fr\_buf\_info *buf); int audio\_put\_frame(int handle, struct fr\_buf\_info* buf); int audio\_read\_frame(int handle, char *buf, int size); int audio\_write\_frame(int handle, char* buf, int size); int audio\_enable\_aec(int handle, int enable); |
| /\* v2.1文档对应的新增api接口*/ int audio\_get\_master\_volume\_ex(const char* dev, int index); int audio\_set\_master\_volume\_ex(const char *dev, int index, int volume); int audio\_pause\_channel(int handle, int flag); int audio\_pause\_device(const char* dev, int enable); int audio\_get\_statistics(int handle, struct channel\_statistic *statistic); int audio\_get\_delay(int handle); int audio\_playbuf\_empty(int handle); int audio\_query\_channel(int handle, audio\_chan\_info\_t* chn\_info); int audio\_get\_dbginfo(audio\_dbg\_info\_t\* devinfo); int audio\_show\_devinfo(const char *dev); int audio\_show\_chninfo(int handle); int audio\_get\_initial\_devpara(dev\_attr\_setting\_t* para); int audio\_set\_audioboxpara(const char *dev, audiobox\_setting\_t* para, int setting\_type); int audio\_set\_devpara(const char *dev, dev\_attr\_setting\_t* para); int audio\_set\_tracepara(const char *dev, trace\_setting\_t* para); int audio\_set\_aecfreq(const char *dev, int freq); int audio\_set\_alsadepth(const char* dev, int depth); ``` |
| 典型的应用中，应用进程、Audiobox进程、Eventhub进程在启动过程中会分别以自身进程的pid为key创建消息队列。启动阶段，Audiobox进程在audiobox\_listener\_init（）函数中调用event\_register\_handler()子函数向eventhub进程注册，声明自己需要eventhub转发消息，并创建一个线程侦听自身消息队列。当应用进程调用Audiobox的API接口时，最终会发送一个消息到eventhub消息队列，eventhub把这个消息转发到audiobox消息队列。audiobox从消息队列获取消息后，根据消息类型调用合适的回调函数处理。应用API的回调函数是audiobox\_listener()，接着它会为不同的命令调用不同的功能函数 |
| 例如，应用进程调用audio\_get\_channel()函数，最终将调用event\_rpc\_call()函数发送AB\_GET\_CHANNEL命令到eventhub消息队列；eventhub转发此消息到audiobox消息队列；audiobox从消息队列中读取到消息后，调用audiobox\_listener()函数，其中调用audiobox\_get\_channel()子函数处理AB\_GET\_CHANNEL命令，创建音频通道 |

## 3. 音频通道管理

### 3.1 基本数据结构

本小节详细描述了三个基本的数据结构audio\_dev\_t、audio\_codec\_t、audio\_chn\_t，以及它们是如何实现音频通道管理的

上图是依次创建如下音频通道时内部数据结构图：1. G711A录音通道；2. G711A录音通道(参数同1)；3. AAC录音通道；4. AAC录音通道(参数同3)；5. G711A放音通道；6. G711A放音通道。从图中可以看出，参数相同的两个录音通道(chan\_0和chan\_1，chan\_2和chan\_3)共享一个codec，而参数相同的放音通道(chan\_4和chan\_5)不会共享codec

**audio\_dev\_t结构体**

audio\_dev\_t定义对应一个音频设备： - node成员 接入audiobox\_devlist链表；添加设备节点的操作由devctrl\_create\_dev()函数中list\_add\_tail(&dev->node, &audiobox\_devlist);完成 - head成员 维护一个本音频设备的通道链表，每个节点对应一个audio\_chn\_t定义的音频通道；添加通道节点的操作由audiobox\_get\_channel()函数中list\_add\_tail(&chn->node, &dev->head);完成 - apulist成员 维护一个本音频设备的codec apu链表，每个节点对应一个apu\_codec\_t定义的code apu；添加节点的操作由\_\_apu\_codec\_create()函数中list\_add\_tail(&p\_codec->node, &dev->apulist);完成

|  |
| --- |
| 成员变量 |
| char devname[32]; |
| snd\_pcm\_t *handle; | PCM handle | | snd\_ctl\_t* ctl; |
| audio\_dev\_attr\_t attr; |
| pthread\_mutex\_t dev\_mutex; |
| pthread\_mutex\_t dev\_state\_mutex; |
| pthread\_cond\_t dev\_cond; |
| int direct; |
| int m\_volume; |
| int foreground\_chan\_counter; |
| pthread\_t hal\_server\_id; |
| int hal\_state; |
| int dev\_fr\_size; |
| int dev\_fr\_num; |
| int fr\_total\_size; |
| struct *apu\_frnode* dev\_fr\_para; |
| struct *apu\_frnode* dev\_fr\_para\_mapped; |
| struct *apu\_frnode* dev\_preprocfr\_para; |
| apu\_aec\_T preproc; |
| pthread\_t preproc\_pid; |
| int aecapu\_state; |
| int preproc\_enabled; |
| int preproc\_actived; |
| int aecvx\_select; |
| pthread\_t decoder\_pid; |
| int decoder\_state; |
| int trace\_mode; |
| int trace\_size; |
| int trace\_fd; |
| int devfr\_counter; |
| int aecfr\_counter; |
| int devfr\_counter\_mapped; |
|  |
| **struct list\_head node;** |
| **struct list\_head head;** |
| **struct list\_head apulist;** |

**apu\_codec\_t结构体**

apu\_codec\_t结构体定义对应一个codec apu： - node成员 接入音频设备的apulist链表 - vchanlist成员 维护一个引用本codec apu的通道链表，每个节点对应一个audio\_chn\_t定义的音频通道，这个链表上的所有通道具有相同的通道参数；添加通道节点的操作由\_\_apu\_codec\_create()函数中list\_add\_tail(&chn->chn\_node, &p\_codec->vchanlist);完成

|  |
| --- |
| 成员变量 |
| int apu\_state; |
| pthread\_t apu\_pid; |
| int ref\_counter; |
| int apu\_activated; |
| char *infrname; | codec输入fr名称 | | char* outfrname; |
| void *softvol\_handle; | softvol handle | | int softvol\_enabled; | 0: no softvol 1: set capture/playback master volume, set playback channel volume | | struct fr\_buf\_info softvol\_frinfo; | softvol process out buffer, virtual fr | | int insize; | recombine处理输入音频帧大小 | | int outsize; | recombine处理输出音频帧大小 | | int recombine\_enabled; | 0: no recombine mudule. 1: recombine mudule enabled | | struct fr\_buf\_info recombine\_frinfo; | | | void* codec\_handle; |
| struct *apu\_frnode* codec\_fr\_para; |
| struct *apu\_frnode* *infr\_para; | refer to input fr: channel's channel\_fr | | struct apu\_frnode* outfr\_para; |
| struct fr\_buf\_info infr\_ref; |
| struct fr\_buf\_info outfr\_buf; |
| struct fr\_buf\_info outfr\_ref; |
| audio\_chn\_fmt\_t fmt; |
| audio\_dev\_t dev; |
| int chan\_id; |
| int apu\_vol; |
| pthread\_mutex\_t vchan\_mutex; |
| int is\_prepared; |
|  |
| **struct list\_head node;** |
| **struct list\_head vchanlist;** |

**audio\_chn\_t结构体定义**

audio\_chn\_t定义对应一个音频通道： - node成员 接入音频通道对应音频设备的head链表 - chn\_node成员 接入本通道对应的codec apu的vchanlist链表

|  |
| --- |
| 成员变量 |
| char chnname[32]; |
| audio\_chn\_fmt\_t fmt; |
| int flag; |
| int timeout; |
| int priority; |
| int direct; |
| int id; |
| int chn\_pid; |
| int req\_stop\_play; |
| audio\_dev\_t dev; |
| int chan\_activated; |
| void \*apu\_handle; | codec handle | | struct *apu\_frnode* channel\_fr; | channel\_fr | | struct fr\_buf\_info chn\_fr\_buf; | for writing to record channel channel\_fr | | int chn\_fr\_size; | 通道fr buffer长度 | | int chn\_fr\_flag; | 通道fr flag | | int chn\_fr\_num; | 通道fr buffer个数 | | | | | **struct list\_head node;** | 通道在设备通道链表中的节点 | | **struct list\_head chn\_node;** | 通道在设备codec链表中的节点 | |

### 3.2 fr子系统

Audiobox的音频流处理基于fr子系统，每个处理子模块都是从一个输入fr中取得音频帧，处理之后输出到另一个输出fr中。Audiobox定义了如下fr: - chan\_fr 录音时，通道dispatch子模块输出fr，应用线程调用audio\_read\_frame()从此fr读取音频帧 - chan\_fr 放音时，通道codec子模块输入fr，应用线程调用audio\_write\_frame()函数将音频帧写入此fr - codec\_fr 通道codec子模块输出fr - recombine\_fr 通道recombine子模块输出fr - softvol\_fr 通道软音量处理子模块输出fr - aec\_fr 设备aec子模块输出fr；放音时不定义此fr - dev\_fr 录音时读取alsa音频流输出到此fr；放音时不定义此fr - dev\_fr\_mapped 使能AEC2.0录音时设备chan\_remap子模块输出fr；放音时不定义此fr

其中recombiner\_fr和softvol\_fr只是普通的buffer，而不是调用apu\_alloc\_fr()申请的实际fr。为了将几个子模块设计为基于fr子系统的统一模型，将这两个buffer抽象为虚拟的fr，并在audiobox实现中使用下面统一接口访问这几个真实的或虚拟的fr：

int \_apu\_fr\_get\_ref(void \*apu\_handle, struct fr\_buf\_info\* ref)  
int \_apu\_fr\_put\_ref(struct fr\_buf\_info\* ref)  
int \_apu\_fr\_get\_buf(void \*apu\_handle, struct fr\_buf\_info\* buf)  
int \_apu\_fr\_put\_buf(struct fr\_buf\_info\* buf)

### 3.3 子模块定义

audiobox\_v2定义若干音频流处理子模块来实现上述的基本需求：

|  |
| --- |
| 子模块 |
| softvol |
| recombine |
| codec |
| AEC |
| dispatch |
| chan-remap |
| capture |
| playback |

现在通道参数可以与设备参数不一致，上述的子模块可以将设备参数的音频流转换为通道格式的音频流。对于不同线程中申请的通道参数相同的多个录音通道，可以共享各个子模块以节省系统资源，音频流可以从同一个codec\_fr被dispatch不同通道的chan\_fr

|  |
| --- |
| ### 3.4 AEC功能 **AEC2.0实现** |
| 对于录音通道，如果没有使能AEC2.0，调用audio\_hal\_read()函数从ALSA读取到dev\_fr的原始音频流为"MIC左声道+MIC右声道"的2声道模式；如果audiobox使能AEC2.0，那么audiobox启动时会调用audiobox\_set\_aec\_version()函数通知内核，之后调用audio\_hal\_read()函数从ALSA读取到dev\_fr的原始音频流将为"MIC左声道+MIC右声道+SPK左声道+SPK右声道"的4声道模式，增加的2个SPK声道为本地放音的环回，通过驱动层的控制，保证SPK和MIC之间的相对延时满足AEC处理的要求 |
| 使能AEC2.0时，非AEC通道从4声道格式的原始音频流中提取"MIC左声道+MIC右声道"，AEC通道从4声道格式的原始音频流中提取"MIC左声道+SPK左声道" |
| **多个aec算法库的管理** |
| AEC模块可以通过aec\_lib\_register()函数注册当前使用哪一个aec-lib，当前提供了基于arm的speexdsp和基于dsp的alango两种aec算法支持 |
| ## 4 通道申请释放流程 上层应用调用audio\_get\_channel()函数创建通道，调用audio\_put\_channel()函数释放通道，本章通过图例描述录音和放音通道的基本流程 |
| ### 4.1 录音通道申请和释放 本节通过图例描述下面场景的流程：申请两个相同通道参数的录音通道，然后依次释放。基于此场景来解析录音通道申请流程，以及同通道参数的两个通道如何共享codec子模块 |
| **申请第一个录音通道** |
|  |
| **申请第二个录音通道，同通道参数** |
|  |
| NOTE: 第二录音通道和第一录音通道参数相同，共享之前创建codec即可，无需新建 |
| **释放第二个录音通道** |
|  |
| NOTE: 此时只释放通道，不释放共享的codec和音频设备 |
| **释放第一个录音通道** |
|  |
| NOTE: 此时释放通道、共享的codec和音频设备 |
| ### 4.2 放音通道申请和释放 本节通过图例描述下面场景的流程：申请两个相同通道参数的放音通道，然后依次释放 |
| **申请第一个放音通道** |
|  |
| **申请第二个放音通道** |
|  |
| NOTE: 放音通道即使通道参数相同，也无法共享codec |
| **释放第二个放音通道** |
|  |
| NOTE: 此时只释放通道和codec，不释放音频设备 |
| **释放第一个放音通道** |
|  |
| NOTE: 此时释放通道、codec和音频设备 |
| NOTE: 因为不可能一个以上放音通道同时放音，所以设备所有放音通道共享同一个thread\_decoder\_workloop()线程，处理当前处于激活状态的通道的音频流。只在设备启动时启动线程，设备停止时退出线程 |

## 5 音频数据流

本章以流程图显示几种典型的通道音频数据流走向。softvol、recombine、codec三个子模块处理在实际简单点的通道配置中，可能会被跳过，这里不再绘图描述

### 5.1 aec使能的录音流程

audiobox从alsa buffer中读取录音数据，处理后分发到音频通道的chan\_fr，上层app调用audio\_read\_frame()函数从chan\_fr读取数据流 ### 5.2 aec禁止的录音流程(AEC2.0)

audiobox从alsa buffer中读取录音数据，处理后分发到音频通道的chan\_fr，上层app调用audio\_read\_frame()函数从chan\_fr读取数据流 ### 5.3 aec禁止的录音流程(no AEC2.0)

audiobox从alsa buffer中读取录音数据，处理后分发到音频通道的chan\_fr，上层app调用audio\_read\_frame()函数从chan\_fr读取数据流 ### 5.4 放音流程

上层app调用audio\_write\_frame()函数，将音频流数据写入chan\_fr，经过处理最终进入alsa buffer被硬件播放