密级状态：绝密(　)　秘密(　)　内部(　)　公开(√)

RK3399_BOX 双屏异显音频说明

（技术部，第二系统产品部）

文件状态： [ ] 正在修改 [√] 正式发布	当前版本：	V1.1
	作　者：	郑应航、刘兴亮
	完成日期：	2017-12-20
	审　核：	
	完成日期：	

福州瑞芯微电子股份有限公司

Fuzhou Rockchip Electronics Co . , Ltd

版本历史

版本号	作者	修改日期	修改说明	备注
V1.0	刘兴亮	2017.10.23	发布初始版本	
V1.1	郑应航	2017.12.20	增加相关说明	

目 录

1 简介

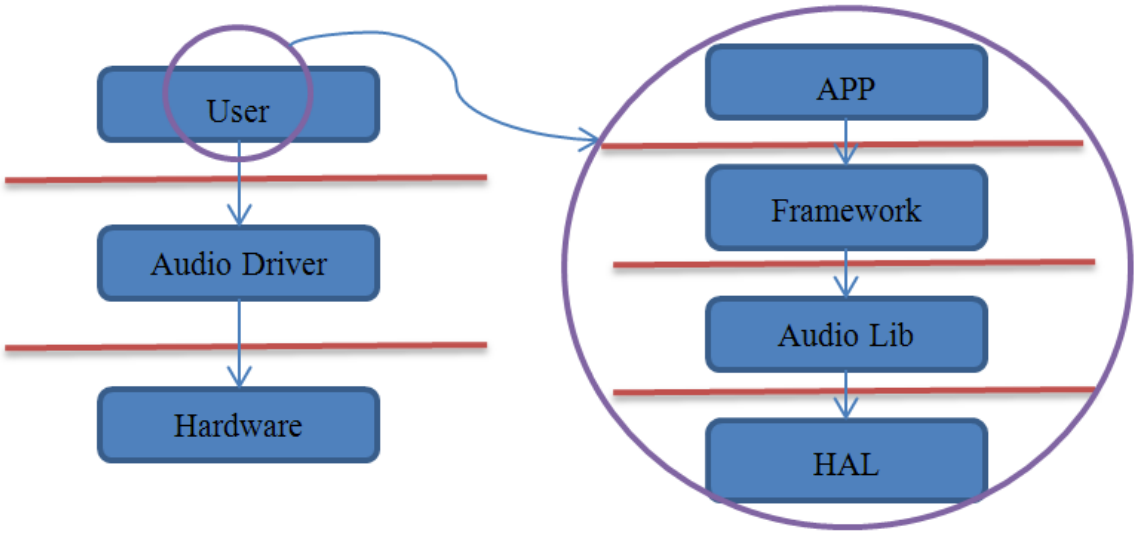
本文主要介绍了 RockChip(以下简称 RK) SDK 平台上支持的异显音频方案，包括系统原理、补丁以及配置方法等，适用于以下 SDK 平台：

- RK 3399_BOX

2 系统原理

关于双屏异显两路音频目前还没有通用的方法，音频方案思路是借鉴 A2DP（蓝牙音频传输协议）；A2DP 的场景是：铃声 触摸声等系统声音从 speaker 直接输出，音乐通过 bt 输出；这种模式和异显的需求是类似的，异显要求主屏的声音从主屏对应的声卡输出、副屏的声音从副屏对应声卡输出，不能有混音。

2.1 Android 系统音频框架



整个框架包括应用层、framework 层、lib 层、hal 层、驱动以及硬件。

- APP

这是整个音频体系的最上层，因此并不是 Android 系统实现异显两路音频输出的重点。比如厂商根据特定需求自己写的一个音乐播放器，游戏中使用到声音，或者调节音频的一类软件等等。

- Framework

Android 提供了两个功能类，AudioTrack 和 AudioRecorder；除此以外，Android 系统还为我们控制音频系统提供了 AudioManager、AudioService 及 AudioSystem 类。这些都是 framework 为便利上层应用开发所设计的。

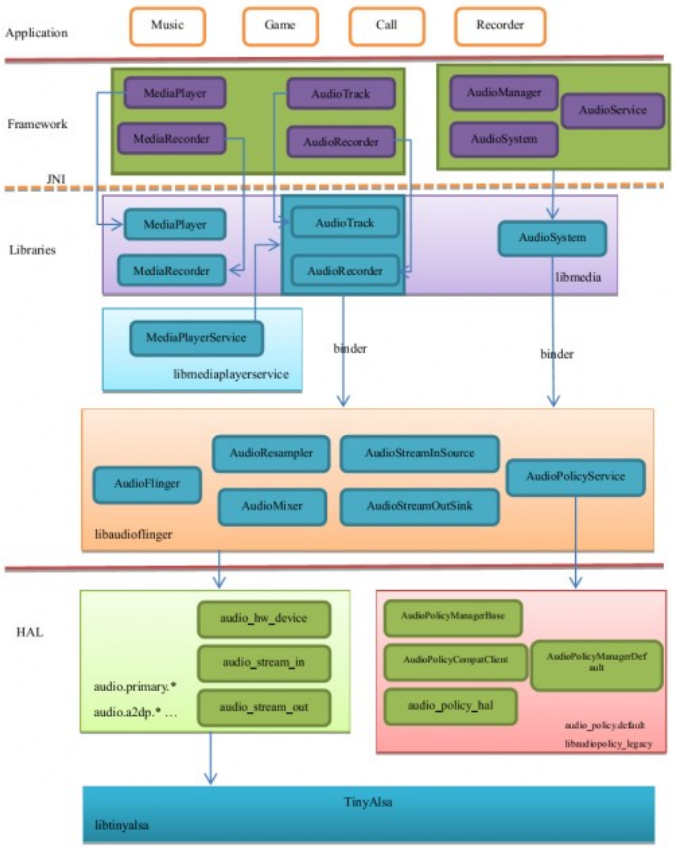
- Libraries

Framework 层的很多类，实际上只是应用程序使用 Android 库文件的"中介"而已。因为上层应用采用 java 语言编写，它们需要最直接的 java 接口的支持，这就是 framework 层存在的意义之一。而作为"中介"，它们并不会真正去实现具体的功能，或者只实现其中的一部分功能，而把主要重心放在库中来完成。比如上面的 AudioTrack、AudioRecorder 等等在库中都能找到相对应的类，这些库多数是 C++语言编写的。除了上面的类库实现外，音频系统还需要一个"核心中控"，或者用 Android 中通用的实现来讲，需要一个系统服务，这就是 AudioFlinger 和 AudioPolicyService。

- HAL

从设计上来看，硬件抽象层是 AudioFlinger 直接访问的对象。这说明了两个问题，一方面 AudioFlinger 并不直接调用底层的驱动程序；另一方面，AudioFlinge 上层(包括和它同一层的 MediaPlayerService)的模块只需要与它进行交互就可以实现音频相关的功能了。因而我们可以认为 AudioFlinger 是 Android 音频系统中真正的"隔离板"，无论下面如何变化，上层的实现都可以保持兼容。音频方面的硬件抽象层主要分为两部分，即 AudioFlinger 和 AudioPolicyService。实际上后者并不是一个真实的设备，只是采用虚拟设备的方式来让厂商可以方便地定制出自己的策略。抽象层的任务是将 AudioFlinger/AudioPolicyService 真正地与硬件设备关联起来，但又必须提供灵活的结构来应对变化——特别是对于 Android 这个更新相当频繁的系统。比如以前 Android 系统中的 Audio 系统依赖于 alsa-lib，但后期就变为了 tinyalsa，这样的转变不应该对上层造成破坏。因而 Audio HAL 提供了统一的接口来定义它与 AudioFlinger/AudioPolicyService 之间的通信方式，这就是 audio_hw_device、audio_stream_in 及 audio_stream_out 等等存在的目的，这些 Struct 数据类型内部大多只是函数指针的定义，是一些"壳"。当 AudioFlinger/AudioPolicyService 初始化时，它们会去寻找系统中最匹配的实现(这些实现驻留在以 audio.primary.*,audio.a2dp.*为名的各种库中)来填充这些"壳"。根据产品的不同，音频设备存在很大差异，在 Android 的音频架构中，这些问题都是由 HAL 层的 audio.primary 等等库来解决的，而不需要大规模地修改上层实现。换句

话说，厂商在定制时的重点就是如何提供这部分库的高效实现了。



2.2 异显音频方案

2.2.1 双路触发

在 PhoneWindow.java 的 superDispatchKeyEvent 里面包含异显触发事件，在里面添加 force_speaker 广播，并设置 media.audio.device_policy.db 的属性，异显状态下设置为“speaker”，否则设置为“hdmi”。

```
try {
    if (WindowManagerHolder.sWindowManager.isShowDualScreen()) {
        trigerSyncScreen();
        Log.i("DualScreen","====== is ctrl down false======sethdmi======");
        Intent broadcast = new Intent("com.android.server.input.force_speaker");
        broadcast.putExtra("force_speaker", false);
        getContext().sendBroadcast(broadcast);
        SystemProperties.set("media.audio.device_policy.db", "hdmi");
    } else {
        trigerDualScreen();
        Log.i("DualScreen","====== is ctrl down true======setspeaker======");
        Intent broadcast = new Intent("com.android.server.input.force_speaker");
        broadcast.putExtra("force_speaker", true);
        getContext().sendBroadcast(broadcast);
        SystemProperties.set("media.audio.device_policy.db", "speaker");
    }
```

在 InputManagerService.java 的 start 里面接收广播，收到广播后，触发耳机线控事件。

```
mContext.registerReceiver(new BroadcastReceiver() {
    @Override
    public void onReceive(Context context, Intent intent) {
        boolean isForceSpeaker = intent.getBooleanExtra("force_speaker", true);
        Slog.i(TAG,"======get broadcast notifyWiredAccessoryChanged isForceSpeaker:"+isForceSpeaker);
        if(mWiredAccessoryCallbacks != null && isForceSpeaker) {
            Slog.i(TAG,"======get broadcast notifyWiredAccessoryChanged true");
            mWiredAccessoryCallbacks.notifyWiredAccessoryChanged(0,1,0x400);
        } else if (mWiredAccessoryCallbacks != null && !isForceSpeaker) {
            Slog.i(TAG,"get broadcast notifyWiredAccessoryChanged false");
            mWiredAccessoryCallbacks.notifyWiredAccessoryChanged(0,0,0x400);
        }
    }
}, new IntentFilter("com.android.server.input.force_speaker"), null, mHandler);
```

2.2.2 切换策略

系统默认从 hdmi 输出，同屏时声音也从 hdmi 输出；异显时，副屏的声音从 speaker 输出，主屏的声音从 hdmi 输出。

首先获取副屏上 activity 的 pid，并设置为“media.audio.activity.pid”属性的值，同屏时，属性值设置为 -1 。

在 moveTransitionToSecondDisplay 中添加

SystemProperties.set("media.audio.activity.pid", String.valueOf(win.mSession.mPid));

在 updateDisplayShowSynchronization 中添加

SystemProperties.set("media.audio.activity.pid", String.valueOf(-1));

返回 AUDIO_DEVICE_OUT_WIRED_HEADSET 表示声音最终从 speaker 输出；

返回 AUDIO_DEVICE_OUT_AUX_DIGITAL 表示声音最终从 hdmi 输出。

2.2.3 循环播放策略

测试发现，当副屏当前视频播放结束并开始播放视频时，声音会跑到主屏，即 speaker 的声音跑到 hdmi 。软件的流程是对的，因为此时 media.audio.device_policy.db 的属性值时 hdmi，循环相关的策略还没有完成。

Android 音频播放之前会先调用 MediaFocusControl.java 中的 requestAudioFocus，在其中获取之前 media.audio.activity.pid 的属性值，进行策略安排。如下：

```
int pidToUse = Integer.parseInt(SystemProperties.get("media.audio.activity.pid"));
Log.i(TAG,"======pidToUse:" + pidToUse);
if(pidToUse > 0){
   if(pidToUse == Binder.getCallingPid()){
        Log.i(TAG, "======setspeaker======");
        SystemProperties.set("media.audio.device_policy.db", "speaker");
    }else{
        Log.i(TAG, "======sethdmi======");
        SystemProperties.set("media.audio.device_policy.db", "hdmi");
    }
}else{
    Log.i(TAG, "======OutOfDualScreen=====setspeaker======");
    SystemProperties.set("media.audio.device_policy.db", "speaker");
}
```

3 HAL 配置

3.1 audio_policy.conf 修改

文件路径：/device/rockchip/common/audio_policy_rk30board.conf

修改的内容如下：

- 更改全局配置里的输出设备（默认输出设备）

  由 AUDIO_DEVICE_OUT_SPEAKER 改为 AUDIO_DEVICE_OUT_AUX_DIGITAL

```
global_configuration {
  attached_output_devices AUDIO_DEVICE_OUT_AUX_DIGITAL
  default_output_device AUDIO_DEVICE_OUT_AUX_DIGITAL
  attached_input_devices AUDIO_DEVICE_IN_BUILTIN_MIC|AUDIO_DEVICE_IN_REMOTE_SUBMIX
}
```

- 更改 primary 默认输出设备

```
primary {
  outputs {
    primary {
      sampling_rates  44100|48000
      channel_masks AUDIO_CHANNEL_OUT_STEREO
      formats AUDIO_FORMAT_PCM_16_BIT
      devices AUDIO_DEVICE_OUT_AUX_DIGITAL|AUDIO_DEVICE_OUT_ALL_SCO|AUDIO_DEVICE_OUT_SPDIF
      flags AUDIO_OUTPUT_FLAG_PRIMARY
    }
```

- 添加 dgtl 输出

```
dgtl {
  outputs{
      dgtl{
        sampling_rates  44100|48000
        channel_masks AUDIO_CHANNEL_OUT_STEREO
        formats AUDIO_FORMAT_PCM_16_BIT
        devices AUDIO_DEVICE_OUT_EARPIECE|AUDIO_DEVICE_OUT_SPEAKER|AUDIO_DEVICE_OUT_WIRED_HEADSET
      }
  }
}
a2dp {
```

3.2 增加 dgtl 库

复制 hardware/rockchip/audio/tinyalsa_hal 到 hardware/rockchip/audio/tinyalsa_hal_dgtl，并修改 Android.mk，如下：

```
LOCAL_PATH := $(call my-dir)

include $(CLEAR_VARS)

LOCAL_MODULE := audio.dgtl.$(TARGET_BOARD_HARDWARE)
```

将 audio_hw.c 中 connect_hdmi 的值改为 false，屏蔽 hdmi。

```
    connect_hdmi = false;
    route_pcm_open(getRouteFromDevice(out->device));

    if (out->device & AUDIO_DEVICE_OUT_AUX_DIGITAL) {
        if (connect_hdmi) {
#ifdef BOX_HAL
#ifdef RK3399
            int ret = 0;
            ret = mixer_mode_set(out);

            if (ret!=0) {
                ALOGE("mixer mode set error,ret=%d!",ret);
            }
#endif
```

编译生成 audio.dgtl.rk30board.so

最终，如果输出要求是 hdmi，AudioFlinger 会调用 audio.dgtl.primary.so；如果输出要求是

speaker，AudioFlinger 会调用 audio.dgtl.rk30board.so。

4 补丁

在 framework/base/下打上补丁 Dual_Audio_framework_base.patch

在 framework/av/下打上补丁 Dual_audio_framework_av.patch

在 hardware/libhardware/下打上补丁 Dual_audio_hardware_libhardware.patch