密级状态：绝密（　　）　秘密（　　）　内部（　　）　公开（ √ ）

RK 音频简介以及常见问题 debug 方法

（技术部）

文件状态： [　] 正在修改 [√] 正式发布	当前版本：	V1.0
	作　　者：	罗肖谭
	完成日期：	2015-10-13
	审　　核：	
	完成日期：	

福州瑞芯微电子有限公司

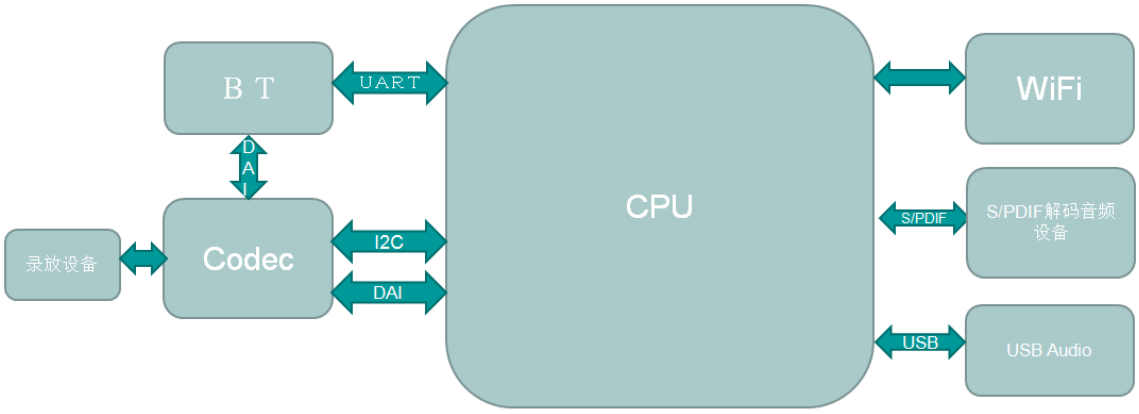
Fuzhou　Rockchips　Semiconductor　Co . , Ltd

版 本 历 史

版本号	作者	修改日期	修改说明	备注

目 录

音频系统基本硬件电路



音频编解码器 Codec 负责处理音频信息，包括 ADC,DAC,Mixer,DSP,输入输出以及音量控制等所有与音频相关的功能。

Codec 与处理器之间通过 I2C 总线和数字音频接口 DAI 进行通信。

I2C 总线 - 实现对 Codec 寄存器数据的读写。

DAI –（digital audio interface）实现音频数据在 CPU 和 Codec 间的通信，包括 I2S、PCM 和 AC97 等。

蓝牙立体声音乐播放走的是蓝牙跟 CPU 直接的 UART 接口。

在只有一组 I2S 的主控上面，蓝牙通话 SCO 暂时是通过 Codec 中继，即通过 codec 来路由选择是否走通话，走的是 I2S 接口 。如果主控有两组 I2S，那么可以将 BT PCM 直接接到主控的 PCM 接口上。

S/DIF（Sony/Philips Digital Interface Format），通过光纤或同轴电缆传输音频，保证音频质量。

Codec 内部通路举例 RK616

12.2 Block Diagram

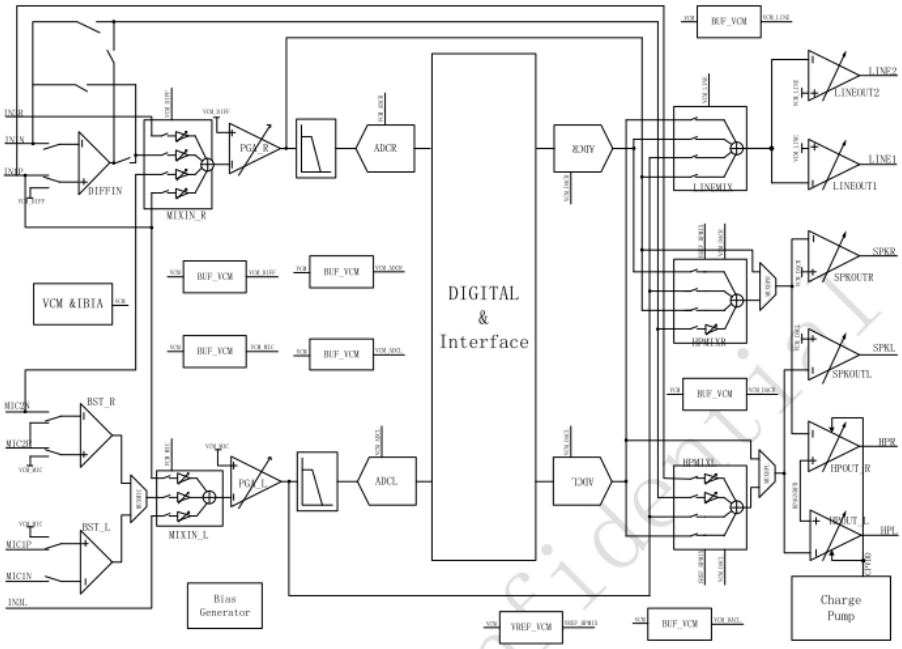


Fig. 12-1 Audio Codec Block Diagram

录音通路 MIC1N/P ->BST-L->MUXMIC-> MIXIN_L->PGAL->ADCL->I2S SDI->主控

放音通路 主控-> I2S SDO->DACL->HPMIX->SPKL->喇叭

I2S 信号如下

主从模式：

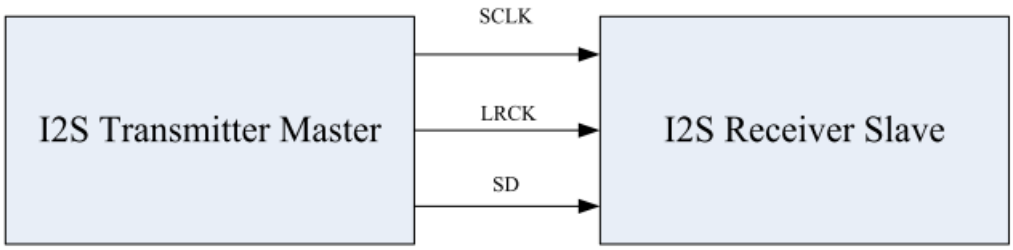


Fig. 17-2 I2S transmitter-master & receiver-slave condition

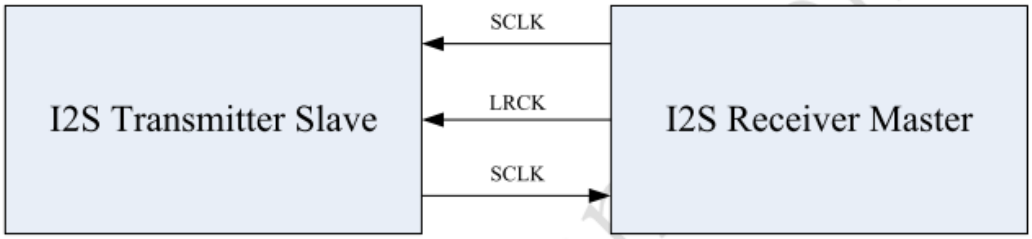


Fig. 17-3 I2S transmitter-slave& receiver-master condition

下图描述的是 RK 8channel 的 i2s 时序格式，一般使用 I2S0_SDO0 I2S0_SDI0 作为一组引出到外部 codec 使用形成左右声道的输入输出立体声效果。

关于 7.1 声道：

7.1 声道输出，是需要使用 I2S0_SDO0~I2S0_SDO3 共 4 组数据线组成 8 声道的音频数据，这里主要给 HDMI 用形成 7.1 声道输出，同时 7.1 声道的话还需要播放器解码器音频源等支持，目前在 BOX SDK 上面可以通过透传的方式支持 HDMI 输出 7.1 声道，MID 不支持。

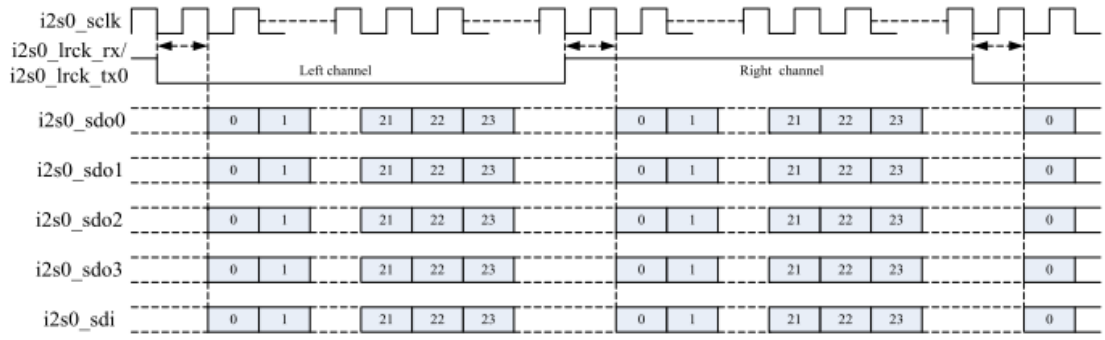


Fig. 17-4 I2S normal mode timing format

关于采样率和采样深度

目前我们 MID Android 系统的支持的最大采样率是 48K 16bit，这个主要是限制是在 android framework 包括解码器和 audio flinger 等， linux 软件驱动是可以支持 192K 32bit 等高采样率高精度的音频格式。

关于 HIFI 播放器 （高采样率高精度 一般要支持 192K 24 BIT WAV FLAC 等无损播放）

方案一

3188 4.2 的 SDK 上面有做过 192K 24bit 的 HIFI 播放器，主要是更改系统的解码器 audio flinger HAL 层以及 linux sound card 驱动， 优点是使用系统的 api 即可播放 hifi

音频，缺点是 SDK 到 4.4 更新之后无法再使用。

另外一种方案是

不使用系统提供的 api 接口，可以自己实现 hifi 播放器，使用 ffmpeg 解码器，将 ffmpeg 解码出来的 PCM 数据通过 alsa 接口写到声卡实现 HiFi 播放，当然这个也需要声卡驱动的支持。优点是 SDK 升级之后还可以使用，因为是不依赖于 android 系统的 api。

Sound card 驱动配置相关代码

Dts

Dts 里面主要是一些资源的配置，例如

```
161         rockchip-es8323{
162                 compatible = "rockchip-es8323";
163                 dais {
164                         dai0 {
165                                 audio-codec = <&es8323>;
166                                 i2s-controller = <&i2s>;
167                                 format = "i2s";
168                                 //continuous-clock;
169                                 //bitclock-inversion;
170                                 //frame-inversion;
171                                 //bitclock-master;
172                                 //frame-master;
173                         };
174                 };
175         };


553 &i2c2 {
554         status = "okay";
555         rt5631: rt5631@1a {
556                 compatible = "rt5631";
557                 reg = <0x1a>;
558         };
559         es8323: es8323@10 {
560                 compatible = "es8323";
```

```
561                 reg = <0x10>;
562         };
```

Sound

Config

ALSA 驱动目录结构

```
lxt@rksz-server101:~/rk312x-sdk-4.4.4/kernel/sound$ tree -L 1
├── ac97_bus.c
├── aoa
├── arm
├── atmel
├── built-in.mod.c
├── built-in.o
├── core
├── drivers
├── firewire
├── i2c
├── isa
├── Kconfig
├── last.c
├── last.o
├── Makefile
├── mips
├── oss
├── parisc
├── pci
├── pcmcia
├── ppc
├── sh
├── soc
├── sound_core.c
├── soundcore.mod.c
├── sound_core.o
├── soundcore.o
├── sound_firmware.c
├── sparc
├── spi
├── synth
```

```
└─── usb

20 directories, 13 files
```

嵌入式设备的音频系统可以被划分为板载硬件（Machine）、Soc（Platform）、Codec 三大部分

同时音频驱动有三部分：

Machine driver

sound/soc/rockchip/rk_rk616.c

其中的 Machine 驱动负责 Platform 和 Codec 之间的耦合以及部分和设备或板子特定的代码采样率时钟配置

Platform driver

sound/soc/ rockchip /rk30_i2s.c

I2S 控制器驱动 采样率时钟 DMA 等配置

Codec driver

sound/soc/codecs/rk616_codec.c

codec 的寄存器通路的配置

想要深入了解 alsa 驱动最后的办法是阅读 kernel 代码

这个 blog 写的不错 http://blog.csdn.net/droidphone/article/details/6271122

alsa HAL 层

android 5.1 BOX MID 的 SDK 之后统一使用这个目录下面的代码

\hardware\rockchip\audio\tinyalsa_hal

4.4 MID HAL 代码路径

hardware\rk29\audio

4.4 BOX HAL 代码路径

hardware\alsa_sound

android audio route 通路介绍

android 定义了很多的音频 devices

如 AudioSystem .java 中：

```
211     // output devices, be sure to update AudioManager.java also
212     public static final int DEVICE_OUT_EARPIECE = 0x1;
213     public static final int DEVICE_OUT_SPEAKER = 0x2;
214     public static final int DEVICE_OUT_WIRED_HEADSET = 0x4;
215     public static final int DEVICE_OUT_WIRED_HEADPHONE = 0x8;
216     public static final int DEVICE_OUT_BLUETOOTH_SCO = 0x10;
217     public static final int DEVICE_OUT_BLUETOOTH_SCO_HEADSET = 0x20;
218     public static final int DEVICE_OUT_BLUETOOTH_SCO_CARKIT = 0x40;
219     public static final int DEVICE_OUT_BLUETOOTH_A2DP = 0x80;
220     public static final int DEVICE_OUT_BLUETOOTH_A2DP_HEADPHONES = 0x100;
221     public static final int DEVICE_OUT_BLUETOOTH_A2DP_SPEAKER = 0x200;
222     public static final int DEVICE_OUT_AUX_DIGITAL = 0x400;
223     public static final int DEVICE_OUT_ANLG_DOCK_HEADSET = 0x800;
224     public static final int DEVICE_OUT_DGTL_DOCK_HEADSET = 0x1000;
225     public static final int DEVICE_OUT_USB_ACCESSORY = 0x2000;
226     public static final int DEVICE_OUT_USB_DEVICE = 0x4000;
227     public static final int DEVICE_OUT_REMOTE_SUBMIX = 0x8000;
```

HAL 层通过一定转换跟 android 上层 对应

在 audio/alsa_route.c 中通过

route_set_controls(route)中通过 get_route_config(int route)将以上的各个 route 值转换为 codec_config 中的 xxx.h 配置文件

```
const struct config_route *get_route_config(unsigned route)
{
    ALOGV("get_route_config() route %d", route);

    if (!route_table) {
        ALOGE("get_route_config() route_table is NULL!");
```

```
        return NULL;
    }
    switch (route) {
    case SPEAKER_NORMAL_ROUTE:
        return &(route_table->speaker_normal);
    case SPEAKER_INCALL_ROUTE:
        return &(route_table->speaker_incall);
    case SPEAKER_RINGTONE_ROUTE:
        return &(route_table->speaker_ringtone);
    case SPEAKER_VOIP_ROUTE:
        return &(route_table->speaker_voip);
    case EARPIECE_NORMAL_ROUTE:
        return &(route_table->earpiece_normal);
    case EARPIECE_INCALL_ROUTE:
        return &(route_table->earpiece_incall);
    case EARPIECE_RINGTONE_ROUTE:
        return &(route_table->earpiece_ringtone);
    case EARPIECE_VOIP_ROUTE:
        return &(route_table->earpiece_voip);
    case HEADPHONE_NORMAL_ROUTE:
        return &(route_table->headphone_normal);
    case HEADPHONE_INCALL_ROUTE:
        return &(route_table->headphone_incall);
case HEADPHONE_RINGTONE_ROUTE:
……
}
}
```

HAL 层中定义的 route table，在通过 route_set_controls 将这些 table 配置，最终设置到 codec 的寄存器

```
struct config_route_table
{
    const struct config_route speaker_normal;
    const struct config_route speaker_incall;
    const struct config_route speaker_ringtone;
    const struct config_route speaker_voip;

    const struct config_route earpiece_normal;
    const struct config_route earpiece_incall;
```

```
    const struct config_route earpiece_ringtone;
    const struct config_route earpiece_voip;

    const struct config_route headphone_normal;
    const struct config_route headphone_incall;
    const struct config_route headphone_ringtone;
    const struct config_route speaker_headphone_normal;
    const struct config_route speaker_headphone_ringtone;
    const struct config_route headphone_voip;

    const struct config_route headset_normal;
    const struct config_route headset_incall;
    const struct config_route headset_ringtone;
    const struct config_route headset_voip;

    const struct config_route bluetooth_normal;
    const struct config_route bluetooth_incall;
    const struct config_route bluetooth_voip;

    const struct config_route main_mic_capture;
    const struct config_route hands_free_mic_capture;
    const struct config_route bluetooth_sco_mic_capture;

    const struct config_route playback_off;
    const struct config_route capture_off;
    const struct config_route incall_off;
    const struct config_route voip_off;

    const struct config_route hdmi_normal;

    const struct config_route usb_normal;
    const struct config_route usb_capture;

    const struct config_route spdif_normal;
};
```

HAL 层会根据 sound card name 选择使用哪个 table，如果没有匹配默认使用 default_config.h

```
struct alsa_sound_card_config sound_card_config_list[] = {
    {
        .sound_card_name = "RKRK616",
```

```
        .route_table = &rk616_config_table,
    },
    {
        .sound_card_name = "RK29RT3224",
        .route_table = &rt3224_config_table,
    },
    {
        .sound_card_name = "RK29RT3261",
        .route_table = &rt3261_config_table,
    },
    {
        .sound_card_name = "RK29WM8960",
        .route_table = &wm8960_config_table,
    },
    {
        .sound_card_name = "RKRT3224",
        .route_table = &rt3224_config_table,
    },
```

RK codec 芯片使用 default_config.h

如下这种 tinyalsa 形式的代码：

参数：

```
static const char *rk616_playback_path_mode[] = {
        "OFF", "RCV", "SPK", "HP", "HP_NO_MIC", "BT", "SPK_HP", //0-6
        "RING_SPK", "RING_HP", "RING_HP_NO_MIC", "RING_SPK_HP"};//7-10
```

Playback Path 对应的值有 11 个，分别对应：

关闭、听筒（NORMAL 模式）、喇叭（NORMAL 模式）、带 MIC 耳机（NORMAL 模式）、不带 MIC 耳机（NORMAL 模式）、蓝牙（NORMAL 模式）、喇叭与耳机（NORMAL 模式）、喇叭（RINGTONE 模式）、带 MIC 耳机（RINGTONE 模式）、不带 MIC 耳机（RINGTONE 模式）、喇叭与耳机（RINGTONE 模式）

当 HAL 层调用设置 Playback Path 对应的 control 是，ALSA 会调用注册时对应的 put 函数，即 rk616_playback_path_put 函数。传入的值为 int 类型，与 controls 注册时的字符串顺序一一对应。如 HAL 层调用‘OFF’，那么调用 put 函数传入的值就为 0，而‘SPK’对应的就是 2。

对于 playback 来说：

（1）、SPK_PATH 与 RING_SPK 相同都为喇叭播放音乐通路

（2）、HP_PATH、HP_NO_MIC、RING_HP、RING_HP_NO_MIC 相同，都为耳机播放音乐通路

（3）、SPK_HP、RING_SPK_HP 相同，都为耳机和喇叭同时播放音乐通路

因此处理参数上有些就可以直接一起归类。

下面简单说明 Playback Path 的 put 函数里需要做的事情：

```
static int rk616_playback_path_put(struct snd_kcontrol *kcontrol,
        struct snd_ctl_elem_value *ucontrol)
{
    switch (ucontrol->value.integer.value[0]) {
    case OFF:
       关闭播放通路包括喇叭和耳机
        break;
    case RCV:
       原先 tinyalsa 关闭 incall 时会调用，HAL 层修改后，现在不会调用。
        break;
    case SPK_PATH:
    case RING_SPK:
       开启喇叭播放音乐通路
        break;
    case HP_PATH:
    case HP_NO_MIC:
    case RING_HP:
    case RING_HP_NO_MIC:
       开启耳机播放音乐通路
        break;
    case BT:
       蓝牙播放音乐通路不会调用 Playback Path，这边不做处理
        break;
    case SPK_HP:
    case RING_SPK_HP:
       开启耳机、喇叭同时播放音乐通路
        break;
    default:
        return -EINVAL;
    }
    return 0;
}
```

如何 debug

第一步首先看 声卡有没有注册

通过看 kernel log 确认 alsa sound card 有没有注册

```
<6>[    2.713102] Enter::rk29_rt3261_init----218
<6>[    2.721654] asoc: rt5639-aif1 <-> rk29_i2s.1 mapping ok
<6>[    2.728488] asoc: rt5639-aif2 <-> rk29_i2s.1 mapping ok
<6>[    2.729289] ALSA device list:
<6>[    2.729318]   #0: RK29_RT5639
```

声卡没有注册参考《Audio 部分常见问题处理方法》声卡注册问题 debug。

第二步 确认 route 是否正常

声卡注册好之后确认 alsa route 是否正确

通过命令 logcat -s alsa_route

```
root@HCTT1:/ # logcat -s alsa_route
logcat -s alsa_route
--------- beginning of /dev/log/system
--------- beginning of /dev/log/main
D/alsa_route(   90): route_info->sound_card 0, route_info->devices 0
E/alsa_route(   90): route_pcm_open() DEVICES_0
D/alsa_route(   90): route_set_controls() set route 0
E/alsa_route(   90): route_pcm_close()route 24
D/alsa_route(   90): route_set_controls() set route 24
```

上面命令跑了两个路由 route 0 route 24 各个 route 表示的意义

在 hardware\rk29\audio \alsa_audio.h 中定义，表示打开了一次 speaker 后面又关闭了一次 speaker

```
typedef enum _AudioRoute {
    SPEAKER_NORMAL_ROUTE = 0,
    SPEAKER_INCALL_ROUTE, // 1
    SPEAKER_RINGTONE_ROUTE,
    SPEAKER_VOIP_ROUTE,

    EARPIECE_NORMAL_ROUTE, // 4
    EARPIECE_INCALL_ROUTE,
```

```
    EARPIECE_RINGTONE_ROUTE,
    EARPIECE_VOIP_ROUTE,

    HEADPHONE_NORMAL_ROUTE, // 8
    HEADPHONE_INCALL_ROUTE,
    HEADPHONE_RINGTONE_ROUTE,
    SPEAKER_HEADPHONE_NORMAL_ROUTE,
    SPEAKER_HEADPHONE_RINGTONE_ROUTE,
    HEADPHONE_VOIP_ROUTE,

    HEADSET_NORMAL_ROUTE, // 14
    HEADSET_INCALL_ROUTE,
    HEADSET_RINGTONE_ROUTE,
    HEADSET_VOIP_ROUTE,

    BLUETOOTH_NORMAL_ROUTE, // 18
    BLUETOOTH_INCALL_ROUTE,
    BLUETOOTH_VOIP_ROUTE,

    MAIN_MIC_CAPTURE_ROUTE, // 21
    HANDS_FREE_MIC_CAPTURE_ROUTE,
    BLUETOOTH_SOC_MIC_CAPTURE_ROUTE,

    PLAYBACK_OFF_ROUTE, // 24
    CAPTURE_OFF_ROUTE,
    INCALL_OFF_ROUTE,
    VOIP_OFF_ROUTE,

    HDMI_NORMAL_ROUTE, // 28

    USB_NORMAL_ROUTE, // 29
    USB_CAPTURE_ROUTE,

    MAX_ROUTE, // 31
} AudioRoute;
```

一般 route 的错误

1)、 耳机喇叭切换 有误 这个时候需要确认耳机检测是否正常

通过 cat sys/class/switch/h2w/state 查看耳机插入状态：

```
root@HCTT1:/sys/class/switch/h2w # cat state
cat state
0
```

state <= 0 表示无耳机插入

state = 1 表示带 Mic 耳机插入

state = 2 表示不带 Mic 耳机插入

如果耳机状态不对则需要确认耳机检测是否正常，首先要确认耳机检测的 GPIO 拔插时候电平是否有高点变化。如果没有插需要进行硬件排查。如果有则需要参考《RK android 带 MIC 耳机检测以及 hook-kernel.pdf》文档中确认 拔插耳机是否有中断。

如果没有则 需要确认 GPIO 配置是否正确，

board.c (3.10 之前的 kernel)参考《Audio 部分常见问题处理方法》处理。

3.10 之后需要 确认 dts 中的 rockchip_headset 配置是否正确

```
666 &adc {
 667         status = "okay";
 668
 669         rockchip_headset {
 670                 compatible = "rockchip_headset";
 671                 headset_gpio = <&gpio7 GPIO_A7 GPIO_ACTIVE_LOW>;
 672                 pinctrl-names = "default";
 673                 pinctrl-0 = <&gpio7_a7>;
 674                 io-channels = <&adc 2>;
 675         /*
 676                 hook_gpio = ;
 677                 hook_down_type = ; //interrupt hook key down status
 678                 */
 679         };
 680
```

三段 四段耳机的区分 有两种方案 一种是通过 adc 另外一种是通过 GPIO 区分是 3 段耳机还是 4 段耳机插入，详情参考《RK android 带 MIC 耳机检测以及 hook-kernel.pdf》、《android 耳机监听路由切换-framework-hal.pdf》

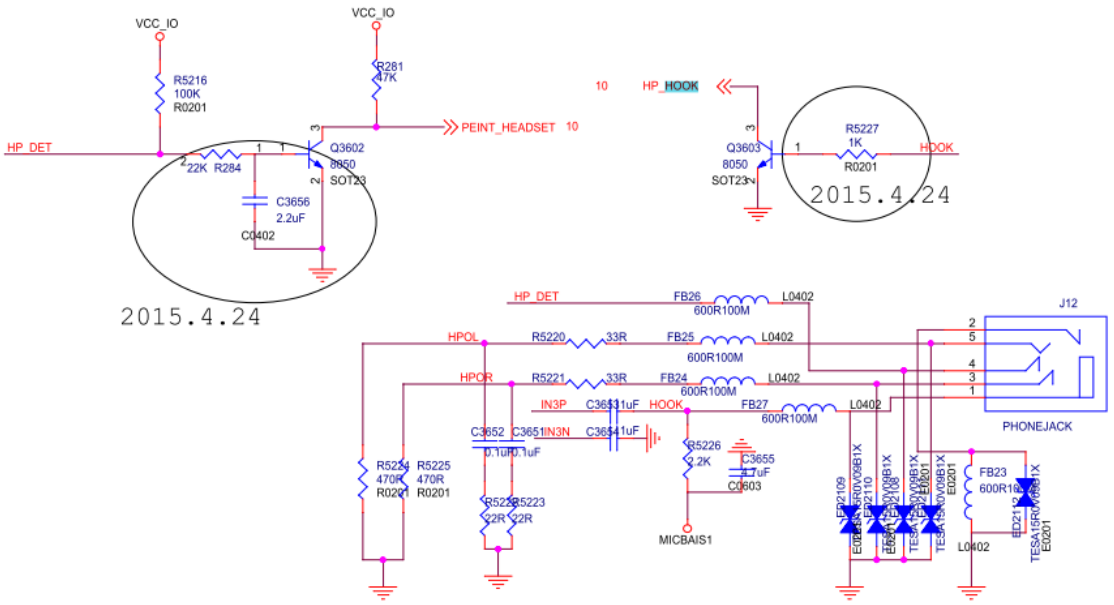
通过 adc 来区分 dts 中配置

```
io-channels = <&adc 2>;
```

通过 GPIO 来区分

```
676                hook_gpio = ;
677                hook_down_type = ; //interrupt hook key down status
```

如下所示是使用 GPIO 来区分 3,4 段耳机的插入，耳机上按键按下获取松开 gpio 电平有高低变化系统可以通过这个变化 向系统上报按键 默认 media。



EARPHONE

2)、多声卡切换

系统默认支持 codec sound card card0

Hdmi(spdif) 为 card 1 插入 HDMI 时候系统会自动切换到 card1 可以通过 route 来确认 route =28 插 HDMI 不需要 进行声卡切换，

可以更改 framework

```
diff --git a/services/java/com/android/server/WiredAccessoryManager.java
b/services/java/com/android/server/WiredAccessoryManager.java
index c8d3510..2fb231e 100644
--- a/services/java/com/android/server/WiredAccessoryManager.java
+++ b/services/java/com/android/server/WiredAccessoryManager.java
@@ -374,7 +374,7 @@ final class WiredAccessoryManager implements WiredAccessoryCallbacks
{
             //
             // If the kernel does not have an "hdmi_audio" switch, just fall back on
the older
             // "hdmi" switch instead.
-            uei = new UEventInfo(NAME_HDMI_AUDIO, BIT_HDMI_AUDIO, 0);
+/*            uei = new UEventInfo(NAME_HDMI_AUDIO, BIT_HDMI_AUDIO, 0);
             if (uei.checkSwitchExists()) {
                 retVal.add(uei);
             } else {
@@ -385,7 +385,7 @@ final class WiredAccessoryManager implements WiredAccessoryCallbacks
{
                     Slog.w(TAG, "This kernel does not have HDMI audio support");
                 }
             }
-
+*/
             return retVal;
         }
```

或者 将 kernel 里面的 HDMI 读取到的 state 不进行上报

```
diff --git a/drivers/video/rockchip/hdmi/rk_hdmi_task.c
b/drivers/video/rockchip/hdmi/rk_hdmi_task.c
index 6cdb9eb..2ca862f 100755
--- a/drivers/video/rockchip/hdmi/rk_hdmi_task.c
+++ b/drivers/video/rockchip/hdmi/rk_hdmi_task.c
@@ -258,7 +258,7 @@ void hdmi_work(struct work_struct *work)
                                         hdmi_dbg(hdmi->dev,"base_audio_support
=%d,sink_hdmi = %d\n",hdmi->edid.base_audio_suppo
                                         #ifdef CONFIG_SWITCH
                                         if(hdmi->edid.base_audio_support == 1 &&
hdmi->edid.sink_hdmi == 1)
```

```
-
switch_set_state(&(hdmi->switch_hdmi), 1);
+
switch_set_state(&(hdmi->switch_hdmi), 0);
                                        #endif
                                        #ifdef CONFIG_RK_HDMI_CTL_CODEC
                                        #ifdef CONFIG_MACH_RK_FAC
```

Usb audio 为 card2 4.4 SDK 默认支持 usb audio route =29 、30, 5.1 之后的 SDK kernel 固定 usb audio 为 card3，使用 google 原生的 usb audio hal。

```
    //usb audio
    .usb_normal = {
        .sound_card = 2,
        .devices = DEVICES_0,
        .controls_count = 0,
    },
    .usb_capture = {
        .sound_card = 2,
        .devices = DEVICES_0,
        .controls_count = 0,
    },
```

3)、两个声卡同时输出

需要同时向两个声卡同时 write audio data 即可， 4.4 的实例代码如下:

```
diff --git a/AudioHardware.cpp b/AudioHardware.cpp
index bc2e553..60df29e 100755
--- a/AudioHardware.cpp
+++ b/AudioHardware.cpp
@@ -84,6 +84,7 @@ AudioHardware::AudioHardware() :
     mInit(false),
     mMicMute(false),
     mPcm(NULL),
+    mPcm1(NULL),
     mPcmOpenCnt(0),
```

```
     mMixerOpenCnt(0),
     mInCallAudioMode(false),
@@ -117,6 +118,10 @@ AudioHardware::~AudioHardware()
         TRACE_DRIVER_OUT
     }

+    if(mPcm1)
+    {
+   route_pcm1_close();
+    }
     TRACE_DRIVER_IN(DRV_MIXER_CLOSE)
     route_uninit();
     TRACE_DRIVER_OUT
@@ -685,6 +690,7 @@ struct pcm *AudioHardware::openPcmOut_l()
             mPcmOpenCnt--;
             mPcm = NULL;
         }
+   mPcm1 = route_pcm1_open(1, flags);
     }
     return mPcm;
 }
@@ -703,6 +709,8 @@ void AudioHardware::closePcmOut_l()
         route_pcm_close(PLAYBACK_OFF_ROUTE);
         TRACE_DRIVER_OUT
         mPcm = NULL;
+   route_pcm1_close();
+   mPcm1= NULL;
     }
 }

@@ -987,7 +995,9 @@ ssize_t AudioHardware::AudioStreamOutALSA::write(const void* buffer,
size_t byte
         }

         TRACE_DRIVER_IN(DRV_PCM_WRITE)
+        // ret = pcm_write(mHardware->getPcm(),(void*) p, bytes);
         ret = pcm_write(mHardware->getPcm(),(void*) p, bytes);
+        ret = pcm_write(mHardware->getPcm1(),(void*) p, bytes);
         TRACE_DRIVER_OUT
```

```
         if (ret == 0) {
diff --git a/AudioHardware.h b/AudioHardware.h
index 4d216df..14adf53 100755
--- a/AudioHardware.h
+++ b/AudioHardware.h
@@ -137,6 +137,7 @@ public:
            void closePcmOut_1();

            struct pcm *getPcm() { return mPcm; };
+           struct pcm *getPcm1() { return mPcm1; };

             android::sp <AudioStreamOutALSA>  output() { return mOutput; }

@@ -151,6 +152,7 @@ private:
      android::SortedVector <  android::sp<AudioStreamInALSA> >   mInputs;
      android::Mutex           mLock;
     struct pcm*     mPcm;
+    struct pcm*     mPcm1;
     uint32_t        mPcmOpenCnt;
     uint32_t        mMixerOpenCnt;
     bool            mInCallAudioMode;
diff --git a/alsa_audio.h b/alsa_audio.h
index 101fa2a..25583b1 100755
--- a/alsa_audio.h
+++ b/alsa_audio.h
@@ -163,5 +163,7 @@ int route_set_input_source(const char *source);
 int route_set_voice_volume(const char *ctlName, float volume);
 int route_set_controls(unsigned route);
 struct pcm *route_pcm_open(unsigned route, unsigned int flags);
+struct pcm *route_pcm1_open(int card, unsigned int flags);
 int route_pcm_close(unsigned route);
+int route_pcm1_close(void);
 #endif
diff --git a/alsa_route.c b/alsa_route.c
index 8e890b6..4006c2f 100755
--- a/alsa_route.c
+++ b/alsa_route.c
@@ -40,6 +40,7 @@
 const struct config_route_table *route_table;
```

```
 struct pcm* mPcm[PCM_MAX + 1];
+struct pcm* mPcm1;
 struct mixer* mMixerPlayback;
 struct mixer* mMixerCapture;

@@ -510,6 +511,34 @@ struct pcm *route_pcm_open(unsigned route, unsigned int flags)
     return is_playback ? mPcm[PCM_DEVICE0_PLAYBACK] : mPcm[PCM_DEVICE0_CAPTURE];
 }

+struct pcm *route_pcm1_open(int card, unsigned flags)
+{
+
+    ALOGD("lxt route_pcm1_open()");
+    flags &= ~PCM_CARD_MASK;
+    switch(card) {
+    case 1:
+        flags |= PCM_CARD1;
+        break;
+    case 2:
+        flags |= PCM_CARD2;
+        break;
+    default:
+        flags |= PCM_CARD1;
+        break;
+    }
+    flags &= ~PCM_DEVICE_MASK;
+    flags |= PCM_DEVICE0;
+    mPcm1 = pcm_open(flags);
+    return mPcm1;
+}
+
+int route_pcm1_close(void)
+{
+   ALOGD("lxt route_pcm1_close()");
+   pcm_close(mPcm1);
+   return 0;
+}
 int route_pcm_close(unsigned route)
 {
     unsigned i;
```

4)、确认 route 都正确但是还是没有声音

可以通过正在录音和放音的寄存器打印出来 跟正常的机器比较看是否是因为 codec 寄存器的配置导致没有声音的，寄存器不一样则需要查阅 datasheet 或者联系 codec FAE 进行排查。

如果寄存器跟正常的一直说明 codec 寄存器配置正确，这个时候需要进行硬件排查，首先确认一下 codec 各路电压、I2S 的 MCLK BCLK LRCLK 是否正常。

```
（1）寄存器打印：
Kernel3.0 中：cat sys/kernel/debug/asoc/RK_RK616/rk616-codec.0/codec_reg
Kernel3.10 中：cat sys/kernel/debug/asoc/RK_RT5616/rt5616.4-001b/codec_reg
（2）设置寄存器：
可以使用 echo ‘reg value’的方法写入寄存器。
Kernel3.0 中：echo ’01 bb’ > sys/kernel/debug/asoc/RK_RK616/rk616-codec.0/codec_reg
Kernel3.10 中：echo ’01 bb’ > sys/kernel/debug/asoc/RK_RT5616/rt5616.4-001b/codec_reg
```

如何调试新的 sound card 驱动

第一步看 codec 芯片资料

确认 codec 内部是否还需要控制一般是 i2c 初始化寄存器或者是通路的配置，如果不需要的可以直接使用 kernel 里面的 HDMI I2S (SND_RK_SOC_HDMI_I2S [=y])这个声卡驱动即可。这个是一个最简单的声卡驱动，有了这个驱动可以是 I2S 控制器能够正常的工作。

Kernel 里面 sound card 驱动的 menuconfig 路径如下，可以看到当前 kernel 支持的 codec 列表

Device Drivers > Sound card support > Advanced Linux Sound Architecture > ALSA for SoC audio support

手上的 3288 4.4 kernel 如下：

```
--- ALSA for SoC audio support
< >   SoC Audio for the Atmel System-on-Chip
< >   Synopsys I2S Device Driver
<*>   SoC Audio for the Rockchip System-on-Chip
      Set audio support for HDMI (HDMI use I2S)  --->
< >   SoC I2S Audio support for rockchip - AK4396
< >   SoC I2S Audio support for rockchip - ES8323
< >   SoC I2S Audio support for rockchip - ES8323 for PCM modem
< >   SoC I2S Audio support for rockchip - WM8988
< >   SoC I2S Audio support for rockchip - WM8900
< >   SoC I2S Audio support for rockchip - RICHTEK5512
< >   SoC I2S Audio support for rockchip - CX2070X
< >   SoC I2S Audio support for rockchip - rt5621
< >   SoC I2S Audio support for rockchip - rt5623
<*>   SoC I2S Audio support for rockchip - RT5631
< >   SoC I2S Audio support for rockchip(phone) - RT5631
< >   SoC I2S Audio support for rockchip - RT5625
< >   SoC I2S Audio support for rockchip - RT5640(RT5642)
<*>   SoC I2S Audio support for rockchip - RT3224
v(+)

        <Select>    < Exit >    < Help >    < Save >    < Load >
```

Codec 需要通路的配置

完成一个sound card 驱动，需要完成3个部分的驱动Machine、Soc/Platform、Codec,其中Soc/Platform平台驱动有soc 厂商做好，不需要再写，那么剩下的工作仅需要完成 Machine 和codec dirver。

以 es8323 为例调试新的驱动

首先在 arch/arm/boot/dts/rk3288-tb_8846.dts 中增加 codec 的描述

```
161         rockchip-es8323{
162                 compatible = "rockchip-es8323";
163                 dais {
164                         dai0 {
165                                 audio-codec = <&es8323>;
166                                 i2s-controller = <&i2s>;
167                                 format = "i2s";
168                                 //continuous-clock;
169                                 //bitclock-inversion;
170                                 //frame-inversion;
171                                 //bitclock-master;
172                                 //frame-master;
```

```
173                         };
174                 };
175         };
```

es8323 是 i2c 设备，因此需要增加 i2c 的描述

```
553 &i2c2 {
554         status = "okay";
555         rt5631: rt5631@1a {
556                 compatible = "rt5631";
557                 reg = <0x1a>;
558         };
559         es8323: es8323@10 {
560                 compatible = "es8323";
561                 reg = <0x10>;
562         };
```

Machine driver 的编写

```
#ifdef CONFIG_OF
static const struct of_device_id rockchip_es8323_of_match[] = {
    { .compatible = "rockchip-es8323", },
    {},
};
MODULE_DEVICE_TABLE(of, rockchip_es8323_of_match);
#endif /* CONFIG_OF */

static struct platform_driver rockchip_es8323_audio_driver = {
    .driver         = {
        .name   = "rockchip-es8323",
        .owner  = THIS_MODULE,
        .pm = &snd_soc_pm_ops,
        .of_match_table = of_match_ptr(rockchip_es8323_of_match),
    },
    .probe          = rockchip_es8323_audio_probe,
    .remove         = rockchip_es8323_audio_remove,
};

module_platform_driver(rockchip_es8323_audio_driver);
```

platform_driver 会通过 rockchip_es8323_of_match 进行查找到在 dts 中注册的资源节点将 sound

card 进行注册

```
static int rockchip_es8323_audio_probe(struct platform_device *pdev)
{
    int ret;
    struct snd_soc_card *card = &rockchip_es8323_snd_card;

    card->dev = &pdev->dev;
   // 解析 dts 中的 rockchip-es8323
    ret = rockchip_of_get_sound_card_info(card);
    if (ret) {
        printk("%s() get sound card info failed:%d\n", __FUNCTION__, ret);
        return ret;
    }

    ret = snd_soc_register_card(card);
    if (ret)
        printk("%s() register card failed:%d\n", __FUNCTION__, ret);

    return ret;
}
```

```
static struct snd_soc_ops rk29_ops = {
      .hw_params = rk29_hw_params,
};

static struct snd_soc_dai_link rk29_dai = {
    .name = "ES8323",
    .stream_name = "ES8323 PCM",
    .codec_dai_name = "ES8323 HiFi",
    .init = rk29_es8323_init,
    .ops = &rk29_ops,
};

static struct snd_soc_card rockchip_es8323_snd_card = {
    .name = "RK_ES8323",
    .dai_link = &rk29_dai,
    .num_links = 1,
};
```

具体可以查阅相关代码，其他 codec 套用这个架构即可。

Codec 驱动 codec 驱动主要是要跟之前的 machine driver 匹配。

kernel\sound\soc\codecs\es8323.c

```
static struct i2c_driver es8323_i2c_driver = {
    .driver = {
    // 这个名字要跟 rk29_dai 里面的大小写要一致否则找不到设备注册不上。
        .name = "ES8323",
        .owner = THIS_MODULE,
    },
    .shutdown = es8323_i2c_shutdown,
    .probe = es8323_i2c_probe,
    .remove = es8323_i2c_remove,
    .id_table = es8323_i2c_id,
};

static int __init es8323_init(void)
{
    return i2c_add_driver(&es8323_i2c_driver);
}

static void __exit es8323_exit(void)
{
    i2c_del_driver(&es8323_i2c_driver);
}

module_init(es8323_init);
module_exit(es8323_exit);
```

es8323_i2c_probe 成功会调用到

ret = snd_soc_register_codec(&i2c->dev,&soc_codec_dev_es8323, &es8323_dai, 1);

进行 codec dai 的注册，

```
static struct snd_soc_dai_driver es8323_dai = {
    // name 字段要跟    .codec_dai_name = "ES8323 HiFi",一致否则也找不到无法注册，
    .name = "ES8323 HiFi",
    .playback = {
        .stream_name = "Playback",
        .channels_min = 1,
```

```
        .channels_max = 2,
        .rates = es8323_RATES,
        .formats = es8323_FORMATS,
    },
    .capture = {
        .stream_name = "Capture",
        .channels_min = 1,
        .channels_max = 2,
        .rates = es8323_RATES,
        .formats = es8323_FORMATS,
     },
    .ops = &es8323_ops,
    .symmetric_rates = 1,
};
```

Sound card 注册不上没有注册参考《Audio 部分常见问题处理方法》声卡注册问题 debug。

如果 codec 的通路比较简单则可以直接在 codec driver 里面进行 通路的配置。为了减小工作量可以不在 HAL 层里面增加 codec 的 config_list.h，当然增加 config list 也是可以的。

如： RK3368_ANDROID5.1-SDK_V1.00_20150415\kernel\sound\soc\codecs\es8316.c

.startup = es8316_pcm_startup, 录音或者放音打开时候 会调用

.shutdown = es8316_pcm_shutdown, 录音或者放音关闭时候会调用

具体可以查阅相关代码

```
static struct snd_soc_dai_ops es8316_ops = {
    .startup = es8316_pcm_startup,
    .hw_params = es8316_pcm_hw_params,
    .set_fmt = es8316_set_dai_fmt,
    .set_sysclk = es8316_set_dai_sysclk,
    .digital_mute = es8316_mute,
    .shutdown = es8316_pcm_shutdown,
};
```

如果 codec 的通路配置比较复杂 例如 alc3224 简单的 startup shutdown 不可能完成所有 case 的通

路配置，则需要增加 codec 的 config_list.h，这个符合 alsa 的标准，需要联系 codec 原厂 fae 进行协助，例如 3224 等。

蓝牙通话 3G 通话的方案 debug

参考《RK_Android 平台蓝牙通话功能说明 v1.3.pdf》

312x 平台 codec debug

参考《RK312X_CODEC_开发说明文档 V1.0.pdf》

POP 音问题

喇叭 pop 音可以增加 mute 电路将 pop 音隔断，待声音正常输出时候再将功放打开可将 pop 音消除。可以参考《RK312X_CODEC_开发说明文档 V1.0.pdf》 pop 音问题 或者《rk616-rk618 常见问题及功能修改.pdf》中的，

```
#define SPK_AMP_DELAY 150（用于喇叭 spk 功放使能后延时时间设置，有些功放需要延时后才能正常
输出，具体时间不同。）
#define HP_MOS_DELAY 5（耳机 mos 管拉高后延时时间设置，有些 mos 管拉高后需要延时后声音才能正
常输出。）
```

其他 codec 也一样增加 mute 电路 调整上电时间可以将 codec 产生的 pop 音消除可以举一反三。

关于 ALC 功能

不改喇叭功率但是要音量加大，codec 如果有 ALC 功能， 可以联系 codec FAE 开启 ALC 功能，

如果 codec 没有 ALC 功能，可以联系 FAE 窗口获取 ALC 补丁。

关于降噪算法

使用的是 speex 的开源算法库，录音默认开启降噪算法，可以将部分噪声过滤，但是同时也会把背景声音也会过滤一部分。同时如果输入的信号是固定的信号比如 1K 正弦波信号，因为算法是针对的是语音信号，对固定信号有衰减的作用，因此 1K 正弦波这则比实际增益小很多。如需要关闭 patch 如下：

关闭降噪 4.4patch

```
hardware/rk29/audio$ git diff ./
diff --git a/AudioHardware.h b/AudioHardware.h
index 4d216df..92bceb2 100755
--- a/AudioHardware.h
+++ b/AudioHardware.h
@@ -77,7 +77,7 @@ namespace android_audio_legacy {

 //1:Enable the denoise funtion ;0: disable the denoise function

-#define SPEEX_DENOISE_ENABLE 1
+#define SPEEX_DENOISE_ENABLE 0
```

关闭 5.1 录音降噪算法 patch

```
hardware/rockchip/audio/tinyalsa_hal$ git diff audio_hw.c
diff --git a/tinyalsa_hal/audio_hw.c b/tinyalsa_hal/audio_hw.c
index 691b8ca..56cfa65 100755
--- a/tinyalsa_hal/audio_hw.c
+++ b/tinyalsa_hal/audio_hw.c
@@ -101,7 +101,7 @@ FILE *in_debug;

 #define AUDIO_HAL_VERSION "ALSA Audio Version: V0.8.0"

-#define SPEEX_DENOISE_ENABLE
+/* #define SPEEX_DENOISE_ENABLE */
```

Alsa 的上层应用程序

如果不希望使用 android 的 api 接口进行录放的话可以查阅相关代码 external/tinyalsa 。

Tinyalsa 是对 alsa-lib 进行了裁剪，alsa 官方提供的比较强大。BOX 4.4 和 4.2 的 SDK 是用 alsa lib。

后面的 5.1 SDK 统一使用 tinyalsa。

tinycap.c 是录音程序;

tinyplay.c 是放音程序;

pcm.c 是对 linux alsa 驱动的一些封装;

mixer.c 是相关的 ctl 设置主要是 codec 寄存器相关。

另外 5.1 SDK 的 audio Hal 层也是引用该目录的代码。

```
lxt@rksz-server101:~/rk3288-4.4.2/external/tinyalsa$ tree
├── Android.mk
├── include
│   └── tinyalsa
│       └── asoundlib.h
├── mixer.c
├── MODULE_LICENSE_BSD
├── NOTICE
├── pcm.c
├── README
├── tinycap.c
├── tinymix.c
├── tinypcminfo.c
└── tinyplay.c

2 directories, 11 files
```