

DroidPhone的专栏

欢迎各位大虾交流，本人联系方式：droid.phx@gmail.com

[目录视图](#)
[摘要视图](#)
[RSS 订阅](#)

个人资料



DroidPhone



访问：382342次
 积分：3434分
 排名：第2116名
 原创：46篇 转载：0篇
 译文：4篇 评论：356条

文章搜索

文章分类

[移动开发之Android \(11\)](#)
[Linux内核架构 \(15\)](#)
[Linux设备驱动 \(16\)](#)
[Linux电源管理 \(3\)](#)
[Linux音频子系统 \(15\)](#)
[Linux中断子系统 \(5\)](#)
[Linux时间管理系统 \(8\)](#)
[Linux输入子系统 \(4\)](#)

文章存档

[2013年11月 \(4\)](#)
[2013年10月 \(3\)](#)
[2013年07月 \(3\)](#)
[2012年12月 \(4\)](#)
[2012年10月 \(4\)](#)

展开

阅读排行

[Android Audio System 之 \(38982\)](#)
[Android Audio System 之 \(25553\)](#)
[Android Audio System 之 \(25317\)](#)
[Linux ALSA声卡驱动之一 \(24001\)](#)
[Linux ALSA声卡驱动之二 \(18421\)](#)
[Android SurfaceFlinger \(18421\)](#)

[投票赢好礼，周周有惊喜！](#)
[2014年4月微软MVP申请开始了！](#)
[消灭0回答，赢下载分](#)
[“我的2013”年度征文活动火爆进行中！](#)
[办公大师系列经典丛书 诚聘英才](#)

Linux ALSA声卡驱动之一：ALSA架构简介

分类：Linux音频子系统

2011-03-23 15:19

24106人阅读

评论(11) 收藏 举报

[linux](#)
[audio](#)
[codec](#)
[timer](#)
[playback](#)
[嵌入式](#)

目录(?) [-]

- 1.
2. 一 概述
3. 二 ALSA设备文件结构
4. 三 驱动的代码文件结构

声明：本博内容均由<http://blog.csdn.net/droidphone>原创，转载请注明出处，谢谢！

一. 概述

ALSA是Advanced Linux Sound Architecture 的缩写，目前已经成为了linux的主流音频体系结构，想了解更多的关于ALSA的这一开源项目的信息和知识，请查看以下网址：<http://www.alsa-project.org/>。

在内核设备驱动层，ALSA提供了alsa-driver，同时在应用层，ALSA为我们提供了alsa-lib，应用程序只要调用alsa-lib提供的API，即可以完成对底层音频硬件的控制。

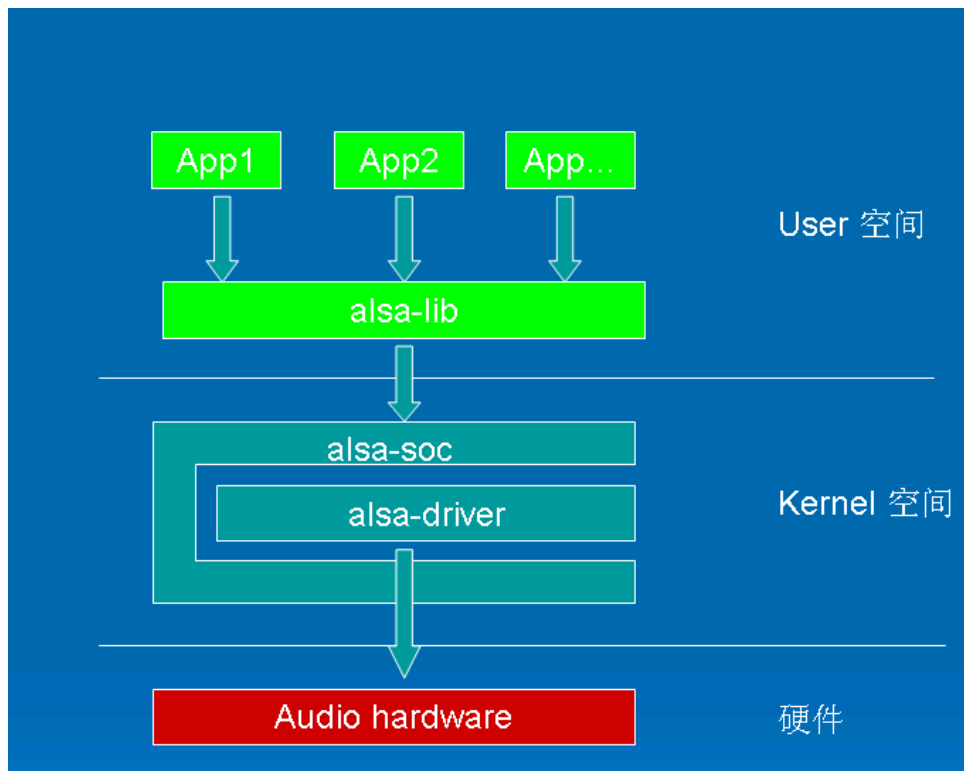


图 1.1 alsa的软件体系结构

由图1.1可以看出，用户空间的alsa-lib对应用程序提供统一的API接口，这样可以隐藏了驱动层的实现细节，简化

[Linux ALSA声卡驱动之三](#) (18248)
[Android中的sp和wp指针](#) (17112)
[Linux ALSA声卡驱动之七](#) (13786)
[Android SurfaceFlinger](#) (13061)
[Android SurfaceFlinger](#) (12550)

评论排行

[Android Audio System 之](#) (49)
[Linux ALSA声卡驱动之八](#) (30)
[Android SurfaceFlinger](#) (21)
[Linux ALSA声卡驱动之二](#) (18)
[Linux ALSA声卡驱动之三](#) (16)
[Android Audio System 之](#) (16)
[Linux中断（interrupt）子](#) (15)
[Android中的sp和wp指针](#) (13)
[Linux中断（interrupt）子](#) (12)
[Android SurfaceFlinger](#) (11)

推荐文章

* [SharePoint 2010/2013 使用 Javascript来判断权限的三种方法](#)
* [坚持前进的方向：总结 2013，规划2014](#)
* [创业者那些鲜为人知的事情](#)
* [ListView具有多种item布局——实现微信对话框](#)
* [实现自己的类加载时，重写方法loadClass与findClass的区别](#)
* [GDAL影像投影转换](#)

最新评论

[Linux输入子系统：多点触控协议 gocyl23](#): 很有用，多谢分享
[ALSA声卡驱动中的DAPM详解之 wsc_168](#): 楼主,您好： 现在正在移植wm8962的驱动，遇到了一些问题，请教一下您。串口信息显示已经扫描...
[Linux ALSA声卡驱动之五：移动i slc](#)ss: @DroidPhone:感谢您的回复，我是新手，想问下这个配置的具体位置在哪里？
[Linux ALSA声卡驱动之五：移动i DroidPhone](#): @u013222557:这种情况通常是你的codec中的音频路径把Mic至HP的路径被打开了，请仔细...
[Linux ALSA声卡驱动之五：移动i slc](#)ss: 您好： 我是一名在读研究生，最近在ALSA架构下搞嵌入式音频程序开发，遇到了一个棘手的问题...
[ALSA声卡驱动中的DAPM详解之 DroidPhone](#): @u012389631:和电源管理和音频路径相关的control需要定义为dapm control (...
[ALSA声卡驱动中的DAPM详解之 ellie](#)pfang: 大侠，你好！ 这两天把您的文章1-7 看了一遍，关于control这个概念在您的文章中有提到过多次...
[ALSA声卡驱动中的DAPM详解之 ellie](#)pfang: @DroidPhone:因为这个是wm8962的machine上的现有代码，但如果我要porting...

了应用程序的实现难度。内核空间中，alsa-soc其实是对alsa-driver的进一步封装，他针对嵌入式设备提供了一些列增强的功能。本系列博文仅对嵌入式系统中的alsa-driver和alsa-soc进行讨论。

二. ALSA设备文件结构

我们从alsa在linux中的设备文件结构开始我们的alsa之旅. 看看我的电脑中的alsa驱动的设备文件结构:

```
$ cd /dev/snd  
$ ls -l
```

```
crw-rw----+ 1 root audio 116, 8 2011-02-23 21:38 controlC0  
crw-rw----+ 1 root audio 116, 4 2011-02-23 21:38 midiC0D0  
crw-rw----+ 1 root audio 116, 7 2011-02-23 21:39 pcmC0D0c  
crw-rw----+ 1 root audio 116, 6 2011-02-23 21:56 pcmC0D0p  
crw-rw----+ 1 root audio 116, 5 2011-02-23 21:38 pcmC0D1p  
crw-rw----+ 1 root audio 116, 3 2011-02-23 21:38 seq  
crw-rw----+ 1 root audio 116, 2 2011-02-23 21:38 timer  
$
```

我们可以看到以下设备文件:

- controlC0 --> 用于声卡的控制，例如通道选择，混音，麦克风的控制等
- midiC0D0 --> 用于播放midi音频
- pcmC0D0c --> 用于录音的pcm设备
- pcmC0D0p --> 用于播放的pcm设备
- seq --> 音序器
- timer --> 定时器

其中，C0D0代表的是声卡0中的设备0，pcmC0D0c最后一个c代表capture，pcmC0D0p最后一个p代表playback，这些都是alsa-driver中的命名规则。从上面的列表可以看出，我的声卡下挂了6个设备，根据声卡的实际能力，驱动实际上可以挂上更多种类的设备，在include/sound/core.h中，定义了以下设备类型:

```
[c-sharp]  
  
01. #define SNDRV_DEV_TOPLEVEL ((__force snd_device_type_t) 0)  
02. #define SNDRV_DEV_CONTROL ((__force snd_device_type_t) 1)  
03. #define SNDRV_DEV_LOWLEVEL_PRE ((__force snd_device_type_t) 2)  
04. #define SNDRV_DEV_LOWLEVEL_NORMAL ((__force snd_device_type_t) 0x1000)  
05. #define SNDRV_DEV_PCM ((__force snd_device_type_t) 0x1001)  
06. #define SNDRV_DEV_RAWMIDI ((__force snd_device_type_t) 0x1002)  
07. #define SNDRV_DEV_TIMER ((__force snd_device_type_t) 0x1003)  
08. #define SNDRV_DEV_SEQUENCER ((__force snd_device_type_t) 0x1004)  
09. #define SNDRV_DEV_HWDEP ((__force snd_device_type_t) 0x1005)  
10. #define SNDRV_DEV_INFO ((__force snd_device_type_t) 0x1006)  
11. #define SNDRV_DEV_BUS ((__force snd_device_type_t) 0x1007)  
12. #define SNDRV_DEV_CODEC ((__force snd_device_type_t) 0x1008)  
13. #define SNDRV_DEV_JACK ((__force snd_device_type_t) 0x1009)  
14. #define SNDRV_DEV_LOWLEVEL ((__force snd_device_type_t) 0x2000)
```

通常，我们更关心的是pcm和control这两种设备。

三. 驱动的代码文件结构

在Linux2.6代码树中，Alsa的代码文件结构如下:

```
sound  
    /core  
    /oss  
    /seq  
    /ioctl32  
    /include  
    /drivers
```

Linux ALSA声卡驱动之六: ASoC
elliepsang: @DroidPhone:关于
snd_soc_dai_link结构中的代码
注释中struct snd_...

ALSA声卡驱动中的DAPM详解之
DroidPhone: @u012389631:这
种名字根据实际的意义自己定义
就好了, 只要符合常识即可。不
过通常还是会和co...

```
/i2c
/synth
/emux
/pci
/(cards)
/isa
/(cards)
/arm
/ppc
/sparc
/usb
/pcmcia /(cards)
/oss
/soc
/codecs
```

- core 该目录包含了ALSA驱动的中间层, 它是整个ALSA驱动的核心部分
- core/oss 包含模拟旧的OSS架构的PCM和Mixer模块
- core/seq 有关音序器相关的代码
- include ALSA驱动的公共头文件目录, 该目录的头文件需要导出给用户空间的应用程序使用, 通常, 驱动模块私有的头文件不应放置在这里
- drivers 放置一些与CPU、BUS架构无关的公用代码
- i2c ALSA自己的I2C控制代码
- pci pci声卡的顶层目录, 子目录包含各种pci声卡的代码
- isa isa声卡的顶层目录, 子目录包含各种isa声卡的代码
- soc 针对system-on-chip体系的中间层代码
- soc/codecs 针对soc体系的各种codec的代码, 与平台无关

更多 1

上一篇: [Android SurfaceFlinger中的Layer,LayerDim,LayerBlur,LayerBuffer](#)

下一篇: [Linux ALSA声卡驱动之二: 声卡的创建](#)

尚观顶级嵌入式开发课程

Embedded Linux Professional Training

尚观教育 · 技术为王

- 全国唯一ARM11驱动&内核开发
- 全程物联网智能嵌入式终端案例
- ARM官方授权培训中心

www.upemb.com

查看评论

9楼 [littlethunder](#) 2013-06-28 15:57发表



很强大~不过我在/usr/include/sound 目录下只找到如下头文件:

asequencer.h asound.h hdsp.h sb16_csp.h
asound_fm.h emu10k1.h hdspm.h sfnt_info.h
是alsa更新后发生变化还是我目录找错了?

Re: [DroidPhone](#) 2013-06-29 19:22发表



回复littlethunder: 呵呵, 你没有下载内核的源代码! www.kernel.org

Re: [littlethunder](#) 2013-06-29 19:30发表



回复DroidPhone: 原来是这样啊~ 不过我是想通过alsa驱动函数取得电脑(ubuntu12.04)扬声器的音频数据流, 这个, 下载内核源码后只用sound部分就可以做到吗? 还是需要安装什么东西? 谢谢~:-)

8楼 [vito_coleone](#) 2013-06-06 14:05发表



写的很好

7楼 [david10000](#) 2012-12-20 18:26发表



整个看了一遍, 感觉很不错。楼主能否研究下 蓝牙语音输入设备 在alsa中的应用, 我目前在做这一块, 资料很少。

6楼 [shenghuafenxiyi](#) 2012-09-11 18:14发表



写的很不错

5楼 天才2012 2012-06-25 13:02发表



再想问一下，是不是现在嵌入式linux下的音频驱动更多的趋向于以ASOC的架构来设计呢？

4楼 天才2012 2012-06-25 13:01发表



博主你好，看了你的关于ALSA和ASOC的介绍，学习到很多。不知道你所示的内核版本是多少来着。因为我在看的2.6.36中很多codec，ASOC平台驱动已经你所说的machine驱动这3部分内容，很多数据的结构体都发生了一定的变换。做了更简单的封装。

3楼 sunweizhong1024 2012-05-23 16:01发表



牛叉啊，很经典

2楼 平凡的人 2012-03-06 15:51发表



很好很强大

1楼 xg1410 2011-09-19 09:25发表



好犀利啊！

您还没有登录,请[\[登录\]](#)或[\[注册\]](#)

* 以上用户言论只代表其个人观点，不代表CSDN网站的观点或立场

专区推荐内容

Hadoop4Win+Eclip...
开启 Android 应用开发新...
如何提高游戏后台数据查找效率
从零开始编写网络游戏
#HTML (HTML5): 多媒体...
[C/C++]类别互相引用



更多招聘职位

我公司职位也要出现在这里

【langoon】java架构师
【北京富邦天成信息技术有限公司】java高级工程师
【南京科莱斯企业管理咨询有限公司】Flex开发工程师（南京）
【深圳西洋镜网络技术有限公司】游戏服务端开发工程师
【博彦科技（上海）有限公司】oracle数据库开发

核心技术类目

全部主题 Java VPN Android iOS ERP IE10 Eclipse CRM JavaScript Ubuntu NFC
WAP jQuery 数据库 BI HTML5 Spring Apache Hadoop .NET API HTML SDK IIS
Fedora XML LBS Unity Splashtop UML components Windows Mobile Rails QEMU KDE
Cassandra CloudStack FTC coremail OPhone CouchBase 云计算 iOS6 Rackspace
Web App SpringSide Maemo Compuware 大数据 aptech Perl Tornado Ruby Hibernate
ThinkPHP Spark HBase Pure Solr Angular Cloud Foundry Redis Scala Django
Bootstrap

公司简介 | 招贤纳士 | 广告服务 | 银行汇款帐号 | 联系方式 | 版权声明 | 法律顾问 | 问题报告

QQ客服 微博客服 论坛反馈 联系邮箱: webmaster@csdn.net 服务热线: 400-600-2320

京 ICP 证 070598 号

北京创新乐知信息技术有限公司 版权所有

世纪乐知(北京)网络技术有限公司 提供技术支持

江苏乐知网络技术有限公司 提供商务支持

Copyright © 1999-2012, CSDN.NET, All Rights Reserved



DroidPhone的专栏

欢迎各位大虾交流，本人联系方式：droid.phx@gmail.com

目录视图 摘要视图 RSS 订阅

个人资料



DroidPhone



访问：382336次

积分：3434分

排名：第2116名

原创：46篇 转载：0篇

译文：4篇 评论：356条

文章搜索

文章分类

移动开发之Android (11)

Linux内核架构 (15)

Linux设备驱动 (16)

Linux电源管理 (3)

Linux音频子系统 (15)

Linux中断子系统 (5)

Linux时间管理系统 (8)

Linux输入子系统 (4)

文章存档

2013年11月 (4)

2013年10月 (3)

2013年07月 (3)

2012年12月 (4)

2012年10月 (4)

展开

阅读排行

Android Audio System 之 (38982)

Android Audio System 之 (25553)

Android Audio System 之 (25317)

Linux ALSA声卡驱动之一 (24001)

Linux ALSA声卡驱动之二 (18421)

Android SurfaceFlinger中

投票赢好礼，周周有惊喜！ 2014年4月微软MVP申请开始了！ 消灭0回答，赢下载分 “我的2013”年度征文活动火爆进行中！ 办公大师系列经典丛书 诚聘英才

Linux ALSA声卡驱动之二：声卡的创建

分类：Linux设备驱动 Linux音频子系统

2011-03-30 19:15

18451人阅读

评论(18)

收藏

举报

linux struct module structure list

目录(?)

[-]

1. struct snd_card
 1. snd_card是什么
 2. snd_card的定义
2. 声卡的建立流程
 1. 第一步创建snd_card的一个实例
 2. 第二步创建声卡的芯片专用数据
 3. 第三步设置Driver的ID和名字
 4. 第四步创建声卡的功能部件逻辑设备例如PCMMixerMIDI等
 5. 第五步注册声卡
 6. 一个实际的例子
3. snd_card_create
4. snd_card_register

声明：本博内容均由<http://blog.csdn.net/droidphone>原创，转载请注明出处，谢谢！

1. struct snd_card

1.1. snd_card是什么

snd_card可以说是整个ALSA音频驱动最顶层的一个结构，整个声卡的软件逻辑结构开始于该结构，几乎所有与声音相关的逻辑设备都是在snd_card的管理之下，声卡驱动的第一个动作通常就是创建一个snd_card结构体。正因为如此，本节中，我们也从 struct snd_card开始吧。

1.2. snd_card的定义

snd_card的定义位于头文件中：include/sound/core.h

```
[c-sharp]
01. /* main structure for soundcard */
02.
03. struct snd_card {
04.     int number;          /* number of soundcard (index to
05.                          snd_cards) */
06.
07.     char id[16];          /* id string of this card */
08.     char driver[16];      /* driver name */
09.     char shortname[32];   /* short name of this soundcard */
10.     char longname[80];    /* name of this soundcard */
11.     char mixername[80];   /* mixer name */
12.     char components[128]; /* card components delimited with
13.                          space */
14.     struct module *module; /* top-level module */
15.
16.     void *private_data;   /* private data for soundcard */
17.     void (*private_free) (struct snd_card *card); /* callback for freeing of
18.                          private data */
19.     struct list_head devices; /* devices */
```

(18248)
Linux ALSA声卡驱动之三
(17112)
Android中的sp和wp指针
(13786)
Linux ALSA声卡驱动之七
(13061)
Android SurfaceFlinger
(12550)

评论排行

Android Audio System 之 (49)
Linux ALSA声卡驱动之八 (30)
Android SurfaceFlinger (21)
Linux ALSA声卡驱动之二 (18)
Linux ALSA声卡驱动之三 (16)
Android Audio System 之 (16)
Linux中断 (interrupt) 子 (15)
Android中的sp和wp指针 (13)
Linux中断 (interrupt) 子 (12)
Android SurfaceFlinger (11)

推荐文章

* SharePoint 2010/2013 使用 Javascript来判断权限的三种方法
* 坚持前进的方向：总结 2013，规划2014
* 创业者那些鲜为人知的事情
* ListView具有多种item布局——实现微信对话列
* 实现自己的类加载时，重写方法loadClass与findClass的区别
* GDAL影像投影转换

最新评论

Linux输入子系统：多点触控协议
gocy123: 很有用，多谢分享

ALSA声卡驱动中的DAPM详解之
wsc_168: 楼主,您好: 现在正在移植wm8962的驱动,遇到了一些问题,请教一下您.串口信息显示已经扫描...

Linux ALSA声卡驱动之五: 移动i
slc55: @DroidPhone:感谢您的回复,我是新手,想问下这个配置的具体位置在哪里?

Linux ALSA声卡驱动之五: 移动i
DroidPhone: @u013222557:这种情况通常是你的codec中的音频路径把Mic至HP的路径被打开了,请仔细...

Linux ALSA声卡驱动之五: 移动i
slc55: 您好: 我是一名在读研究生,最近在ALSA架构下搞嵌入式音频程序开发,遇到了一个棘手的问题...

ALSA声卡驱动中的DAPM详解之
DroidPhone: @u012389631:和电源管理和音频路径相关的control需要定义为dapm control (...)

ALSA声卡驱动中的DAPM详解之
elliepfang: 大侠,你好! 这两天把您的文章1-7 看了一遍,关于control这个概念在您的文章中有提到过多次...

ALSA声卡驱动中的DAPM详解之
elliepfang: @DroidPhone:因为这个是wm8962的machine上的现有代码,但如果我要porting...

```
20.
21.     unsigned int last_numid;    /* last used numeric ID */
22.     struct rw_semaphore controls_rwsem; /* controls list lock */
23.     rwlock_t ctl_files_rwlock; /* ctl_files list lock */
24.     int controls_count;    /* count of all controls */
25.     int user_ctl_count;    /* count of all user controls */
26.     struct list_head controls; /* all controls for this card */
27.     struct list_head ctl_files; /* active control files */
28.
29.     struct snd_info_entry *proc_root; /* root for soundcard specific files */
30.     struct snd_info_entry *proc_id; /* the card id */
31.     struct proc_dir_entry *proc_root_link; /* number link to real id */
32.
33.     struct list_head files_list; /* all files associated to this card */
34.     struct snd_shutdown_f_ops *s_f_ops; /* file operations in the shutdown
35.                                         state */
36.     spinlock_t files_lock; /* lock the files for this card */
37.     int shutdown; /* this card is going down */
38.     int free_on_last_close; /* free in context of file_release */
39.     wait_queue_head_t shutdown_sleep;
40.     struct device *dev; /* device assigned to this card */
41. #ifndef CONFIG_SYSFS_DEPRECATED
42.     struct device *card_dev; /* cardX object for sysfs */
43. #endif
44.
45. #ifdef CONFIG_PM
46.     unsigned int power_state; /* power state */
47.     struct mutex power_lock; /* power lock */
48.     wait_queue_head_t power_sleep;
49. #endif
50.
51. #if defined(CONFIG_SND_MIXER_OSS) || defined(CONFIG_SND_MIXER_OSS_MODULE)
52.     struct snd_mixer_oss *mixer_oss;
53.     int mixer_oss_change_count;
54. #endif
55. };
```

- struct list_head devices 记录该声卡下所有逻辑设备的链表
- struct list_head controls 记录该声卡下所有的控制单元的链表
- void *private_data 声卡的私有数据，可以在创建声卡时通过参数指定数据的大小

2. 声卡的建立流程

2.1.1. 第一步，创建snd_card的一个实例

```
[c-sharp]
01. struct snd_card *card;
02. int err;
03. ....
04. err = snd_card_create(index, id, THIS_MODULE, 0, &card);
```

- index 一个整数值，该声卡的编号
- id 字符串，声卡的标识符
- 第四个参数 该参数决定在创建snd_card实例时，需要同时额外分配的私有数据的大小，该数据的指针最终会赋值给snd_card的private_data数据成员
- card 返回所创建的snd_card实例的指针

2.1.2. 第二步，创建声卡的芯片专用数据

声卡的专用数据主要用于存放该声卡的一些资源信息，例如中断资源、io资源、dma资源等。可以有两种创建方法：

- 通过上一步中snd_card_create()中的第四个参数，让snd_card_create自己创建

```
[c-sharp]
01. // struct mychip 用于保存专用数据
```

Linux ALSA声卡驱动之六: ASoC
elliepfang: @DroidPhone:关于
snd_soc_dai_link结构中的代码
注释中struct snd_...

ALSA声卡驱动中的DAPM详解之
DroidPhone: @u012389631:这
种名字根据实际的意义自己定义
就好了, 只要符合常识即可。不
过通常还是会和co...

```
02. err = snd_card_create(index, id, THIS_MODULE,  
03.                          sizeof(struct mychip), &card);  
04. // 从private_data中取出  
05. struct mychip *chip = card->private_data;
```

- 自己创建:

```
[c-sharp]  
  
01. struct mychip {  
02.     struct snd_card *card;  
03.     ....  
04. };  
05. struct snd_card *card;  
06. struct mychip *chip;  
07.  
08. chip = kzalloc(sizeof(*chip), GFP_KERNEL);  
09. ....  
10. err = snd_card_create(index[dev], id[dev], THIS_MODULE, 0, &card);  
11. // 专用数据记录snd_card实例  
12. chip->card = card;  
13. ....
```

然后, 把芯片的专有数据注册为声卡的一个低阶设备:

```
[c-sharp]  
  
01. static int snd_mychip_dev_free(struct snd_device *device)  
02. {  
03.     return snd_mychip_free(device->device_data);  
04. }  
05.  
06. static struct snd_device_ops ops = {  
07.     .dev_free = snd_mychip_dev_free,  
08. };  
09. ....  
10. snd_device_new(card, SNDRV_DEV_LOWLEVEL, chip, &ops);
```

注册为低阶设备主要是为了当声卡被注销时, 芯片专用数据所占用的内存可以被自动地释放。

2.1.3. 第三步, 设置Driver的ID和名字

```
[c-sharp]  
  
01. strcpy(card->driver, "My Chip");  
02. strcpy(card->shortname, "My Own Chip 123");  
03. sprintf(card->longname, "%s at 0x%lx irq %i",  
04.         card->shortname, chip->ioport, chip->irq);
```

snd_card的driver字段保存着芯片的ID字符串, user空间的alsa-lib会使用到该字符串, 所以必须要保证该ID的唯一性。shortname字段更多地用于打印信息, longname字段则会出现在/proc/asound/cards中。

2.1.4. 第四步, 创建声卡的功能部件(逻辑设备), 例如PCM, Mixer, MIDI等

这时候可以创建声卡的各种功能部件了, 还记得开头的snd_card结构体的devices字段吗? 每一种部件的创建最终会调用snd_device_new()来生成一个snd_device实例, 并把该实例链接到snd_card的devices链表中。

通常, alsa-driver的已经提供了一些常用的部件的创建函数, 而不必直接调用snd_device_new(), 比如:

```
PCM ---- snd_pcm_new()  
  
RAWMIDI -- snd_rawmidi_new()  
  
CONTROL -- snd_ctl_create()  
  
TIMER -- snd_timer_new()  
  
INFO -- snd_card_proc_new()  
  
JACK -- snd_jack_new()
```


2.1.5. 第五步，注册声卡

```
[c-sharp]

01. err = snd_card_register(card);
02. if (err < 0) {
03.     snd_card_free(card);
04.     return err;
05. }
```

2.2. 一个实际的例子

我把/sound/arm/pxa2xx-ac97.c的部分代码贴上来：

```
[cpp]

01. static int __devinit pxa2xx_ac97_probe(struct platform_device *dev)
02. {
03.     struct snd_card *card;
04.     struct snd_ac97_bus *ac97_bus;
05.     struct snd_ac97_template ac97_template;
06.     int ret;
07.     pxa2xx_audio_ops_t *pdata = dev->dev.platform_data;
08.
09.     if (dev->id >= 0) {
10.         dev_err(&dev->dev, "PXA2xx has only one AC97 port./n");
11.         ret = -ENXIO;
12.         goto err_dev;
13.     }
14.     ///(1)///
15.     ret = snd_card_create(SNDRV_DEFAULT_IDX1, SNDRV_DEFAULT_STR1,
16.                          THIS_MODULE, 0, &card);
17.     if (ret < 0)
18.         goto err;
19.
20.     card->dev = &dev->dev;
21.     ///(3)///
22.     strncpy(card->driver, dev->dev.driver->name, sizeof(card->driver));
23.
24.     ///(4)///
25.     ret = pxa2xx_pcm_new(card, &pxa2xx_ac97_pcm_client, &pxa2xx_ac97_pcm);
26.     if (ret)
27.         goto err;
28.     ///(2)///
29.     ret = pxa2xx_ac97_hw_probe(dev);
30.     if (ret)
31.         goto err;
32.
33.     ///(4)///
34.     ret = snd_ac97_bus(card, 0, &pxa2xx_ac97_ops, NULL, &ac97_bus);
35.     if (ret)
36.         goto err_remove;
37.     memset(&ac97_template, 0, sizeof(ac97_template));
38.     ret = snd_ac97_mixer(ac97_bus, &ac97_template, &pxa2xx_ac97_ac97);
39.     if (ret)
40.         goto err_remove;
41.     ///(3)///
42.     snprintf(card->shortname, sizeof(card->shortname),
43.              "%s", snd_ac97_get_short_name(pxa2xx_ac97_ac97));
44.     snprintf(card->longname, sizeof(card->longname),
45.              "%s (%s)", dev->dev.driver->name, card->mixername);
46.
47.     if (pdata && pdata->codec_pdata[0])
48.         snd_ac97_dev_add_pdata(ac97_bus->codec[0], pdata->codec_pdata[0]);
49.     snd_card_set_dev(card, &dev->dev);
50.     ///(5)///
51.     ret = snd_card_register(card);
52.     if (ret == 0) {
53.         platform_set_drvdata(dev, card);
54.         return 0;
55.     }
56. }
```



```

57.     err_remove:
58.         pxa2xx_ac97_hw_remove(dev);
59.     err:
60.         if (card)
61.             snd_card_free(card);
62.     err_dev:
63.         return ret;
64.     }
65.
66.     static int __devexit pxa2xx_ac97_remove(struct platform_device *dev)
67.     {
68.         struct snd_card *card = platform_get_drvdata(dev);
69.
70.         if (card) {
71.             snd_card_free(card);
72.             platform_set_drvdata(dev, NULL);
73.             pxa2xx_ac97_hw_remove(dev);
74.         }
75.
76.         return 0;
77.     }
78.
79.     static struct platform_driver pxa2xx_ac97_driver = {
80.         .probe      = pxa2xx_ac97_probe,
81.         .remove     = __devexit_p(pxa2xx_ac97_remove),
82.         .driver      = {
83.             .name     = "pxa2xx-ac97",
84.             .owner     = THIS_MODULE,
85. #ifdef CONFIG_PM
86.             .pm       = &pxa2xx_ac97_pm_ops,
87. #endif
88.         },
89.     };
90.
91.     static int __init pxa2xx_ac97_init(void)
92.     {
93.         return platform_driver_register(&pxa2xx_ac97_driver);
94.     }
95.
96.     static void __exit pxa2xx_ac97_exit(void)
97.     {
98.         platform_driver_unregister(&pxa2xx_ac97_driver);
99.     }
100.
101.     module_init(pxa2xx_ac97_init);
102.     module_exit(pxa2xx_ac97_exit);
103.
104.     MODULE_AUTHOR("Nicolas Pitre");
105.     MODULE_DESCRIPTION("AC97 driver for the Intel PXA2xx chip");

```

驱动程序通常由probe回调函数开始，对一下2.1中的步骤，是否有相似之处？

经过以上的创建步骤之后，声卡的逻辑结构如下图所示：

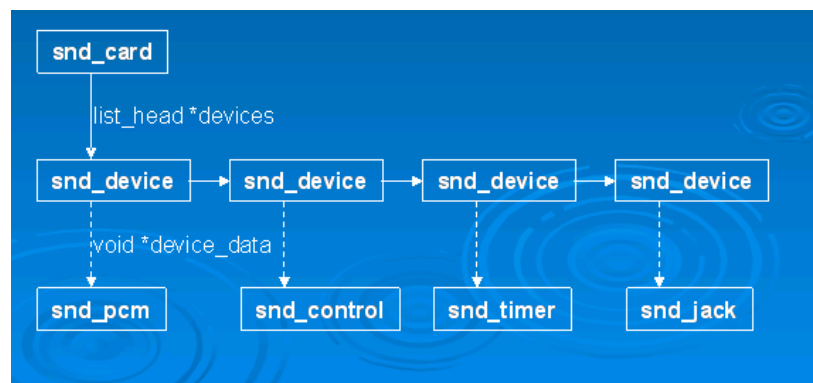


图 2.2.1 声卡的软件逻辑结构

下面的章节里我们分别讨论一下snd_card_create()和snd_card_register()这两个函数。

3. snd_card_create()

snd_card_create()在/sound/core/init.c中定义。

```
[cpp]
01.  /**
02.   * snd_card_create - create and initialize a soundcard structure
03.   * @idx: card index (address) [0 ... (SNDRV_CARDS-1)]
04.   * @xid: card identification (ASCII string)
05.   * @module: top level module for locking
06.   * @extra_size: allocate this extra size after the main soundcard structure
07.   * @card_ret: the pointer to store the created card instance
08.   *
09.   * Creates and initializes a soundcard structure.
10.   *
11.   * The function allocates snd_card instance via kzalloc with the given
12.   * space for the driver to use freely. The allocated struct is stored
13.   * in the given card_ret pointer.
14.   *
15.   * Returns zero if successful or a negative error code.
16.   */
17. int snd_card_create(int idx, const char *xid,
18.                     struct module *module, int extra_size,
19.                     struct snd_card **card_ret)
```

首先，根据extra_size参数的大小分配内存，该内存区可以作为芯片的专有数据使用（见前面的介绍）：

```
[c-sharp]
01. card = kzalloc(sizeof(*card) + extra_size, GFP_KERNEL);
02. if (!card)
03.     return -ENOMEM;
```

拷贝声卡的ID字符串：

```
[c-sharp]
01. if (xid)
02.     strncpy(card->id, xid, sizeof(card->id));
```

如果传入的声卡编号为-1，自动分配一个索引编号：

```
[c-sharp]
01. if (idx < 0) {
02.     for (idx2 = 0; idx2 < SNDRV_CARDS; idx2++)
03.         /* idx == -1 == 0xffff means: take any free slot */
04.         if (~snd_cards_lock & idx & 1<<idx2) {
05.             if (module_slot_match(module, idx2)) {
06.                 idx = idx2;
07.                 break;
08.             }
09.         }
10. }
11. if (idx < 0) {
12.     for (idx2 = 0; idx2 < SNDRV_CARDS; idx2++)
13.         /* idx == -1 == 0xffff means: take any free slot */
14.         if (~snd_cards_lock & idx & 1<<idx2) {
15.             if (!slots[idx2] || !*slots[idx2]) {
16.                 idx = idx2;
17.                 break;
18.             }
19.         }
20. }
```

初始化snd_card结构中必要的字段：

```
[c-sharp]
01. card->number = idx;
```

```

02.     card->module = module;
03.     INIT_LIST_HEAD(&card->devices);
04.     init_rwsem(&card->controls_rwsem);
05.     rwlock_init(&card->ctl_files_rwlock);
06.     INIT_LIST_HEAD(&card->controls);
07.     INIT_LIST_HEAD(&card->ctl_files);
08.     spin_lock_init(&card->files_lock);
09.     INIT_LIST_HEAD(&card->files_list);
10.     init_waitqueue_head(&card->shutdown_sleep);
11. #ifdef CONFIG_PM
12.     mutex_init(&card->power_lock);
13.     init_waitqueue_head(&card->power_sleep);
14. #endif

```

建立逻辑设备：Control

```

[c-sharp]

01.  /* the control interface cannot be accessed from the user space until */
02.  /* snd_cards_bitmask and snd_cards are set with snd_card_register */
03.  err = snd_ctl_create(card);

```

建立proc文件中的info节点：通常就是/proc/asound/card0

```

[c-sharp]

01.  err = snd_info_card_create(card);

```

把第一步分配的内存指针放入private_data字段中：

```

[c-sharp]

01.  if (extra_size > 0)
02.      card->private_data = (char *)card + sizeof(struct snd_card);

```

4. snd_card_register()

snd_card_create()在/sound/core/init.c中定义。

```

[c-sharp]

01.  /**
02.   * snd_card_register - register the soundcard
03.   * @card: soundcard structure
04.   *
05.   * This function registers all the devices assigned to the soundcard.
06.   * Until calling this, the ALSA control interface is blocked from the
07.   * external accesses. Thus, you should call this function at the end
08.   * of the initialization of the card.
09.   *
10.   * Returns zero otherwise a negative error code if the register failed.
11.   */
12.  int snd_card_register(struct snd_card *card)

```

首先，创建sysfs下的设备：

```

[c-sharp]

01.  if (!card->card_dev) {
02.      card->card_dev = device_create(sound_class, card->dev,
03.                                   MKDEV(0, 0), card,
04.                                   "card%i", card->number);
05.      if (IS_ERR(card->card_dev))
06.          card->card_dev = NULL;
07.  }

```

其中，sound_class是在/sound/sound_core.c中创建的：

```

[c-sharp]

01.  static char *sound_devnode(struct device *dev, mode_t *mode)
02.  {
03.      if (MAJOR(dev->devt) == SOUND_MAJOR)
04.          return NULL;
05.      return kasprintf(GFP_KERNEL, "snd/%s", dev_name(dev));
06.  }
07.  static int __init init_soundcore(void)
08.  {

```

```

09.     int rc;
10.
11.     rc = init_oss_soundcore();
12.     if (rc)
13.         return rc;
14.
15.     sound_class = class_create(THIS_MODULE, "sound");
16.     if (IS_ERR(sound_class)) {
17.         cleanup_oss_soundcore();
18.         return PTR_ERR(sound_class);
19.     }
20.
21.     sound_class->devnode = sound_devnode;
22.
23.     return 0;
24. }

```

由此可见，声卡的class将会出现在文件系统的/sys/class/sound/下面，并且，sound_devnode()也决定了相应的设备节点也将出现在/dev/snd/下面。

接下来的步骤，通过snd_device_register_all()注册所有挂在该声卡下的逻辑设备，snd_device_register_all()实际上是通过snd_card的devices链表，遍历所有的snd_device，并且调用snd_device的ops->dev_register()来实现各自设备的注册的。

```

[c-sharp]
01. if ((err = snd_device_register_all(card)) < 0)
02.     return err;

```

最后就是建立一些相应的proc和sysfs下的文件或属性节点，代码就不贴了。

至此，整个声卡完成了建立过程。

更多 1

上一篇: [Linux ALSA声卡驱动之一: ALSA架构简介](#)

下一篇: [Linux ALSA声卡驱动之三: PCM设备的创建](#)

linux		电源管理
嵌入式li		linux
teamview		本田商务

查看评论

13楼 [辉捺天韵](#) 2013-12-10 11:58发表



谢谢!!!

12楼 [shallot0000](#) 2013-08-15 11:39发表



真的不错，很希望自己有一天也能写出类似的文章跟大家分享，膜拜中。。。

11楼 [ljf10000](#) 2013-08-14 16:03发表



2.1.2 里有个错误（自己创建）

```

01.struct mychip {
02. struct snd_card *card;
03. ....
04.};
05.struct snd_card *card;
06.struct mychip *chip;
07.err = snd_card_create(index[dev], id[dev], THIS_MODULE, 0, &card);
08.// 专用数据记录snd_card实例
09.chip->card = card; // 这一步chip还没分配内存，
10.....
11.chip = kzalloc(sizeof(*chip), GFP_KERNEL);

```

Re: [DroidPhone](#) 2013-08-14 21:39发表



回复ljf10000: 嗯，是的，已修改。
Thanks!

10楼 [vito_coleone](#) 2013-05-24 14:32发表



分析的好

9楼 [牧风听](#) 2013-03-19 16:57发表



漂亮~

8楼 [hua3x](#) 2012-10-30 16:03发表



很牛呀，没有几年的经验写不出来呀，佩服。

7楼 [wulong117](#) 2012-09-12 09:41发表



分析的非常好，大哥是牛人啊！！

6楼 [haokaihaohe110](#) 2012-05-17 09:24发表



很好，很详细。

博主，你好，我想问一下，我在android下使用alsa时，在asound.conf中设置扬声器音量为100，为什么在启动时会出现暴音呢，这个问题我一直找不到解决办法，也不知道怎样在代码中修改。求博主帮忙啊，下面是我发的求助帖子：

Android2.3上面的音频系统ALSA的asound.conf中有一段设置扬声器音量的位置，即：

```
pcm.AndroidPlayback_Speaker_normal {
    type hooks
    slave.pcm {
        type hw
        card 1
        device 0 # Must be of type "digital audio playback"
    }
    hooks.0 {
        type ctl_elems
        hook_args [

{
    name 'Line DAC Playback Volume'
    value.0 100 //这个位置
    value.1 100
}
{
    name 'HP DAC Playback Volume'
    value.0 0
    value.1 0
}
{
    name 'Telephony'
    value Off
}

]
}
}
```

将 name 'Line DAC Playback Volume'下面的value设置为最大值100或者小于100的时候，启动时总会听到扬声器出现一声“噗”的爆音，设置为0的时候，系统就没有声音了。

请大侠们帮帮忙，应该怎么解决这个问题？

Re: [DroidPhone](#) 2012-05-17 10:34发表



回复haokaihaohe110: 这个应该查一下你的驱动中开启speaker时，codec中各个子部件的上电和unmute顺序，功放最好在最后打开，否则设置dac等参数时，可能造成异响。

Re: [haokaihaohe110](#) 2012-05-17 11:04发表



回复DroidPhone: 博主，您好，感谢您的回复。我使用的是TLV320的codec芯片，在aic3x_init函数里面有这么一段代码，好像是unmute的（我不是很懂驱动代码）：

```
/* unmute all outputs */
reg = snd_soc_read(codec, LLOPM_CTRL);
snd_soc_write(codec, LLOPM_CTRL, reg | UNMUTE);
reg = snd_soc_read(codec, RLOPM_CTRL);
snd_soc_write(codec, RLOPM_CTRL, reg | UNMUTE);
reg = snd_soc_read(codec, MONOLOPM_CTRL);
snd_soc_write(codec, MONOLOPM_CTRL, reg | UNMUTE);
reg = snd_soc_read(codec, HPLOUT_CTRL);
```

snd_soc_write(codec, HPLOUT_CTRL, reg | UNMUTE);
reg = snd_soc_read(codec, HPROUT_CTRL);
snd_soc_write(codec, HPROUT_CTRL, reg | UNMUTE);
reg = snd_soc_read(codec, HPLCOM_CTRL);
snd_soc_write(codec, HPLCOM_CTRL, reg | UNMUTE);
reg = snd_soc_read(codec, HPRCOM_CTRL);
snd_soc_write(codec, HPRCOM_CTRL, reg | UNMUTE);
我不清楚打开speaker和设置音量的位置。
还请博主帮忙啊。多谢了。

Re: [cax_x](#) 2012-05-21 19:10发表



回复haokaihaoh110: unmute状态下, DAC、HPLOUT、HPRPUT上电都容易引起pop音, 给你两个建议:
1、init过程中, codec复位后, 先mute住(应该是reg43和reg44), 然后对DAC等各部件进行操作, 最后unmute。
2、其他内容不改, 只把HPLOUT Output Level Control Register 和 HPROUT Output Level Control Register 写成0x9a

5楼 [jzp0409](#) 2012-05-08 11:22发表



请问一下您所介绍的这些资料所引用的内核版本是哪一个版本啊?

Re: [DroidPhone](#) 2012-05-08 12:20发表



回复jzp0409: 如果没有记错, 这篇好像是基于2.6.35的吧。

4楼 [hainei_](#) 2012-02-15 19:15发表



谢谢楼主了!

3楼 [allen6268198](#) 2011-09-19 10:08发表



so nice

2楼 [xubo830711](#) 2011-05-10 15:31发表



[e01]

1楼 [sxjx2009](#) 2011-04-28 08:54发表



[e03], 持续关注中。。。。

您还没有登录,请[\[登录\]](#)或[\[注册\]](#)

* 以上用户言论只代表其个人观点, 不代表CSDN网站的观点或立场

专区推荐内容

用于开发移动应用的 HTML5 ...
线程挂起, 恢复与中止等操作
使用 Thread 类创建与启动...
下载QNX白皮书助您完成卓越的嵌...
在HTML5中, 如何从C#执行网...
如何从Javascript传送字...



更多招聘职位

我公司职位也要出现在这里

【南京科莱斯企业管理咨询有限公司】Flex开发工程师 (南京)
【深圳西洋镜网络技术有限公司】游戏服务端开发工程师
【博彦科技(上海)有限公司】oracle数据库开发
【长沙中科院文化创意与科技产业研究院】Java研发工程师

核心技术类目

全部主题 Java VPN Android iOS ERP IE10 Eclipse CRM JavaScript Ubuntu NFC
WAP jQuery 数据库 BI HTML5 Spring Apache Hadoop .NET API HTML SDK IIS
Fedora XML LBS Unity Splashtop UML components Windows Mobile Rails QEMU KDE
Cassandra CloudStack FTC coremail OPhone CouchBase 云计算 iOS6 Rackspace
Web App SpringSide Maemo Compuware 大数据 aptech Perl Tomado Ruby Hibernate
ThinkPHP Spark HBase Pure Solr Angular Cloud Foundry Redis Scala Django
Bootstrap

世纪乐知(北京)网络技术有限公司 提供技术支持

江苏乐知网络技术有限公司 提供商务支持

Copyright © 1999-2012, CSDN.NET, All Rights Reserved 

DroidPhone的专栏

欢迎各位大虾交流，本人联系方式：droid.phx@gmail.com

目录视图 摘要视图 RSS 订阅

个人资料



DroidPhone



访问：382342次

积分：3434分

排名：第2116名

原创：46篇 转载：0篇

译文：4篇 评论：356条

文章搜索

文章分类

移动开发之Android (11)

Linux内核架构 (15)

Linux设备驱动 (16)

Linux电源管理 (3)

Linux音频子系统 (15)

Linux中断子系统 (5)

Linux时间管理系统 (8)

Linux输入子系统 (4)

文章存档

2013年11月 (4)

2013年10月 (3)

2013年07月 (3)

2012年12月 (4)

2012年10月 (4)

展开

阅读排行

Android Audio System 之 (38982)

Android Audio System 之 (25553)

Android Audio System 之 (25317)

Linux ALSA声卡驱动之一 (24001)

Linux ALSA声卡驱动之二 (18421)

Android SurfaceFlinger中

投票赢好礼，周周有惊喜！ 2014年4月微软MVP申请开始了！ 消灭0回答，赢下载分 “我的2013”年度征文活动火爆进行中！ 办公大师系列经典丛书 诚聘英才

Linux ALSA声卡驱动之三：PCM设备的创建

分类：Linux设备驱动 Linux音频子系统

2011-04-07 21:18

17129人阅读

评论(16)

收藏

举报

linux playback struct stream file autoload

目录(?)

[-]

- 1.
2. PCM是什么
3. alsa-driver中的PCM中间层
4. 新建一个pcm
5. 设备文件节点的建立devsndpcmCxxDxxppcmCxxDxxc
6. struct snd_minor
7. 设备文件的建立
8. 层层深入从应用程序到驱动层pcm
9. 字符设备注册
10. 打开pcm设备

声明：本博内容均由<http://blog.csdn.net/droidphone>原创，转载请注明出处，谢谢！

1. PCM是什么

PCM是英文Pulse-code modulation的缩写，中文译名是脉冲编码调制。我们知道在现实生活中，人耳听到的声音是模拟信号，PCM就是要将声音从模拟转换成数字信号的一种技术，他的原理简单地说就是利用一个固定的频率对模拟信号进行采样，采样后的信号在波形上看就像一串连续的幅值不一的脉冲，把这些脉冲的幅值按一定的精度进行量化，这些量化后的数值被连续地输出、传输、处理或记录到存储介质中，所有这些组成了数字音频的产生过程。

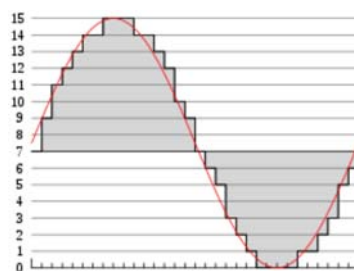


图1.1 模拟音频的采样、量化

PCM信号的两个重要指标是采样频率和量化精度，目前，CD音频的采样频率通常为44100Hz，量化精度是16bit。通常，播放音乐时，应用程序从存储介质中读取音频数据（MP3、WMA、AAC.....），经过解码后，最终送到音频驱动程序中的就是PCM数据，反过来，在录音时，音频驱动不停地把采样所得的PCM数据送回给应用程序，由应用程序完成压缩、存储等任务。所以，音频驱动的两大核心任务就是：

- playback 如何把用户空间的应用程序发过来的PCM数据，转化为人耳可以辨别的模拟音频
- capture 把mic拾取得模拟信号，经过采样、量化，转换为PCM信号送回给用户空间的应用程序

2. alsa-driver中的PCM中间层

Linux ALSA声卡驱动之三	(18248)
Android中的sp和wp指针	(17112)
Linux ALSA声卡驱动之七	(13786)
Android SurfaceFlinger中	(13061)
	(12550)

评论排行

- Android Audio System 之 (49)
- Linux ALSA 声卡驱动之八 (30)
- Android SurfaceFlinger 中 (21)
- Linux ALSA 声卡驱动之二 (18)
- Linux ALSA 声卡驱动之三 (16)
- Android Audio System 之 (16)
- Linux 中断 (interrupt) 子 (15)
- Android 中的 sp 和 wp 指针 (13)
- Linux 中断 (interrupt) 子 (12)
- Android SurfaceFlinger 中 (11)

推荐文章

- * SharePoint 2010/2013 使用 Javascript来判断权限的三种方法
- * 坚持前进的方向：总结 2013，规划2014
- * 创业者那些鲜为人知的事情
- * ListView具有多种item布局——实现微信对话框
- * 实现自己的类加载时，重写方法loadClass与findClass的区别
- * GDAL影像投影转换

Linux ALSA声卡驱动之六: ASoC
elliepsang: @DroidPhone:关于
snd_soc_dai_link结构中的代码
注释中struct snd_...

ALSA声卡驱动中的DAPM详解之
DroidPhone: @u012389631:这
种名字根据实际的意义自己定义
就好了, 只要符合常识即可。不
过通常还是会和co...

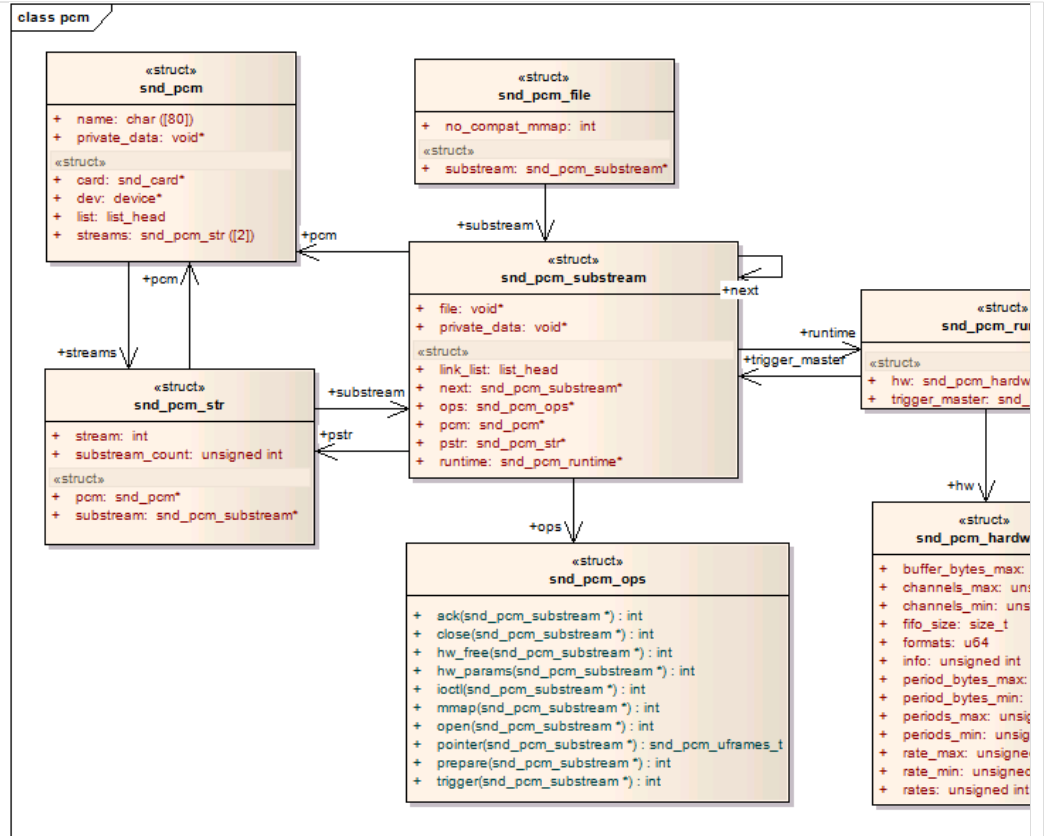


图2.2 pcm中间层的几个重要的结构体的关系图

- snd_pcm是挂在snd_card下面的一个snd_device
- snd_pcm中的字段：streams[2]，该数组中的两个元素指向两个snd_pcm_str结构，分别代表playback stream和capture stream
- snd_pcm_str中的substream字段，指向snd_pcm_substream结构
- snd_pcm_substream是pcm中间层的核心，绝大部分任务都是在substream中处理，尤其是他的ops（snd_pcm_ops）字段，许多user空间的应用程序通过alsa-lib对驱动程序请求都是由该结构中的函数处理。它的runtime字段则指向snd_pcm_runtime结构，snd_pcm_runtime记录这substream的一些重要的软件和硬件运行环境和参数。

3. 新建一个pcm

alsa-driver的中间层已经为我们提供了新建pcm的api:

```
int snd_pcm_new(struct snd_card *card, const char *id, int device, int playback_count, int capture_count,
                struct snd_pcm ** rpcm);
```

参数device 表示目前创建的是该声卡下的第几个pcm，第一个pcm设备从0开始。

参数playback_count 表示该pcm将会有几个playback substream。

参数capture_count 表示该pcm将会有几个capture substream。

另一个用于设置pcm操作函数接口的api:

void snd_pcm_set_ops(struct snd_pcm *pcm, int direction, struct snd_pcm_ops *ops);

新建一个pcm可以用下面一张新建pcm的调用的序列图进行描述：

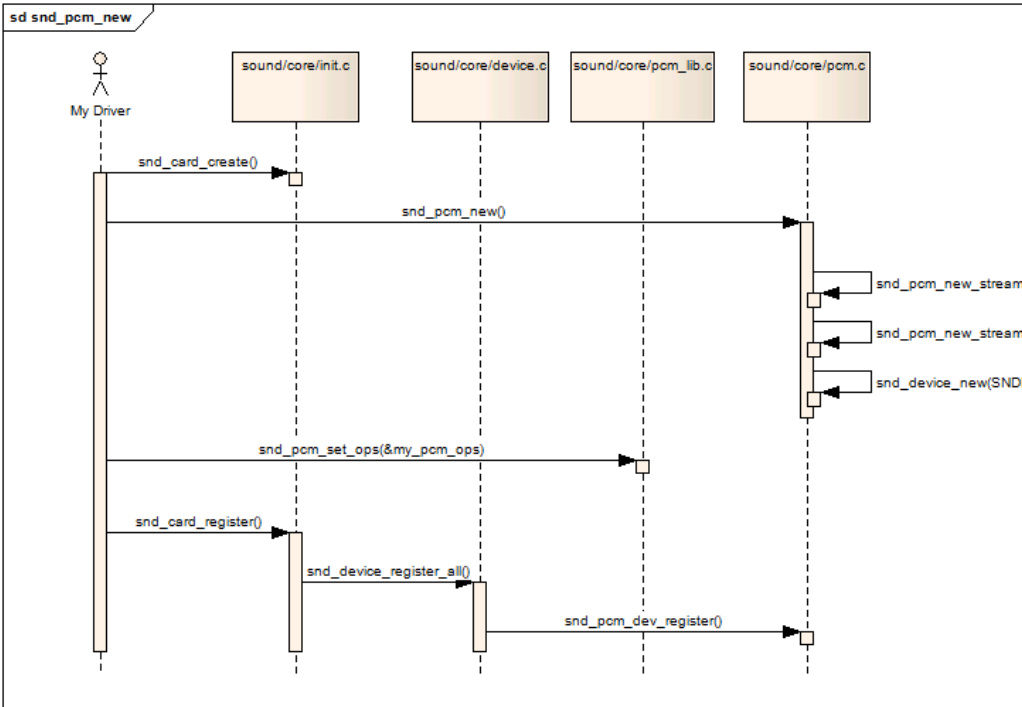


图3.1 新建pcm的序列图

- **snd_card_create** pcm是声卡下的一个设备（部件），所以第一步是要创建一个声卡
- **snd_pcm_new** 调用该api创建一个pcm，才该api中会做以下事情
 - 如果有，建立playback stream，相应的substream也同时建立
 - 如果有，建立capture stream，相应的substream也同时建立
 - 调用snd_device_new()把该pcm挂到声卡中，参数ops中的dev_register字段指向了函数snd_pcm_dev_register，这个回调函数会在声卡的注册阶段被调用。
- **snd_pcm_set_ops** 设置操作该pcm的控制/操作接口函数，参数中的snd_pcm_ops结构中的函数通常就是我们驱动要实现的函数
- **snd_card_register** 注册声卡，在这个阶段会遍历声卡下的所有逻辑设备，并且调用各设备的注册回调函数，对于pcm，就是第二步提到的snd_pcm_dev_register函数，该回调函数建立了和用户空间应用程序（alsa-lib）通信所用的设备文件节点:/dev/snd/pcmCxxDxxp和/dev/snd/pcmCxxDxxc

4. 设备文件节点的建立（dev/snd/pcmCxxDxxp、pcmCxxDxxc）

4.1 struct snd_minor

每个snd_minor结构体保存了声卡下某个逻辑设备的上下文信息，他在逻辑设备建立阶段被填充，在逻辑设备被使用时就可以从该结构体中得到相应的信息。pcm设备也不例外，也需要使用该结构体。该结构体在include/sound/core.h中定义。

```
[c-sharp]
01. struct snd_minor {
02.     int type;           /* SNDRV_DEVICE_TYPE_XXX */
03.     int card;           /* card number */
04.     int device;         /* device number */
05.     const struct file_operations *f_ops; /* file operations */
06.     void *private_data; /* private data for f_ops->open */
```

```

07.     struct device *dev;      /* device for sysfs */
08. };

```

在sound/sound.c中定义了一个snd_minor指针的全局数组:

```

[c-sharp]
01. static struct snd_minor *snd_minors[256];

```

前面说过, 在声卡的注册阶段 (snd_card_register), 会调用pcm的回调函数snd_pcm_dev_register(), 这个函数里会调用函数snd_register_device_for_dev():

```

[c-sharp]
01. static int snd_pcm_dev_register(struct snd_device *device)
02. {
03.     .....
04.
05.     /* register pcm */
06.     err = snd_register_device_for_dev(devtype, pcm->card,
07.                                       pcm->device,
08.                                       &snd_pcm_f_ops[cidx],
09.                                       pcm, str, dev);
10.     .....
11. }

```

我们再进入snd_register_device_for_dev():

```

[c-sharp]
01. int snd_register_device_for_dev(int type, struct snd_card *card, int dev,
02.                                const struct file_operations *f_ops,
03.                                void *private_data,
04.                                const char *name, struct device *device)
05. {
06.     int minor;
07.     struct snd_minor *preg;
08.
09.     if (snd_BUG_ON(!name))
10.         return -EINVAL;
11.     preg = kmalloc(sizeof *preg, GFP_KERNEL);
12.     if (preg == NULL)
13.         return -ENOMEM;
14.     preg->type = type;
15.     preg->card = card ? card->number : -1;
16.     preg->device = dev;
17.     preg->f_ops = f_ops;
18.     preg->private_data = private_data;
19.     mutex_lock(&sound_mutex);
20. #ifdef CONFIG_SND_DYNAMIC_MINORS
21.     minor = snd_find_free_minor();
22. #else
23.     minor = snd_kernel_minor(type, card, dev);
24.     if (minor >= 0 && snd_minors[minor])
25.         minor = -EBUSY;
26. #endif
27.     if (minor < 0) {
28.         mutex_unlock(&sound_mutex);
29.         kfree(preg);
30.         return minor;
31.     }
32.     snd_minors[minor] = preg;
33.     preg->dev = device_create(sound_class, device, MKDEV(major, minor),
34.                             private_data, "%s", name);
35.     if (IS_ERR(preg->dev)) {
36.         snd_minors[minor] = NULL;
37.         mutex_unlock(&sound_mutex);
38.         minor = PTR_ERR(preg->dev);
39.         kfree(preg);
40.         return minor;
41.     }
42.
43.     mutex_unlock(&sound_mutex);
44.     return 0;
45. }

```

- 首先, 分配并初始化一个snd_minor结构中的各字段

- type: SNDRV_DEVICE_TYPE_PCM_PLAYBACK/SNDRV_DEVICE_TYPE_PCM_CAPTURE
- card: card的编号
- device: pcm实例的编号, 大多数情况为0
- f_ops: snd_pcm_f_ops
- private_data: 指向该pcm的实例
- 根据type, card和pcm的编号, 确定数组的索引值minor, minor也作为pcm设备的此设备号
- 把该snd_minor结构的地址放入全局数组snd_minors[minor]中
- 最后, 调用device_create创建设备节点

4.2 设备文件的建立

在4.1节的最后, 设备文件已经建立, 不过4.1节的重点在于snd_minors数组的赋值过程, 在本节中, 我们把重点放在设备文件中。

回到pcm的回调函数snd_pcm_dev_register()中:

```
[c-sharp]
01. static int snd_pcm_dev_register(struct snd_device *device)
02. {
03.     int cidx, err;
04.     char str[16];
05.     struct snd_pcm *pcm;
06.     struct device *dev;
07.
08.     pcm = device->device_data;
09.     .....
10.     for (cidx = 0; cidx < 2; cidx++) {
11.         .....
12.         switch (cidx) {
13.             case SNDRV_PCM_STREAM_PLAYBACK:
14.                 sprintf(str, "pcmC%iD%i", pcm->card->number, pcm->device);
15.                 devtype = SNDRV_DEVICE_TYPE_PCM_PLAYBACK;
16.                 break;
17.             case SNDRV_PCM_STREAM_CAPTURE:
18.                 sprintf(str, "pcmC%iD%i", pcm->card->number, pcm->device);
19.                 devtype = SNDRV_DEVICE_TYPE_PCM_CAPTURE;
20.                 break;
21.         }
22.         /* device pointer to use, pcm->dev takes precedence if
23.          * it is assigned, otherwise fall back to card's device
24.          * if possible */
25.         dev = pcm->dev;
26.         if (!dev)
27.             dev = snd_card_get_device_link(pcm->card);
28.         /* register pcm */
29.         err = snd_register_device_for_dev(devtype, pcm->card,
30.                                           pcm->device,
31.                                           &snd_pcm_f_ops[cidx],
32.                                           pcm, str, dev);
33.         .....
34.     }
35.     .....
36. }
```

以上代码我们可以看出, 对于一个pcm设备, 可以生成两个设备文件, 一个用于playback, 一个用于capture, 代码中也确定了他们的命名规则:

- playback -- pcmCxDxp, 通常系统中只有一各声卡和一个pcm, 它就是pcmC0D0p
- capture -- pcmCxDxc, 通常系统中只有一各声卡和一个pcm, 它就是pcmC0D0c

snd_pcm_f_ops

snd_pcm_f_ops是一个标准的文件系统file_operations结构数组，它的定义在sound/core/pcm_native.c中：

```
[c-sharp]
01.  const struct file_operations snd_pcm_f_ops[2] = {
02.      {
03.          .owner =      THIS_MODULE,
04.          .write =      snd_pcm_write,
05.          .aio_write =  snd_pcm_aio_write,
06.          .open =       snd_pcm_playback_open,
07.          .release =    snd_pcm_release,
08.          .llseek =     no_llseek,
09.          .poll =       snd_pcm_playback_poll,
10.          .unlocked_ioctl = snd_pcm_playback_ioctl,
11.          .compat_ioctl = snd_pcm_ioctl_compat,
12.          .mmap =       snd_pcm_mmap,
13.          .fasync =     snd_pcm_fasync,
14.          .get_unmapped_area = snd_pcm_get_unmapped_area,
15.      },
16.      {
17.          .owner =      THIS_MODULE,
18.          .read =       snd_pcm_read,
19.          .aio_read =   snd_pcm_aio_read,
20.          .open =       snd_pcm_capture_open,
21.          .release =    snd_pcm_release,
22.          .llseek =     no_llseek,
23.          .poll =       snd_pcm_capture_poll,
24.          .unlocked_ioctl = snd_pcm_capture_ioctl,
25.          .compat_ioctl = snd_pcm_ioctl_compat,
26.          .mmap =       snd_pcm_mmap,
27.          .fasync =     snd_pcm_fasync,
28.          .get_unmapped_area = snd_pcm_get_unmapped_area,
29.      }
30.  };
```

snd_pcm_f_ops作为snd_register_device_for_dev的参数被传入，并被记录在snd_minors[minor]中的字段f_ops中。最后，在snd_register_device_for_dev中创建设备节点：

```
[c-sharp]
01.  snd_minors[minor] = preg;
02.  preg->dev = device_create(sound_class, device, MKDEV(major, minor),
03.                          private_data, "%s", name);
```

4.3 层层深入，从应用程序到驱动层pcm

4.3.1 字符设备注册

在sound/core/sound.c中有alsa_sound_init()函数，定义如下：

```
[c-sharp]
01.  static int __init alsa_sound_init(void)
02.  {
03.      snd_major = major;
04.      snd_ecards_limit = cards_limit;
05.      if (register_chrdev(major, "alsa", &snd_fops)) {
06.          snd_printk(KERN_ERR "unable to register native major device number %d/n", major);
07.          return -EIO;
08.      }
09.      if (snd_info_init() < 0) {
10.          unregister_chrdev(major, "alsa");
11.          return -ENOMEM;
12.      }
13.      snd_info_minor_register();
14.      return 0;
15.  }
```

register_chrdev中的参数major与之前创建pcm设备是device_create时的major是同一个，这样的结果是，当应用程序open设备文件/dev/snd/pcmCxDxp时，会进入snd_fops的open回调函数，我们将在下一节中讲述open的过程。

4.3.2 打开pcm设备

从上一节中我们得知，`open`一个pcm设备时，将会调用`snd_fops`的`open`回调函数，我们先看看`snd_fops`的定义：

```
[c-sharp]
01. static const struct file_operations snd_fops =
02. {
03.     .owner =    THIS_MODULE,
04.     .open =    snd_open
05. };
```

跟入`snd_open`函数，它首先从`inode`中取出此设备号，然后以次设备号为索引，从`snd_minors`全局数组中取出当初注册pcm设备时填充的`snd_minor`结构（参看4.1节的内容），然后从`snd_minor`结构中取出pcm设备的`f_ops`，并且把`file->f_op`替换为pcm设备的`f_ops`，紧接着直接调用pcm设备的`f_ops->open()`，然后返回。因为`file->f_op`已经被替换，以后，应用程序的所有`read/write/ioctl`调用都会进入pcm设备自己的回调函数中，也就是4.2节中提到的`snd_pcm_f_ops`结构中定义的回调。

```
[c-sharp]
01. static int snd_open(struct inode *inode, struct file *file)
02. {
03.     unsigned int minor = iminor(inode);
04.     struct snd_minor *mptr = NULL;
05.     const struct file_operations *old_fops;
06.     int err = 0;
07.
08.     if (minor >= ARRAY_SIZE(snd_minors))
09.         return -ENODEV;
10.     mutex_lock(&sound_mutex);
11.     mptr = snd_minors[minor];
12.     if (mptr == NULL) {
13.         mptr = autoload_device(minor);
14.         if (!mptr) {
15.             mutex_unlock(&sound_mutex);
16.             return -ENODEV;
17.         }
18.     }
19.     old_fops = file->f_op;
20.     file->f_op = fops_get(mptr->f_ops);
21.     if (file->f_op == NULL) {
22.         file->f_op = old_fops;
23.         err = -ENODEV;
24.     }
25.     mutex_unlock(&sound_mutex);
26.     if (err < 0)
27.         return err;
28.
29.     if (file->f_op->open) {
30.         err = file->f_op->open(inode, file);
31.         if (err) {
32.             fops_put(file->f_op);
33.             file->f_op = fops_get(old_fops);
34.         }
35.     }
36.     fops_put(old_fops);
37.     return err;
38. }
```

下面的序列图展示了应用程序如何最终调用到`snd_pcm_f_ops`结构中的回调函数：

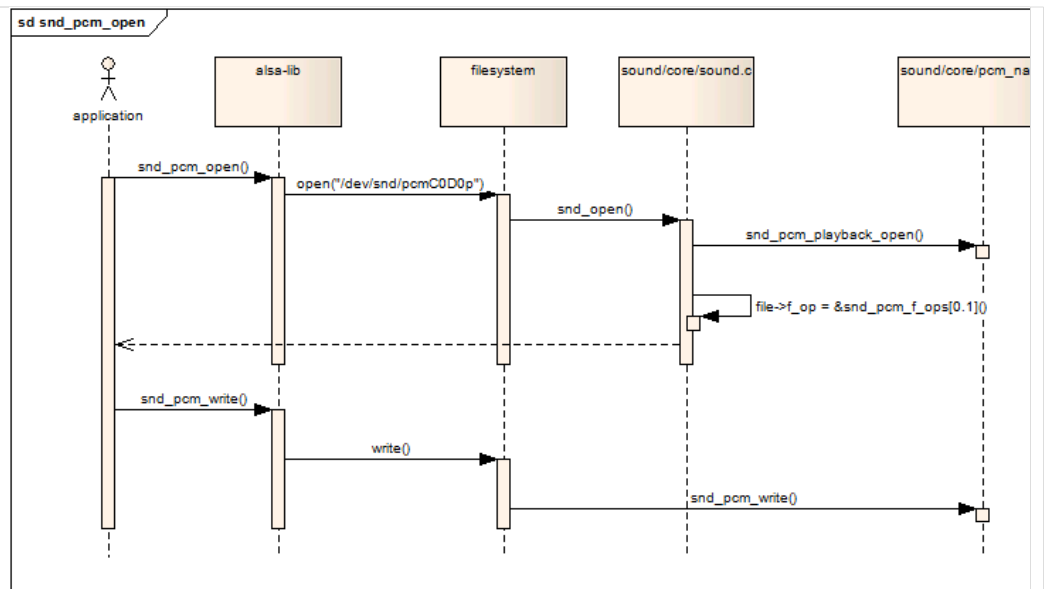


图4.3.2.1 应用程序操作pcm设备

更多 1

上一篇: [Linux ALSA声卡驱动之二: 声卡的创建](#)

下一篇: [Linux ALSA声卡驱动之四: Control设备的创建](#)

查看评论

15楼 [hello_xiaowen](#) 2013-11-11 11:45发表



这是我看过的关于ALSA最好的文章

14楼 [BBQLOVEYOU](#) 2013-07-02 10:00发表



牛人!

13楼 [裂空](#) 2013-06-17 11:54发表



赞一个!

12楼 [fjbwinston](#) 2013-06-04 21:37发表



顶一个!!!

11楼 [vito_coleone](#) 2013-05-26 20:20发表



真的讲的很好, 十分感谢。

10楼 [kuangreng](#) 2012-09-21 15:59发表



第二遍阅读, 十分感谢!

9楼 [dongshengzou95](#) 2012-08-23 17:27发表



看到这里我已经忍不住要说两句了, 楼主能把这么复杂的alsa驱动讲解的这么清楚, 思路如此清晰, 佩服! 我之前好几个疑惑, 都在这里能找到精确的答案, 感激!

8楼 [kuangreng](#) 2012-05-24 12:04发表



正在学习, 讲得非常好, 十分感谢!!!

7楼 [A_VS_Z](#) 2012-03-15 16:13发表



的确好,

6楼 阿曼 2011-12-14 22:41发表



真心佩服博主。不仅自己看懂了，还写出来让更过的人看懂，真心感谢。

5楼 stone10000y 2011-06-22 11:37发表



在你的Linux音频驱动之二
里面找到答案了，不好意思，谢谢

4楼 stone10000y 2011-06-22 11:16发表



赞！ 有个问题请教下：创建好的设备文件为什么是在/dev/snd下，不是直接就在/dev下呢？

3楼 ustcxiangchun 2011-06-08 17:14发表



有个疑问请教一下，后面具体会转到哪个芯片的驱动去又是怎么样的呢？期待楼主也讲讲这个

Re: DroidPhone 2011-06-09 12:37发表



回复 ustcxiangchun: 对于嵌入式设备，具体到芯片级的要在soc层实现，以后写到soc时才会涉及。

2楼 ustcxiangchun 2011-06-08 17:12发表



写得非常的好呀，赞一个

1楼 sxjyx2009 2011-04-28 09:17发表



[e03]持续关注中

您还没有登录,请[\[登录\]](#)或[\[注册\]](#)

* 以上用户言论只代表其个人观点，不代表CSDN网站的观点或立场

专区推荐内容

HTML5应用性能调优工具WAP...
Android应用开发
NDK 安卓应用移植方法
Struts2 高危漏洞修复方案
HTML5 经典小游戏之坦克
新用户编译器基本用法



更多招聘职位

我公司职位也要出现在这里

【广州友魄信息科技有限公司】Asp.net (C#) 开发工程师
【北京奥鹏远程教育中心有限公司】专职/兼职IT培训讲师
(Web前端开发, PHP, 3G开发方向)
【langoon】java架构师
【北京富邦天成信息技术有限公司】java高级工程师
【南京科莱斯企业管理咨询有限公司】Flex开发工程师

核心技术类目

全部主题 Java VPN Android iOS ERP IE10 Eclipse CRM JavaScript Ubuntu NFC
WAP jQuery 数据库 BI HTML5 Spring Apache Hadoop .NET API HTML SDK IIS
Fedora XML LBS Unity Splashtop UML components Windows Mobile Rails QEMU KDE
Cassandra CloudStack FTC coremail OPhone CouchBase 云计算 iOS6 Rackspace
Web App SpringSide Maemo Compuware 大数据 aptech Perl Tomado Ruby Hibernate
ThinkPHP Spark HBase Pure Solr Angular Cloud Foundry Redis Scala Django
Bootstrap

公司简介 | 招贤纳士 | 广告服务 | 银行汇款帐号 | 联系方式 | 版权声明 | 法律顾问 | 问题报告

QQ客服 微博客服 论坛反馈 联系邮箱: webmaster@csdn.net 服务热线: 400-600-2320

京 ICP 证 070598 号

北京创新乐知信息技术有限公司 版权所有

世纪乐知(北京)网络技术有限公司 提供技术支持

江苏乐知网络技术有限公司 提供商务支持

Copyright © 1999-2012, CSDN.NET, All Rights Reserved



Linux ALSA声卡驱动之三	(18248)
Android中的sp和wp指针	(17112)
Linux ALSA声卡驱动之七	(13786)
Android SurfaceFlinger	(13061)
	(12550)

评论排行	
Android Audio System 之	(49)
Linux ALSA声卡驱动之八	(30)
Android SurfaceFlinger	(21)
Linux ALSA声卡驱动之二	(18)
Linux ALSA声卡驱动之三	(16)
Android Audio System 之	(16)
Linux中断（interrupt）子	(15)
Android中的sp和wp指针	(13)
Linux中断（interrupt）子	(12)
Android SurfaceFlinger	(11)

推荐文章	
* SharePoint 2010/2013 使用 Javascript来判断权限的三种方法	
* 坚持前进的方向：总结 2013，规划2014	
* 创业者那些鲜为人知的事情	
* ListView具有多种item布局——实现微信对话列	
* 实现自己的类加载时，重写方法loadClass与findClass的区别	
* GDAL影像投影转换	

最新评论	
Linux输入子系统：多点触控协议 gocy123: 很有用，多谢分享	
ALSA声卡驱动中的DAPM详解之 wsc_168: 楼主,您好： 现在正在移植wm8962的驱动，遇到了一些问题，请教一下您。串口信息显示已经扫描...	
Linux ALSA声卡驱动之五：移动i slcsss: @DroidPhone:感谢您的回复，我是新手，想问下这个配置的具体位置在哪里？	
Linux ALSA声卡驱动之五：移动i DroidPhone: @u013222557:这种情况通常是你的codec中的音频路径把Mic至HP的路径被打开了，请仔细...	
Linux ALSA声卡驱动之五：移动i slcsss: 您好： 我是一名在读研究生，最近在ALSA架构下搞嵌入式音频程序开发，遇到了一个棘手的问题...	
ALSA声卡驱动中的DAPM详解之 DroidPhone: @u012389631:和电源管理和音频路径相关的control需要定义为dapm control (...	
ALSA声卡驱动中的DAPM详解之 elliepfang: 大侠，你好！ 这两天把您的文章1-7 看了一遍，关于control这个概念在您的文章中有提到过多次...	
ALSA声卡驱动中的DAPM详解之 elliepfang: @DroidPhone:因为这个是wm8962的machine上的现有代码，但如果我要porting...	

iface字段指出了control的类型，alsa定义了几种类型（SNDDRV_CTL_ELEM_IFACE_XXX），常用的类型是MIXER，当然也可以定义属于全局的CARD类型，也可以定义属于某类设备的类型，例如HWDEP, PCMRawMIDI, TIMER等，这时需要在device和subdevice字段中指出卡的设备逻辑编号。

name字段是该control的名字，从ALSA 0.9.x开始，control的名字是变得比较重要，因为control的作用是按名字来归类的。ALSA已经预定义了一些control的名字，我们再Control Name一节详细讨论。

index字段用于保存该control的在该卡中的编号。如果声卡中有不止一个codec，每个codec中有相同名字的control，这时我们可以通过index来区分这些controls。当index为0时，则可以忽略这种区分策略。

access字段包含了该control的访问类型。每一个bit代表一种访问类型，这些访问类型可以多个“或”运算组合在一起。

private_value字段包含了一个任意的长整数类型值。该值可以通过info，get，put这几个回调函数访问。你可以自己决定如何使用该字段，例如可以把它拆分成多个位域，又或者是一个指针，指向某一个数据结构。

tlv字段为该control提供元数据。

Control的名字

control的名字需要遵循一些标准，通常可以分成3部分来定义control的名字：源--方向--功能。

- 源，可以理解为该control的输入端，alsa已经预定义了一些常用的源，例如：Master, PCM, CD, Line等等。
- 方向，代表该control的数据流向，例如：Playback, Capture, Bypass, Bypass Capture等等，也可以不定义方向，这时表示该Control是双向的（playback和capture）。
- 功能，根据control的功能，可以是以下字符串：Switch, Volume, Route等等。

也有一些命名上的特例：

- 全局的capture和playback "Capture Source", "Capture Volume", "Capture Switch", 它们用于全局的capture source, switch和volume。同理, "Playback Volume", "Playback Switch", 它们用于全局的输出switch和volume。
- Tone-controls 音调控制的开关和音量命名为：Tone Control - XXX, 例如, "Tone Control - Switch", "Tone Control - Bass", "Tone Control - Center"。
- 3D controls 3D控件的命名规则：, "3D Control - Switch", "3D Control - Center", "3D Control - Space"。
- Mic boost 麦克风音量加强控件命名为: "Mic Boost"或"Mic Boost(6dB)"。

访问标志（ACCESS Flags）

Access字段是一个bitmask，它保存了改control的访问类型。默认的访问类型

是：SNDDRV_CTL_ELEM_ACCESS_READWRITE，表明该control支持读和写操作。如果access字段没有定义（.access==0），此时也认为是READWRITE类型。

如果是一个只读control，access应该设置为：SNDDRV_CTL_ELEM_ACCESS_READ，这时，我们不必定义put回调函数。类似地，如果是只写control，access应该设置为：SNDDRV_CTL_ELEM_ACCESS_WRITE，这时，我们不必定义get回调函数。

Linux ALSA声卡驱动之六: ASoC
elliepsang: @DroidPhone:关于
snd_soc_dai_link结构中的代码
注释中struct snd_...

ALSA声卡驱动中的DAPM详解之
DroidPhone: @u012389631:这
种名字根据实际的意义自己定义
就好了, 只要符合常识即可。不
过通常还是会和co...

如果control的值会频繁地改变(例如: 电平表), 我们可以使用VOLATILE类型, 这意味着该control会在没有通知的情况下改变, 应用程序应该定时地查询该control的值。

回调函数

info回调函数

info回调函数用于获取control的详细信息。它的主要工作就是填充通过参数传入的snd_ctl_elem_info对象, 以下例子是一个具有单个元素的boolean型control的info回调:

```
[c-sharp]
01. static int snd_myctl_mono_info(struct snd_kcontrol *kcontrol,
02.     struct snd_ctl_elem_info *uinfo)
03. {
04.     uinfo->type = SNDRV_CTL_ELEM_TYPE_BOOLEAN;
05.     uinfo->count = 1;
06.     uinfo->value.integer.min = 0;
07.     uinfo->value.integer.max = 1;
08.     return 0;
09. }
```

type字段指出该control的值类型, 值类型可以是BOOLEAN, INTEGER, ENUMERATED, BYTES,IEC958和INTEGER64之一。count字段指出了改control中包含有多少个元素单元, 比如, 立体声的音量control左右两个声道的音量值, 它的count字段等于2。value字段是一个联合体(union), value的内容和control的类型有关。其中, boolean和integer类型是相同的。

ENUMERATED类型有些特殊。它的value需要设定一个字符串和字符串的索引, 请看以下例子:

```
[c-sharp]
01. static int snd_myctl_enum_info(struct snd_kcontrol *kcontrol,
02.     struct snd_ctl_elem_info *uinfo)
03. {
04.     static char *texts[4] = {
05.         "First", "Second", "Third", "Fourth"
06.     };
07.     uinfo->type = SNDRV_CTL_ELEM_TYPE_ENUMERATED;
08.     uinfo->count = 1;
09.     uinfo->value.enumerated.items = 4;
10.     if (uinfo->value.enumerated.item > 3)
11.         uinfo->value.enumerated.item = 3;
12.     strcpy(uinfo->value.enumerated.name,
13.         texts[uinfo->value.enumerated.item]);
14.     return 0;
15. }
```

alsa已经为我们实现了一些通用的info回调函数, 例

如: snd_ctl_boolean_mono_info(), snd_ctl_boolean_stereo_info()等等。

get回调函数

该回调函数用于读取control的当前值, 并返回给用户空间的应用程序。

```
[c-sharp]
01. static int snd_myctl_get(struct snd_kcontrol *kcontrol,
02.     struct snd_ctl_elem_value *ucontrol)
03. {
04.     struct mychip *chip = snd_kcontrol_chip(kcontrol);
05.     ucontrol->value.integer.value[0] = get_some_value(chip);
06.     return 0;
07. }
```

value字段的赋值依赖于**control**的类型（如同**info**回调）。很多声卡的驱动利用它存储硬件寄存器的地址、**bit-shift**和**bit-mask**，这时，**private_value**字段可以按以下例子进行设置：

```
.private_value = reg | (shift << 16) | (mask << 24);
```

然后，**get**回调函数可以这样实现：

```
static int snd_sbmixer_get_single(struct snd_kcontrol *kcontrol,
    struct snd_ctl_elem_value *ucontrol)

{
    int reg = kcontrol->private_value & 0xff;
    int shift = (kcontrol->private_value >> 16) & 0xff;
    int mask = (kcontrol->private_value >> 24) & 0xff;
    ....

    //根据以上的值读取相应寄存器的值并填入value中
}
```

如果**control**的**count**字段大于1，表示**control**有多个元素单元，**get**回调函数也应该为**value**填充多个数值。

put回调函数

put回调函数用于把应用程序的控制值设置到**control**中。

```
[c-sharp]
01. static int snd_myctl_put(struct snd_kcontrol *kcontrol,
02.     struct snd_ctl_elem_value *ucontrol)
03. {
04.     struct mychip *chip = snd_kcontrol_chip(kcontrol);
05.     int changed = 0;
06.     if (chip->current_value !=
07.         ucontrol->value.integer.value[0]) {
08.         change_current_value(chip,
09.             ucontrol->value.integer.value[0]);
10.         changed = 1;
11.     }
12.     return changed;
13. }
```

如上述例子所示，当**control**的值被改变时，**put**回调必须要返回1，如果值没有被改变，则返回0。如果发生了错误，则返回一个负数的错误号。

和**get**回调一样，当**control**的**count**大于1时，**put**回调也要处理多个**control**中的元素值。

创建Controls

当把以上讨论的内容都准备好了以后，我们就可以创建我们自己的**control**了。**alsa-driver**为我们提供了两个用于创建**control**的API：

- **snd_ctl_new1()**
- **snd_ctl_add()**

我们可以用以下最简单的方式创建**control**：


```
[c-sharp]
```

```
01.     err = snd_ctl_add(card, snd_ctl_new1(&my_control, chip));
02.     if (err < 0)
03.         return err;
```

在这里，`my_control`是一个之前定义好的`snd_kcontrol_new`对象，`chip`对象将会被赋值在`kcontrol->private_data`字段，该字段可以在回调函数中访问。

`snd_ctl_new1()`会分配一个新的`snd_kcontrol`实例，并把`my_control`中相应的值复制到该实例中，所以，在定义`my_control`时，通常我们可以加上`__devinitdata`前缀。`snd_ctl_add`则把该`control`绑定到声卡对象`card`当中。

元数据（Metadata）

很多mixer control需要提供以dB为单位的信息，我们可以使用`DECLARE_TLV_xxx`宏来定义一些包含这种信息的变量，然后把`control`的`tlv.p`字段指向这些变量，最后，在`access`字段中加上`SNDRV_CTL_ELEM_ACCESS_TLV_READ`标志，就像这样：

```
static DECLARE_TLV_DB_SCALE(db_scale_my_control, -4050, 150, 0);
```

```
static struct snd_kcontrol_new my_control __devinitdata = {
    ...
    .access = SNDRV_CTL_ELEM_ACCESS_READWRITE |
              SNDRV_CTL_ELEM_ACCESS_TLV_READ,
    ...
    .tlv.p = db_scale_my_control,
};
```

`DECLARE_TLV_DB_SCALE`宏定义的mixer control，它所代表的值按一个固定的dB值的步长变化。该宏的第一个参数是要定义变量的名字，第二个参数是最小值，以0.01dB为单位。第三个参数是变化的步长，也是以0.01dB为单位。如果该control处于最小值时会做出mute时，需要把第四个参数设为1。

`DECLARE_TLV_DB_LINEAR`宏定义的mixer control，它的输出随值的变化而线性变化。该宏的第一个参数是要定义变量的名字，第二个参数是最小值，以0.01dB为单位。第二个参数是最大值，以0.01dB为单位。如果该control处于最小值时会做出mute时，需要把第二个参数设为`TLV_DB_GAIN_MUTE`。

这两个宏实际上就是定义一个整形数组，所谓tlv，就是Type-Lenght-Value的意思，数组的第0各元素代表数据的类型，第1个元素代表数据的长度，第三个元素和之后的元素保存该变量的数据。

Control设备的建立

Control设备和PCM设备一样，都属于声卡下的逻辑设备。用户空间的应用程序通过`alsa-lib`访问该Control设备，读取或控制control的控制状态，从而达到控制音频Codec进行各种Mixer等控制操作。

Control设备的创建过程大体上和PCM设备的创建过程相同。详细的创建过程可以参考本博的另一篇文章：[Linux 音频驱动之三：PCM设备的创建](#)。下面我们只讨论有区别的地方。

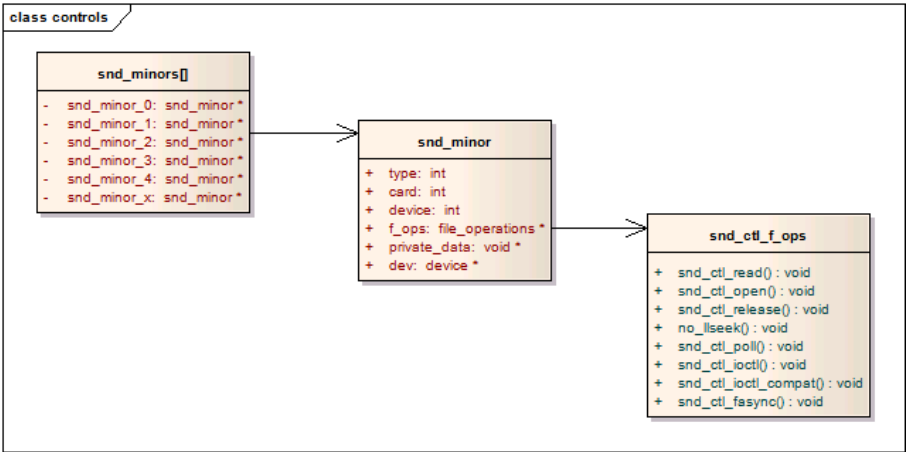
我们需要在我们的驱动程序初始化时主动调用`snd_pcm_new()`函数创建pcm设备，而control设备则在

snd_card_create()内被创建，snd_card_create()通过调用snd_ctl_create()函数创建control设备节点。所以我们无需显式地创建control设备，只要建立声卡，control设备被自动地创建。

和pcm设备一样，control设备的名字遵循一定的规则：controlCxx，这里的xx代表声卡的编号。我们也可以通过代码正是这一点，下面的是snd_ctl_dev_register()函数的代码：

```
[c-sharp]
01.  /*
02.   * registration of the control device
03.   */
04.  static int snd_ctl_dev_register(struct snd_device *device)
05.  {
06.      struct snd_card *card = device->device_data;
07.      int err, cardnum;
08.      char name[16];
09.
10.      if (snd_BUG_ON(!card))
11.          return -ENXIO;
12.      cardnum = card->number;
13.      if (snd_BUG_ON(cardnum < 0 || cardnum >= SNDRV_CARDS))
14.          return -ENXIO;
15.      /* control设备的名字 */
16.      sprintf(name, "controlC%i", cardnum);
17.      if ((err = snd_register_device(SNDRV_DEVICE_TYPE_CONTROL, card, -1,
18.                                   &snd_ctl_f_ops, card, name)) < 0)
19.          return err;
20.      return 0;
21.  }
```

snd_ctl_dev_register()函数会在snd_card_register()中，即声卡的注册阶段被调用。注册完成后，control设备的相关信息被保存在snd_minors[]数组中，用control设备的此设备号作索引，即可在snd_minors[]数组中找出相关的信息。注册完成后的数据结构关系可以用下图进行表述：



control设备的操作函数入口

用户程序需要打开control设备时，驱动程序通过snd_minors[]全局数组和此设备号，可以获得snd_ctl_f_ops结构中的各个回调函数，然后通过这些回调函数访问control中的信息和数据（最终会调用control的几个回调函数get, put, info）。详细的代码我就不贴了，大家可以读一下代码：/sound/core/control.c。

查看评论

8楼 [xiaoyaosuifeng_2012](#) 2013-07-24 17:21发表



写的相当好！最近我在研究Andriod的ALSA结构，看了你的文章后获益匪浅，非常感谢！

7楼 [ziwuchen](#) 2013-02-17 14:37发表



写得很有条理，获益匪浅！

6楼 [Hens007](#) 2013-01-16 10:06发表



确实写得很深入

5楼 [waitlongbj](#) 2012-12-11 20:02发表



高手,一直都是看你的文章学习

4楼 [wanglining1987](#) 2012-05-07 17:55发表



这是我看过的最好的了，非常感谢了

3楼 [rnmichelle](#) 2011-12-03 16:36发表



强大!赞!能不能把HAL调ALSA的过程再分析下就更好了.期待高手的精彩文章.

2楼 [TXPG2001](#) 2011-07-14 23:01发表



期待下一篇

1楼 [qtandroid](#) 2011-06-07 21:42发表



你是高手啊~。~！这里我看了好久了，这篇文章虽然没有trace callbacks，但是写得很好了，而且还有uml图，不错不错~~！持续关注中！！

您还没有登录,请[\[登录\]](#)或[\[注册\]](#)

* 以上用户言论只代表其个人观点，不代表CSDN网站的观点或立场

专区推荐内容

介绍一种服务器缓存结构-多级 H...
一次编写，随处运行 英特尔HTM...
开源 - 开放式虚拟化解决方案
绳命不息，代码不止
跟燕青一起学Android应用开...
在英特尔凌动平台上进行Andro...



更多招聘职位

我公司职位也要出现在这里

【哈票网络科技（北京）有限公司】Java研发组组长
【万维嘉信（山东）电子商务有限公司】高级运维经理
【广州友魄信息科技有限公司】Asp.net（C#）开发工程师
【北京奥鹏远程教育中心有限公司】专职/兼职IT培训讲师（Web前端开发，PHP，3G开发方向）
【langoon】.java架构师

核心技术类目

全部主题 Java VPN Android iOS ERP IE10 Eclipse CRM JavaScript Ubuntu NFC
WAP jQuery 数据库 BI HTML5 Spring Apache Hadoop .NET API HTML SDK IIS
Fedora XML LBS Unity Splashtop UML components Windows Mobile Rails QEMU KDE
Cassandra CloudStack FTC coremail OPhone CouchBase 云计算 iOS6 Rackspace
Web App SpringSide Maemo Compuware 大数据 aptech Perl Tornado Ruby Hibernate
ThinkPHP Spark HBase Pure Solr Angular Cloud Foundry Redis Scala Django
Bootstrap

公司简介 | 招贤纳士 | 广告服务 | 银行汇款帐号 | 联系方式 | 版权声明 | 法律顾问 | 问题报告

QQ客服 微博客服 论坛反馈 联系邮箱: webmaster@csdn.net 服务热线: 400-600-2320

京 ICP 证 070598 号

北京创新乐知信息技术有限公司 版权所有

世纪乐知(北京)网络技术有限公司 提供技术支持

江苏乐知网络技术有限公司 提供商务支持

Copyright © 1999-2012, CSDN.NET, All Rights Reserved



DroidPhone的专栏

欢迎各位大虾交流，本人联系方式：droid.phx@gmail.com

目录视图

摘要视图

RSS 订阅

个人资料



DroidPhone

访问：382343次

积分：3434分

排名：第2116名

原创：46篇

转载：0篇

译文：4篇

评论：356条

文章搜索

文章分类

[移动开发之Android \(11\)](#)
[Linux内核架构 \(15\)](#)
[Linux设备驱动 \(16\)](#)
[Linux电源管理 \(3\)](#)
[Linux音频子系统 \(15\)](#)
[Linux中断子系统 \(5\)](#)
[Linux时间管理系统 \(8\)](#)
[Linux输入子系统 \(4\)](#)

文章存档

[2013年11月 \(4\)](#)
[2013年10月 \(3\)](#)
[2013年07月 \(3\)](#)
[2012年12月 \(4\)](#)
[2012年10月 \(4\)](#)

展开

阅读排行

[Android Audio System 之 \(38982\)](#)
[Android Audio System 之 \(25553\)](#)
[Android Audio System 之 \(25317\)](#)
[Linux ALSA声卡驱动之一 \(24001\)](#)
[Linux ALSA声卡驱动之二 \(18421\)](#)
[Android SurfaceFlinger \(4\)](#)

投票赢好礼，周周有惊喜！

2014年4月微软MVP申请开始了！

消灭0回答，赢下载分

“我的2013”年度征文活动火爆进行中！

办公大师系列经典丛书 诚聘英才

Linux ALSA声卡驱动之五：移动设备中的ALSA (ASoC)

分类：Linux设备驱动 Linux音频子系统
 2012-01-17 14:16
 10045人阅读
 评论(6)
 收藏
 举报

linux

codec

数据结构

嵌入式

工作

平台

目录(?)

[-]

1. ASoC的由来
 2. 硬件架构
 3. 软件架构
 4. 数据结构
 5. 版内核对ASoC的改进

1. ASoC的由来

ASoC--ALSA System on Chip，是建立在标准ALSA驱动层上，为了更好地支持嵌入式处理器和移动设备中的音频Codec的一套软件体系。在ASoc出现之前，内核对于SoC中的音频已经有部分的支持，不过会有一些局限性：

- Codec驱动与SoC CPU的底层耦合过于紧密，这种不理想会导致代码的重复，例如，仅是wm8731的驱动，当时Linux中有分别针对4个平台的驱动代码。
- 音频事件没有标准的方法来通知用户，例如耳机、麦克风的插拔和检测，这些事件在移动设备中是非常普通的，而且通常都需要特定于机器的代码进行重新对音频路劲进行配置。
- 当进行播放或录音时，驱动会让整个codec处于上电状态，这对于PC没问题，但对于移动设备来说，这意味着浪费大量的电量。同时也不支持通过改变过取样频率和偏置电流来达到省电的目的。

ASoC正是为了解决上述种种问题而提出的，目前已经被整合至内核的代码树中：sound/soc。ASoC不能单独存在，他只是建立在标准ALSA驱动上的一个它必须和标准的ALSA驱动框架相结合才能工作。

```

/*****
声明：本博内容均由http://blog.csdn.net/droidphone原创，转载请注明出处，谢谢！
*****/

```

2. 硬件架构

通常，就像软件领域里的抽象和重用一样，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，如下图所示：



图2.1 音频系统结构

- Machine** 是指某一款机器，可以是某款设备，某款开发板，又或者是某款智能手机，由此可以看出Machine几乎是不可重用的，每个Machine上的硬件实现可能都不一样，CPU不一样，Codec不一样，音频

Linux ALSA声卡驱动之三	(18248)
Android中的sp和wp指针	(17112)
Linux ALSA声卡驱动之七	(13786)
Android SurfaceFlinger什	(13061)
	(12550)

评论排行	
Android Audio System 之	(49)
Linux ALSA声卡驱动之八	(30)
Android SurfaceFlinger什	(21)
Linux ALSA声卡驱动之二	(18)
Linux ALSA声卡驱动之三	(16)
Android Audio System 之	(16)
Linux中断（interrupt）子	(15)
Android中的sp和wp指针	(13)
Linux中断（interrupt）子	(12)
Android SurfaceFlinger什	(11)

推荐文章	
* SharePoint 2010/2013 使用 Javascript来判断权限的三种方法	
* 坚持前进的方向：总结 2013，规划2014	
* 创业者那些鲜为人知的事情	
* ListView具有多种item布局——实现微信对话列	
* 实现自己的类加载时，重写方法loadClass与findClass的区别	
* GDAL影像投影转换	

最新评论	
Linux输入子系统：多点触控协议 gocy123 : 很有用，多谢分享	
ALSA声卡驱动中的DAPM详解之 wsc_168 : 楼主,您好： 现在正在移植wm8962的驱动，遇到了一些问题，请教一下您。串口信息显示已经扫描...	
Linux ALSA声卡驱动之五：移动i slcsss : @DroidPhone:感谢您的回复，我是新手，想问下这个配置的具体位置在哪里？	
Linux ALSA声卡驱动之五：移动i DroidPhone : @u013222557:这种情况通常是你的codec中的音频路径把Mic至HP的路径被打开了，请仔细...	
Linux ALSA声卡驱动之五：移动i slcsss : 您好： 我是一名在读研究生，最近在ALSA架构下搞嵌入式音频程序开发，遇到了一个棘手的问题...	
ALSA声卡驱动中的DAPM详解之 DroidPhone : @u012389631:和电源管理和音频路径相关的control需要定义为dapm control（...	
ALSA声卡驱动中的DAPM详解之 elliepfang : 大侠，你好！ 这两天把您的文章1-7 看了一遍，关于control这个概念在您的文章中有提到过多次...	
ALSA声卡驱动中的DAPM详解之 elliepfang : @DroidPhone:因为这个是wm8962的machine上的现有代码，但如果我要porting...	

的输入、输出设备也不一样，Machine为CPU、Codec、输入输出设备提供了一个载体。

- **Platform** 一般是指某一个SoC平台，比如pxxxx,s3cxxx,omapxxx等等，与音频相关的通常包含该SoC中的时钟、DMA、I2S、PCM等等，只要指定了SoC，那么我们可以认为它会有一个对应的Platform，它只与SoC相关，与Machine无关，这样我们就可以把Platform抽象出来，使得同一款SoC不用做任何的改动，就可以用在不同的Machine中。实际上，把Platform认为是某个SoC更好理解。
- **Codec** 字面上的意思就是编解码器，Codec里面包含了I2S接口、D/A、A/D、Mixer、PA（功放），通常包含多种输入（Mic、Line-in、I2S、PCM）和多个输出（耳机、喇叭、听筒，Line-out），Codec和Platform一样，是可重用的部件，同一个Codec可以被不同的Machine使用。嵌入式Codec通常通过I2C对内部的寄存器进行控制。

3. 软件架构

在软件层面，ASoC也把嵌入式设备的音频系统同样分为3大部分，Machine，Platform和Codec。

- **Codec驱动** ASoC中的一个重要设计原则就是要求Codec驱动是平台无关的，它包含了一些音频的控件（Controls），音频接口，DAMP（动态音频电源管理）的定义和某些Codec IO功能。为了保证硬件无关性，任何特定于平台和机器的代码都要移到Platform和Machine驱动中。所有的Codec驱动都要提供以下特性：
 - Codec DAI 和 PCM的配置信息；
 - Codec的IO控制方式（I2C，SPI等）；
 - Mixer和其他的音频控件；
 - Codec的ALSA音频操作接口；

必要时，也可以提供以下功能：

- DAPM描述信息；
- DAPM事件处理程序；
- DAC数字静音控制

- **Platform驱动** 它包含了该SoC平台的音频DMA和音频接口的配置和控制（I2S，PCM，AC97等等）；它也不能包含任何与板子或机器相关的代码。
- **Machine驱动** Machine驱动负责处理机器特有的一些控件和音频事件（例如，当播放音频时，需要先行打开一个放大器）；单独的Platform和Codec驱动是不能工作的，它必须由Machine驱动把它们结合在一起才能完成整个设备的音频处理工作。

4. 数据结构

整个ASoC是由一些列数据结构组成，要搞清楚ASoC的工作机理，必须要理解这一系列数据结构之间的关系和作用，下面的关系图展示了ASoC中重要的数据结构之间的关联方式：

Linux ALSA声卡驱动之六: ASoC
elliepsang: @DroidPhone:关于
snd_soc_dai_link结构中的代码
注释中struct snd_...
ALSA声卡驱动中的DAPM详解之
DroidPhone: @u012389631:这
种名字根据实际的意义自己定义
就好了, 只要符合常识即可。不
过通常还是会和co...

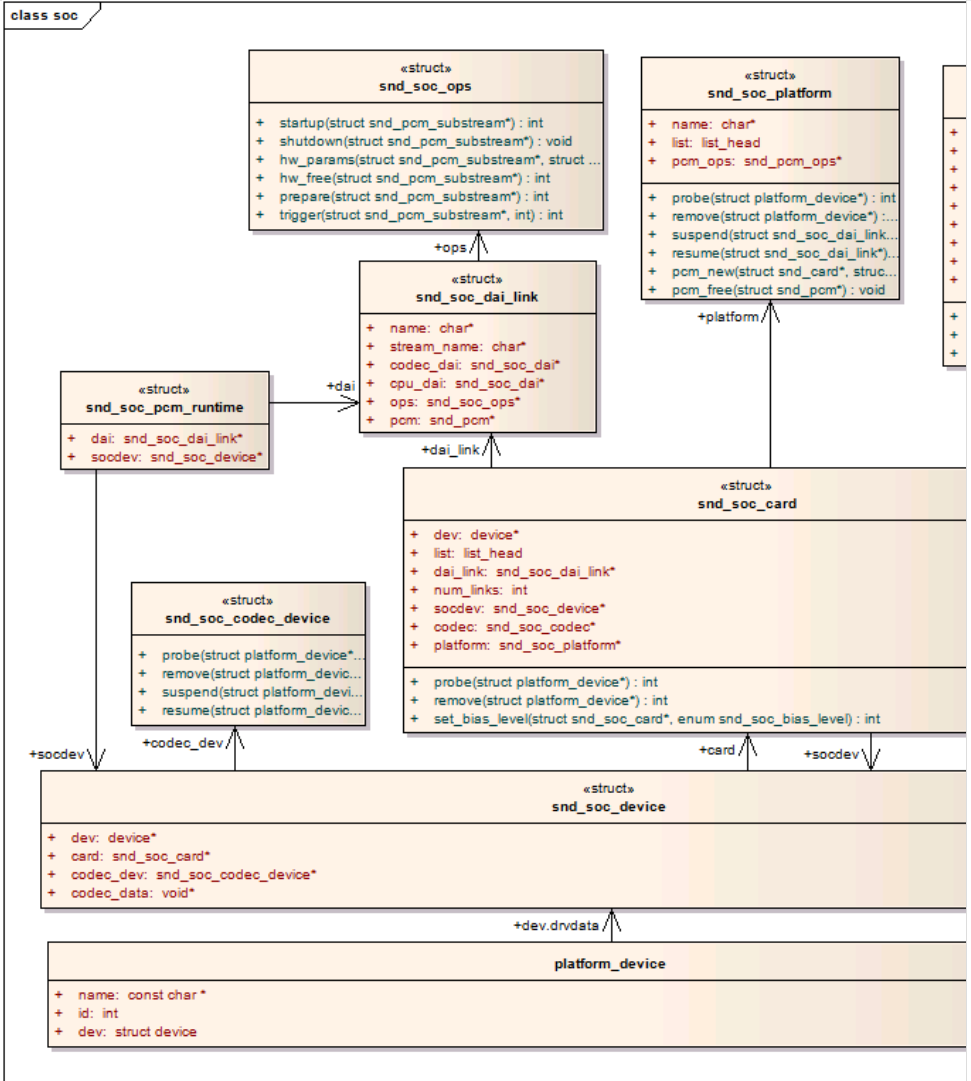


图4.1 Kernel-2.6.35-ASoC中各个结构

的静态关系

ASoC把声卡实现为一个Platform Device, 然后利用Platform_device结构中的dev字段: dev.drvdata, 它实际上指向一个snd_soc_device结构。可以认为snd_soc_device是整个ASoC数据结构的根本, 由他开始, 引出一系列的数据结构用于表述音频的各种特性和功能。snd_soc_device结构引出了snd_soc_card和soc_codec_device两个结构, 然后snd_soc_card又引出了snd_soc_platform、snd_soc_dai_link和snd_soc_codec结构。如上所述, ASoC被划分为Machine、Platform和Codec三大部分, 如果从这些数据结构看来, snd_codec_device和snd_soc_card代表着Machine驱动, snd_soc_platform则代表着Platform驱动, snd_soc_codec和soc_codec_device则代表了Codec驱动, 而snd_soc_dai_link则负责连接Platform和Codec。

5. 3.0版内核对ASoC的改进

本来写这篇文章的时候参考的内核版本是2.6.35, 不过有CSDN的朋友提出在内核版本3.0版本中, ASoC做了较大的变化。故特意下载了3.0的代码, 发现确实有所变化, 下面先贴出数据结构的静态关系图:

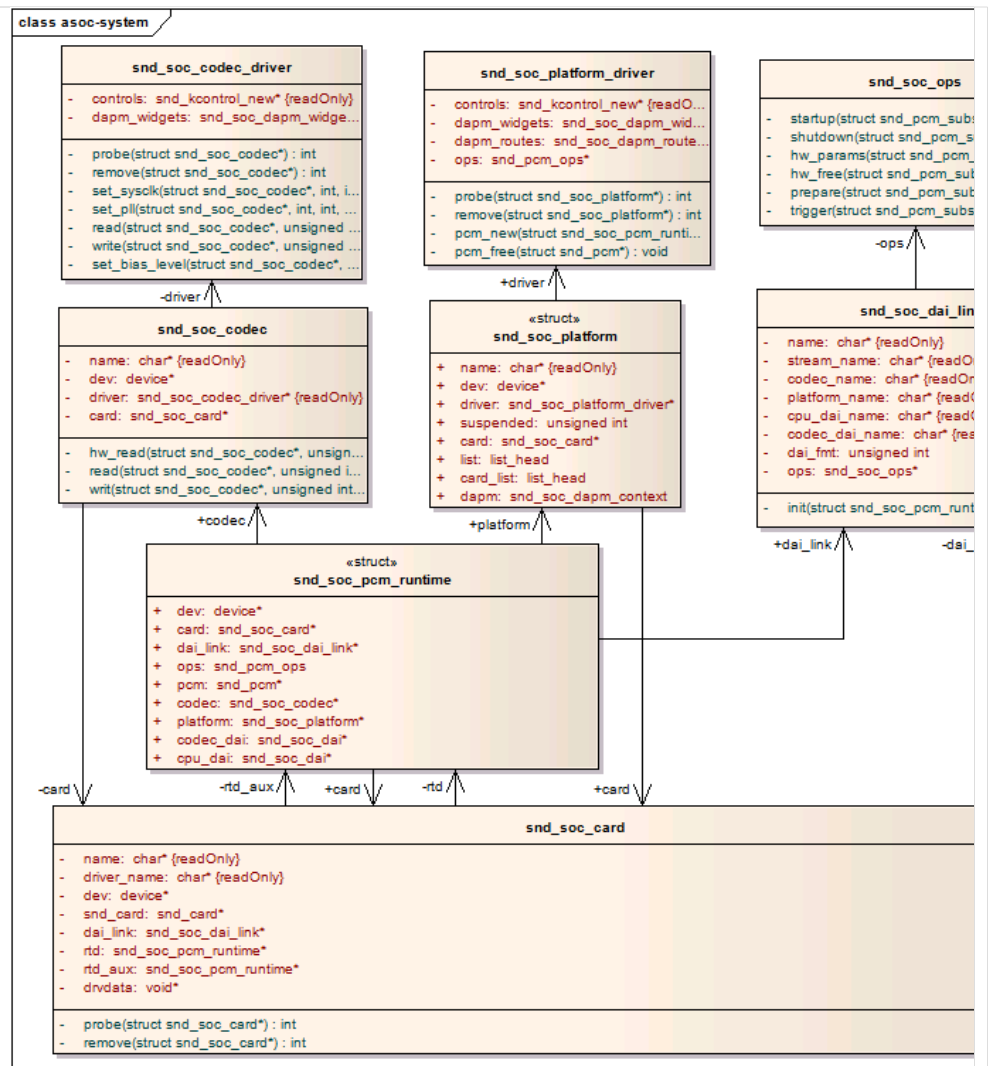


图5.1 Kernel 3.0中的ASoC数据结构

由上图我们可以看出，3.0中的数据结构更为合理和清晰，取消了snd_soc_device结构，直接用snd_soc_card取代了它，并且强化了snd_soc_pcm_runtime的作用，同时还增加了另外两个数据结构snd_soc_codec_driver和snd_soc_platform_driver，用于明确代表Codec驱动和Platform驱动。

后续的章节中将会逐一介绍Machine和Platform以及Codec驱动的工作细节和关联。

更多 1

上一篇: [翻译: Linux的电源管理架构](#)

下一篇: [Linux ALSA声卡驱动之六: ASoC架构中的Machine](#)

查看评论

4楼 slcsss 4天前 10:09发表



您好:

我是一名在读研究生，最近在ALSA架构下搞嵌入式音频程序开发，遇到了一个棘手的问题，望您赐教
开发环境是ubuntu11.04，内核是2.6.38，开发板是S3C2440+UDA1341。

程序很简单，实现的功能就是录音和播放，但是在只运行录音程序时，也就是只调用snd_pcm_readi()这个函数时，虽然能够录音（生成wav文件），但同时也能在耳机里听到所录的声音，但是我并没有调用snd_pcm_writel()这个函数。利用lsdf -g 命令查看，发现录音程序运行时只调用了pcmC0D0c这个设备，并没有pcmC0D0p这个设备，所以感觉是不是底层DMA通道配置出了问题，但是这个通道应该怎样修改我就不太清楚了。

望您给出一些建议，谢谢了！！！！

Re: DroidPhone 4天前 10:45发表



回复u013222557: 这种情况通常是你的codec中的音频路径把Mic至HP的路径被打开了, 请仔细检查Codec中的路径配置信息。

Re: slcsss 4天前 11:14发表



回复DroidPhone: 感谢您的回复, 我是新手, 想问下这个配置的具体位置在哪里?

3楼 zcf19871112 2012-09-20 10:14发表



分析得很好, 楼主, 您画UML图用的是什么软件?

2楼 wantianpei 2012-07-27 15:05发表



楼主: 我的理解是这样的, 不知道是否有问题? 请指教。

Codec: 相当于解码, 编码纯软件的代码, 都可重复利用。

Platform: i2s, pcm.....芯片级硬件控制接口相关代码, 同一种soc芯片可以重复利用。

Machine: 把platform驱动和codec驱动绑定在一起, 并实现一些附加操作, 比如说音乐播放时给功放机上电, 播放完后功放关电。这部分跟机器电路相关, 所以一般不能重复利用。

是这样的吗?

我对machine部分还是不太理解, 它还有别的作用吗? 能详细描述下吗? 多谢!

1楼 liuyan4043 2012-02-01 16:42发表



持续关注中, 期待下一篇!

您还没有登录, 请[\[登录\]](#)或[\[注册\]](#)

* 以上用户言论只代表其个人观点, 不代表CSDN网站的观点或立场

专区推荐内容

介绍一种服务器缓存结构-多级 H...
一次编写, 随处运行 英特尔HTM...
开源 - 开放式虚拟化解决方案
绳命不息, 代码不止
跟燕青一起学Android应用开...
在英特尔凌动平台上进行Andro...



更多招聘职位

我公司职位也要出现在这里

【langoon】java架构师
【北京富邦天成信息技术有限公司】java高级工程师
【南京科莱斯企业管理咨询有限公司】Flex开发工程师
(南京)
【深圳西洋镜网络技术有限公司】游戏服务端开发工程师
【博彦科技(上海)有限公司】oracle数据库开发

核心技术类目

全部主题 Java VPN Android iOS ERP IE10 Eclipse CRM JavaScript Ubuntu NFC
WAP jQuery 数据库 BI HTML5 Spring Apache Hadoop .NET API HTML SDK IIS
Fedora XML LBS Unity Splashtop UML components Windows Mobile Rails QEMU KDE
Cassandra CloudStack FTC coremail OPhone CouchBase 云计算 iOS6 Rackspace
Web App SpringSide Maemo Compuware 大数据 aptech Perl Tomado Ruby Hibernate
ThinkPHP Spark HBase Pure Solr Angular Cloud Foundry Redis Scala Django
Bootstrap

公司简介 | 招贤纳士 | 广告服务 | 银行汇款帐号 | 联系方式 | 版权声明 | 法律顾问 | 问题报告

QQ客服 微博客服 论坛反馈 联系邮箱: webmaster@csdn.net 服务热线: 400-600-2320

京 ICP 证 070598 号

北京创新乐知信息技术有限公司 版权所有

世纪乐知(北京)网络技术有限公司 提供技术支持

江苏乐知网络技术有限公司 提供商务支持

Copyright © 1999-2012, CSDN.NET, All Rights Reserved



DroidPhone的专栏

欢迎各位大虾交流，本人联系方式：droid.phx@gmail.com

[目录视图](#)
[摘要视图](#)
[RSS 订阅](#)

个人资料



DroidPhone



访问：382349次
 积分：3434分
 排名：第2116名

 原创：46篇 转载：0篇
 译文：4篇 评论：356条

文章搜索

文章分类

- [移动开发之Android](#) (11)
- [Linux内核架构](#) (15)
- [Linux设备驱动](#) (16)
- [Linux电源管理](#) (3)
- [Linux音频子系统](#) (15)
- [Linux中断子系统](#) (5)
- [Linux时间管理系统](#) (8)
- [Linux输入子系统](#) (4)

文章存档

- [2013年11月](#) (4)
- [2013年10月](#) (3)
- [2013年07月](#) (3)
- [2012年12月](#) (4)
- [2012年10月](#) (4)

展开

阅读排行

- [Android Audio System 之](#) (38982)
- [Android Audio System 之](#) (25553)
- [Android Audio System 之](#) (25317)
- [Linux ALSA声卡驱动之一](#) (24001)
- [Linux ALSA声卡驱动之二](#) (18421)
- [Android SurfaceFlinger](#)

[投票赢好礼，周周有惊喜！](#)
[2014年4月微软MVP申请开始了！](#)
[消灭0回答，赢下载分](#)
[“我的2013”年度征文活动火爆进行中！](#)
[办公大师系列经典丛书 诚聘英才](#)

Linux ALSA声卡驱动之六：ASoC架构中的Machine

分类：[Linux设备驱动](#) [Linux音频子系统](#)

2012-02-03 19:09

11924人阅读

[评论\(10\)](#)

[收藏](#)

[举报](#)

[linux](#)
[codec](#)
[playback](#)
[list](#)
[数据结构](#)
[struct](#)

- 目录(?) [-]
1. 注册Platform Device
 2. 注册Platform Driver
 3. 初始化入口soc_probe

前面一节的内容我们提到，ASoC被分为Machine、Platform和Codec三大部分，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，再次引用上一节的内容：Machine驱动负责处理机器特有的一些控件和音频事件（例如，当播放音频时，需要先行打开一个放大器）；单独的Platform和Codec驱动是不能工作的，它必须由Machine驱动把它们结合在一起才能完成整个设备的音频处理工作。

ASoC的一切都从Machine驱动开始，包括声卡的注册，绑定Platform和Codec驱动等等，下面就让我们从Machine驱动开始讨论吧。

/*****
 声明：本博客内容均由<http://blog.csdn.net/droidphone>原创，转载请注明出处，谢谢！
 *****/

1. 注册Platform Device

ASoC把声卡注册为Platform Device，我们以装配有WM8994的一款Samsung的开发板SMDK为例子做说明，WM8994是一颗Wolfson生产的多功能Codec芯片。

代码的位于：[/sound/soc/samsung/smdk_wm8994.c](#)，我们关注模块的初始化函数：

```

[cpp]
01. static int __init smdk_audio_init(void)
02. {
03.     int ret;
04.
05.     smdk_snd_device = platform_device_alloc("soc-audio", -1);
06.     if (!smdk_snd_device)
07.         return -ENOMEM;
08.
09.     platform_set_drvdata(smdk_snd_device, &smdk);
10.
11.     ret = platform_device_add(smdk_snd_device);
12.     if (ret)
13.         platform_device_put(smdk_snd_device);
14.
15.     return ret;
16. }
    
```

由此可见，模块初始化时，注册了一个名为soc-audio的Platform设备，同时把smdk设到platform_device结构的dev_drvdata字段中，这里引出了第一个数据结构snd_soc_card的实例smdk，他的定义如下：

(18248)
[Linux ALSA声卡驱动之三](#)
(17112)
[Android中的sp和wp指针](#)
(13786)
[Linux ALSA声卡驱动之七](#)
(13061)
[Android SurfaceFlinger中](#)
(12550)

评论排行

[Android Audio System 之](#) (49)
[Linux ALSA声卡驱动之八](#) (30)
[Android SurfaceFlinger中](#) (21)
[Linux ALSA声卡驱动之二](#) (18)
[Linux ALSA声卡驱动之三](#) (16)
[Android Audio System 之](#) (16)
[Linux中断（interrupt）子](#) (15)
[Android中的sp和wp指针](#) (13)
[Linux中断（interrupt）子](#) (12)
[Android SurfaceFlinger中](#) (11)

推荐文章

* [SharePoint 2010/2013 使用 Javascript来判断权限的三种方法](#)
* [坚持前进的方向：总结 2013，规划2014](#)
* [创业者那些鲜为人知的事情](#)
* [ListView具有多种item布局——实现微信对话框](#)
* [实现自己的类加载时，重写方法loadClass与findClass的区别](#)
* [GDAL影像投影转换](#)

最新评论

[Linux输入子系统：多点触控协议 gpcy123](#): 很有用，多谢分享
[ALSA声卡驱动中的DAPM详解之 wsc_168](#): 楼主,您好： 现在正在移植wm8962的驱动，遇到了一些问题，请教一下您。串口信息显示已经扫描...
[Linux ALSA声卡驱动之五：移动i slcsss](#): @DroidPhone:感谢您的回复，我是新手，想问下这个配置的具体位置在哪里？
[Linux ALSA声卡驱动之五：移动i DroidPhone](#): @u013222557:这种情况通常是你的codec中的音频路径把Mic至HP的路径被打开了，请仔细...
[Linux ALSA声卡驱动之五：移动i slcsss](#): 您好： 我是一名在读研究生，最近在ALSA架构下搞嵌入式音频程序开发，遇到了一个棘手的问题...
[ALSA声卡驱动中的DAPM详解之 DroidPhone](#): @u012389631:和电源管理和音频路径相关的control需要定义为dapm control (...
[ALSA声卡驱动中的DAPM详解之 elliepfang](#): 大侠，你好！ 这两天把您的文章1-7 看了一遍，关于control这个概念在您的文章中有提到过多次...
[ALSA声卡驱动中的DAPM详解之 elliepfang](#): @DroidPhone:因为这个是wm8962的machine上的现有代码，但如果我要porting...

[cpp]

```
01. static struct snd_soc_dai_link smdk_dai[] = {
02.     { /* Primary DAI i/f */
03.         .name = "WM8994 AIF1",
04.         .stream_name = "Pri_Dai",
05.         .cpu_dai_name = "samsung-i2s.0",
06.         .codec_dai_name = "wm8994-aif1",
07.         .platform_name = "samsung-audio",
08.         .codec_name = "wm8994-codec",
09.         .init = smdk_wm8994_init_paiftx,
10.         .ops = &smdk_ops,
11.     }, { /* Sec_Fifo Playback i/f */
12.         .name = "Sec_FIFO TX",
13.         .stream_name = "Sec_Dai",
14.         .cpu_dai_name = "samsung-i2s.4",
15.         .codec_dai_name = "wm8994-aif1",
16.         .platform_name = "samsung-audio",
17.         .codec_name = "wm8994-codec",
18.         .ops = &smdk_ops,
19.     },
20. };
21.
22. static struct snd_soc_card smdk = {
23.     .name = "SMDK-I2S",
24.     .owner = THIS_MODULE,
25.     .dai_link = smdk_dai,
26.     .num_links = ARRAY_SIZE(smdk_dai),
27. };
```

通过snd_soc_card结构，又引出了Machine驱动的另外两个数据结构：

- snd_soc_dai_link（实例：smdk_dai[]）
- snd_soc_ops（实例：smdk_ops）

其中，snd_soc_dai_link中，指定了Platform、Codec、codec_dai、cpu_dai的名字，稍后Machine驱动将会利用这些名字去匹配已经在系统中注册的platform，codec，dai，这些注册的部件都是在另外相应的Platform驱动和Codec驱动的代码文件中定义的，这样看来，Machine驱动的设备初始化代码无非就是选择合适Platform和Codec以及dai，用他们填充以上几个数据结构，然后注册Platform设备即可。当然还要实现连接Platform和Codec的dai_link对应的ops实现，本例就是smdk_ops，它只实现了hw_params函数：smdk_hw_params。

2. 注册Platform Driver

按照Linux的设备模型，有platform_device，就一定要有platform_driver。ASoC的platform_driver在以下文件中定义：sound/soc/soc-core.c。

还是先从模块的入口看起：

[cpp]

```
01. static int __init snd_soc_init(void)
02. {
03.     .....
04.     return platform_driver_register(&soc_driver);
05. }
```

soc_driver的定义如下：

[cpp]

```
01. /* ASoC platform driver */
02. static struct platform_driver soc_driver = {
03.     .driver      = {
04.         .name      = "soc-audio",
05.         .owner      = THIS_MODULE,
06.         .pm        = &soc_pm_ops,
07.     },
08.     .probe       = soc_probe,
09.     .remove      = soc_remove,
10. };
```

我们看到platform_driver的name字段为soc-audio，正好与platform_device中的名字相同，按照Linux的设备模

Linux ALSA声卡驱动之六: ASoC
elliepsang: @DroidPhone:关于
snd_soc_dai_link结构中的代码
注释中struct snd_...

ALSA声卡驱动中的DAPM详解之
DroidPhone: @u012389631:这
种名字根据实际的意义自己定义
就好了, 只要符合常识即可。不
过通常还是会和co...

型, platform总线会匹配这两个名字相同的device和driver, 同时会触发soc_probe的调用, 它正是整个ASoC驱动初始化的入口。

3. 初始化入口soc_probe()

soc_probe函数本身很简单, 它先从platform_device参数中取出snd_soc_card, 然后调用snd_soc_register_card, 通过snd_soc_register_card, 为snd_soc_pcm_runtime数组申请内存, 每一个dai_link对应snd_soc_pcm_runtime数组的一个单元, 然后把snd_soc_card中的dai_link配置复制到相应的snd_soc_pcm_runtime中, 最后, 大部分的工作都在snd_soc_instantiate_card中实现, 下面就看看snd_soc_instantiate_card做了些什么:

该函数首先利用card->instantiated来判断该卡是否已经实例化, 如果已经实例化则直接返回, 否则遍历每一对dai_link, 进行codec、platform、dai的绑定工作, 下只是代码的部分选节, 详细的代码请直接参考完整的代码树。

```
[cpp]
01.  /* bind DAIs */
02.  for (i = 0; i < card->num_links; i++)
03.      soc_bind_dai_link(card, i);
```

ASoC定义了三个全局的链表头变量: codec_list、dai_list、platform_list, 系统中所有的Codec、DAI、Platform都在注册时连接到这三个全局链表上。soc_bind_dai_link函数逐个扫描这三个链表, 根据card->dai_link[]中的名称进行匹配, 匹配后把相应的codec, dai和platform实例赋值到card->rtd[]中(snd_soc_pcm_runtime)。经过这个过程后, snd_soc_pcm_runtime: (card->rtd)中保存了本Machine中使用的Codec, DAI和Platform驱动的信息。

snd_soc_instantiate_card接着初始化Codec的寄存器缓存, 然后调用标准的alsa函数创建声卡实例:

```
[cpp]
01.  /* card bind complete so register a sound card */
02.  ret = snd_card_create(SNDRV_DEFAULT_IDX1, SNDRV_DEFAULT_STR1,
03.      card->owner, 0, &card->snd_card);
04.  card->snd_card->dev = card->dev;
05.
06.  card->dapm.bias_level = SND_SOC_BIAS_OFF;
07.  card->dapm.dev = card->dev;
08.  card->dapm.card = card;
09.  list_add(&card->dapm.list, &card->dapm_list);
```

然后, 依次调用各个子结构的probe函数:

```
[cpp]
01.  /* initialise the sound card only once */
02.  if (card->probe) {
03.      ret = card->probe(card);
04.      if (ret < 0)
05.          goto card_probe_error;
06.  }
07.
08.  /* early DAI link probe */
09.  for (order = SND_SOC_COMP_ORDER_FIRST; order <= SND_SOC_COMP_ORDER_LAST;
10.      order++) {
11.      for (i = 0; i < card->num_links; i++) {
12.          ret = soc_probe_dai_link(card, i, order);
13.          if (ret < 0) {
14.              pr_err("asoc: failed to instantiate card %s: %d\n",
15.                  card->name, ret);
16.              goto probe_dai_err;
17.          }
18.      }
19.  }
20.
21.  for (i = 0; i < card->num_aux_devs; i++) {
22.      ret = soc_probe_aux_dev(card, i);
23.      if (ret < 0) {
24.          pr_err("asoc: failed to add auxiliary devices %s: %d\n",
25.              card->name, ret);
26.          goto probe_aux_dev_err;
27.      }
```

```
28.     }
```

在上面的soc_probe_dai_link()函数中做了比较多的事情，把他展开继续讨论：

```
[cpp]
01. static int soc_probe_dai_link(struct snd_soc_card *card, int num, int order)
02. {
03.     .....
04.     /* set default power off timeout */
05.     rtd->pmdown_time = pmdown_time;
06.
07.     /* probe the cpu_dai */
08.     if (!cpu_dai->probed &&
09.         cpu_dai->driver->probe_order == order) {
10.
11.         if (cpu_dai->driver->probe) {
12.             ret = cpu_dai->driver->probe(cpu_dai);
13.         }
14.         cpu_dai->probed = 1;
15.         /* mark cpu_dai as probed and add to card dai list */
16.         list_add(&cpu_dai->card_list, &card->dai_dev_list);
17.     }
18.
19.     /* probe the CODEC */
20.     if (!codec->probed &&
21.         codec->driver->probe_order == order) {
22.         ret = soc_probe_codec(card, codec);
23.     }
24.
25.     /* probe the platform */
26.     if (!platform->probed &&
27.         platform->driver->probe_order == order) {
28.         ret = soc_probe_platform(card, platform);
29.     }
30.
31.     /* probe the CODEC DAI */
32.     if (!codec_dai->probed && codec_dai->driver->probe_order == order) {
33.         if (codec_dai->driver->probe) {
34.             ret = codec_dai->driver->probe(codec_dai);
35.         }
36.
37.         /* mark codec_dai as probed and add to card dai list */
38.         codec_dai->probed = 1;
39.         list_add(&codec_dai->card_list, &card->dai_dev_list);
40.     }
41.
42.     /* complete DAI probe during last probe */
43.     if (order != SND_SOC_COMP_ORDER_LAST)
44.         return 0;
45.
46.     ret = soc_post_component_init(card, codec, num, 0);
47.     if (ret)
48.         return ret;
49.     .....
50.     /* create the pcm */
51.     ret = soc_new_pcm(rtd, num);
52.     .....
53.     return 0;
54. }
```

该函数出了挨个调用了codec，dai和platform驱动的probe函数外，在最后还调用了soc_new_pcm()函数用于创建标准alsa驱动的pcm逻辑设备。现在把该函数的部分代码也贴出来：

```
[cpp]
01. /* create a new pcm */
02. int soc_new_pcm(struct snd_soc_pcm_runtime *rtd, int num)
03. {
04.     .....
05.     struct snd_pcm_ops *soc_pcm_ops = &rtd->ops;
06.
07.     soc_pcm_ops->open = soc_pcm_open;
```

```

08.     soc_pcm_ops->close    = soc_pcm_close;
09.     soc_pcm_ops->hw_params = soc_pcm_hw_params;
10.     soc_pcm_ops->hw_free  = soc_pcm_hw_free;
11.     soc_pcm_ops->prepare  = soc_pcm_prepare;
12.     soc_pcm_ops->trigger  = soc_pcm_trigger;
13.     soc_pcm_ops->pointer  = soc_pcm_pointer;
14.
15.     ret = snd_pcm_new(rtd->card->snd_card, new_name,
16.                      num, playback, capture, &pcm);
17.
18.     /* DAPM dai link stream work */
19.     INIT_DELAYED_WORK(&rtd->delayed_work, close_delayed_work);
20.
21.     rtd->pcm = pcm;
22.     pcm->private_data = rtd;
23.     if (platform->driver->ops) {
24.         soc_pcm_ops->mmap = platform->driver->ops->mmap;
25.         soc_pcm_ops->pointer = platform->driver->ops->pointer;
26.         soc_pcm_ops->iocctl = platform->driver->ops->iocctl;
27.         soc_pcm_ops->copy = platform->driver->ops->copy;
28.         soc_pcm_ops->silence = platform->driver->ops->silence;
29.         soc_pcm_ops->ack = platform->driver->ops->ack;
30.         soc_pcm_ops->page = platform->driver->ops->page;
31.     }
32.
33.     if (playback)
34.         snd_pcm_set_ops(pcm, SNDRV_PCM_STREAM_PLAYBACK, soc_pcm_ops);
35.
36.     if (capture)
37.         snd_pcm_set_ops(pcm, SNDRV_PCM_STREAM_CAPTURE, soc_pcm_ops);
38.
39.     if (platform->driver->pcm_new) {
40.         ret = platform->driver->pcm_new(rtd);
41.         if (ret < 0) {
42.             pr_err("asoc: platform pcm constructor failed\n");
43.             return ret;
44.         }
45.     }
46.
47.     pcm->private_free = platform->driver->pcm_free;
48.     return ret;
49. }

```

该函数首先初始化snd_soc_runtime中的snd_pcm_ops字段，也就是rtd->ops中的部分成员，例如open, close, hw_params等，紧接着调用标准alsa驱动中的创建pcm的函数snd_pcm_new()创建声卡的pcm实例，pcm的private_data字段设置为该runtime变量rtd，然后用platform驱动中的snd_pcm_ops替换部分pcm中的snd_pcm_ops字段，最后，调用platform驱动的pcm_new回调，该回调实现该platform下的dma内存申请和dma初始化等相关工作。到这里，声卡和他的pcm实例创建完成。

回到snd_soc_instantiate_card函数，完成snd_card和snd_pcm的创建后，接着对dapm和dai支持的格式做出一些初始化设置工作后，调用了 card->late_probe(card)进行一些最后的初始化设置工作，最后则是调用标准alsa驱动的声卡注册函数对声卡进行注册：

```

[cpp]
01.  if (card->late_probe) {
02.      ret = card->late_probe(card);
03.      if (ret < 0) {
04.          dev_err(card->dev, "%s late_probe() failed: %d\n",
05.                  card->name, ret);
06.          goto probe_aux_dev_err;
07.      }
08.  }
09.
10.  snd_soc_dapm_new_widgets(&card->dapm);
11.
12.  if (card->fully_routed)
13.      list_for_each_entry(codec, &card->codec_dev_list, card_list)
14.          snd_soc_dapm_auto_nc_codec_pins(codec);
15.
16.  ret = snd_card_register(card->snd_card);
17.  if (ret < 0) {
18.      printk(KERN_ERR "asoc: failed to register soundcard for %s\n", card->name);
19.      goto probe_aux_dev_err;
20.  }

```

至此，整个Machine驱动的初始化已经完成，通过各个子结构的probe调用，实际上，也完成了部分Platfrom驱动和Codec驱动的初始化工作，整个过程可以用一下的序列图表示：

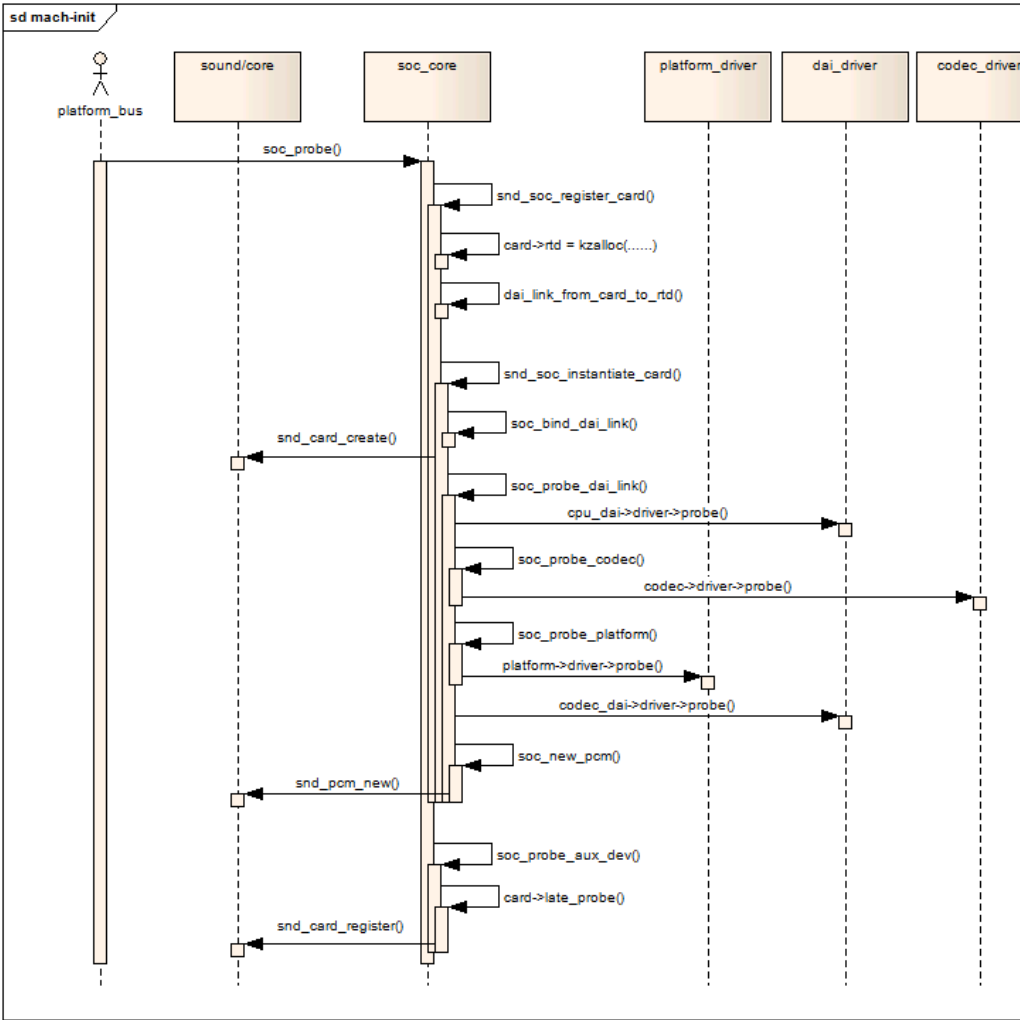


图3.1 基于3.0内核 soc_probe序列图

下面的序列图是本文第一个版本，基于内核2.6.35，大家也可以参考一下两个版本的差异：

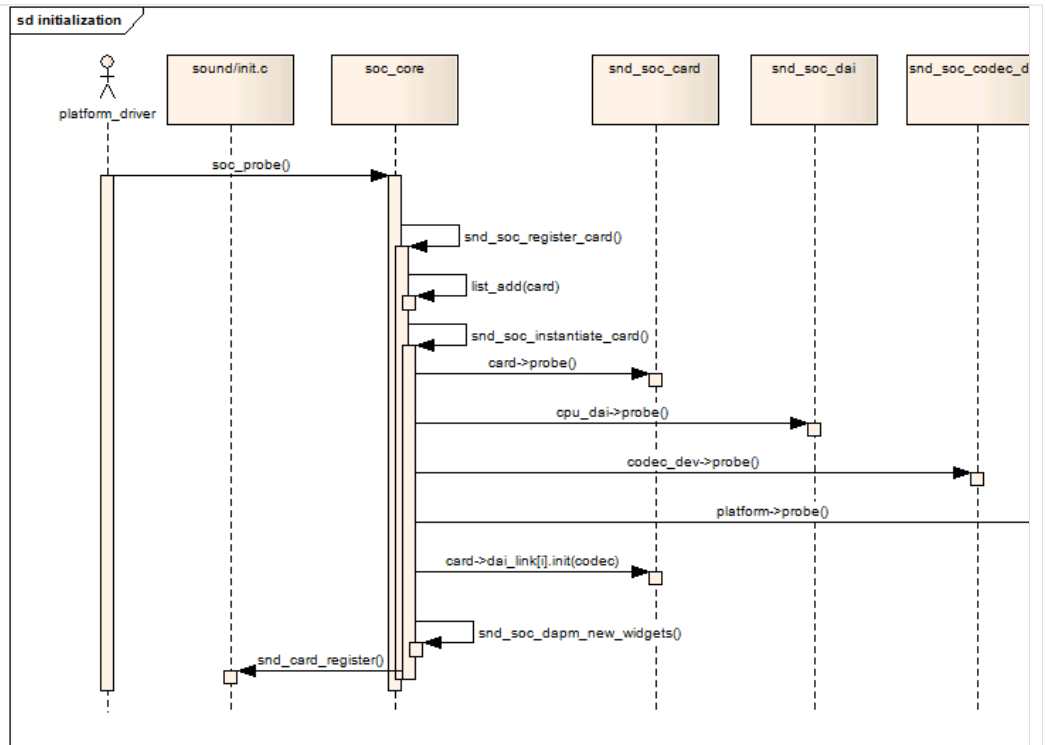


图3.2 基于2.6.35 soc_probe序列图

上一篇: [Linux ALSA声卡驱动之五: 移动设备中的ALSA \(ASoC\)](#)

下一篇: [Linux ALSA声卡驱动之七: ASoC架构中的Codec](#)

更多 1

电源管理		linux
linux		嵌入式li
游戏		游戏

查看评论

7楼 [jdwwhy](#) 2013-11-11 16:33发表



楼主, 首先膜拜下, 请问楼主, 如果只想把音频芯片如wm8904移植到ARM板上 (s5pv210), 这个Machine文件怎么写。能不能具体点讲下。

Re: [DroidPhone](#) 2013-11-11 18:23发表



回复jdwwhy: 参考本章第一节的内容即可, 请参考sound/soc/samsung/smdk_wm8994.c。主要是要在snd_soc_dai_link结构中指定正确codec、platform和dai的名字。

Re: [elliepsang](#) 2013-12-16 22:21发表



回复DroidPhone: 关于snd_soc_dai_link结构中的代码注释中

```

struct snd_soc_dai_link {
    /* config - must be set by machine driver */
    const char *name; /* Codec name */
    const char *stream_name; /* Stream name */
    const char *codec_name; /* for multi-codec */
    const char *platform_name; /* for multi-platform */
    const char *cpu_dai_name;
    const char *codec_dai_name;
    .....
};
  
```

其中的stream_name; codec_name; codec_dai_name可以在wm8994.c中找到
platform_name; cpu_dai_name; 也可以在soc的samsung目录中找到相应的, 可是.name =
"WM8994 AIF1", 却找不到

我在关于wm8962上也是没有看见相关的
static struct snd_soc_dai_link imx_dai[] = {
{
.name = "HiFi",
.stream_name = "HiFi",
.codec_dai_name = "wm8962",
.codec_name = "wm8962.0-001a",


```
.cpu_dai_name = "imx-ssi.1",
.platform_name = "imx-pcm-audio.1",
.init = imx_wm8962_init,
.ops = &imx_hifi_ops,
},
};
而且这里的name = "HiFi",
.stream_name = "HiFi",
.codec_name = "wm8962.0-001a",
也不知道是如何定义的
```

6楼 [girlkoo](#) 2013-07-12 15:36发表



对于probe函数的解释，我有个疑问，在分配snd_soc_pcm_runtime数组时分配的长度是card->num_links+card->num_aux_devs，而不是楼主所说的只有dai_links的长度，内核源码如下：

```
card->rtd = kzalloc(sizeof(struct snd_soc_pcm_runtime) *
(card->num_links + card->num_aux_devs),
GFP_KERNEL);
if (card->rtd == NULL)
return -ENOMEM;
card->rtd_aux = &card->rtd[card->num_links];
```

```
for (i = 0; i < card->num_links; i++)
card->rtd[i].dai_link = &card->dai_link[i];
另外，for循环并没有拷贝dai_link，而是将rtd的dai_link指针指向了实际的dai_link对象。
```

5楼 [vito_coleone](#) 2013-05-28 13:34发表



给力，十分感谢楼主！

4楼 [lamaboy2012](#) 2012-12-14 17:30发表



我现在是usb 接口的音频设备驱动，你建议下我是用usb-alsa 还是 usb-soc的啊，我现在没什么思路

3楼 [天才2012](#) 2012-06-25 15:25发表



奇怪的是为何3.0的反而复杂一些

2楼 [天才2012](#) 2012-06-25 15:25发表



博主在这里看到了，版本的區別你这里提到了，没看仔细不好意思

1楼 [markzliu](#) 2012-03-13 16:04发表



我现在使用3.0的kernel，soc-audio在/sys/devices/platform/soc-audio下创建成功，soc-audio driver 也注册成功，但是soc_probe没有调用到（加了打印但是没有输出），可能是什么原因呢？非常感谢

Re: [DroidPhone](#) 2012-03-13 18:51发表



回复markzliu: soc_probe都没执行，你怎么确定driver注册成功？

您还没有登录,请[\[登录\]](#)或[\[注册\]](#)

* 以上用户言论只代表其个人观点，不代表CSDN网站的观点或立场

专区推荐内容

提升Android* 模拟器的速...
Javascript: this用...
利用IA SIMD技术来加速游戏...
英特尔软件开发工具
超极本触摸屏一种全新的动手方式
[Intel CPU中的安全指令...]



更多招聘职位

我公司职位也要出现在这里

【广州友魄信息科技有限公司】Asp.net（C#）开发工程师
【北京奥鹏远程教育中心有限公司】专职/兼职IT培训讲师（Web前端开发，PHP，3G开发方向）
【langoon】java架构师
【北京富邦天成信息技术有限公司】java高级工程师
【南京科莱斯企业管理咨询有限公司】Flex开发工程师

核心技术类目

全部主题 Java VPN Android iOS ERP IE10 Eclipse CRM JavaScript Ubuntu NFC
WAP jQuery 数据库 BI HTML5 Spring Apache Hadoop .NET API HTML SDK IIS
Fedora XML LBS Unity Splashtop UML components Windows Mobile Rails QEMU KDE
Cassandra CloudStack FTC coremail OPhone CouchBase 云计算 iOS6 Rackspace
Web App SpringSide Maemo Compuware 大数据 aptech Perl Tomado Ruby Hibernate
ThinkPHP Spark HBase Pure Solr Angular Cloud Foundry Redis Scala Django

公司简介 | 招贤纳士 | 广告服务 | 银行汇款帐号 | 联系方式 | 版权声明 | 法律顾问 | 问题报告
QQ客服 微博客服 论坛反馈 联系邮箱: webmaster@csdn.net 服务热线: 400-600-2320
京 ICP 证 070598 号

北京创新乐知信息技术有限公司 版权所有

世纪乐知(北京)网络技术有限公司 提供技术支持

江苏乐知网络技术有限公司 提供商务支持

Copyright © 1999-2012, CSDN.NET, All Rights Reserved 

DroidPhone的专栏

欢迎各位大虾交流，本人联系方式：droid.phx@gmail.com

[目录视图](#)
[摘要视图](#)
[RSS 订阅](#)

个人资料



DroidPhone



访问：382342次
 积分：3434分
 排名：第2116名
 原创：46篇 转载：0篇
 译文：4篇 评论：356条

文章搜索

文章分类

- [移动开发之Android](#) (11)
- [Linux内核架构](#) (15)
- [Linux设备驱动](#) (16)
- [Linux电源管理](#) (3)
- [Linux音频子系统](#) (15)
- [Linux中断子系统](#) (5)
- [Linux时间管理系统](#) (8)
- [Linux输入子系统](#) (4)

文章存档

- [2013年11月](#) (4)
- [2013年10月](#) (3)
- [2013年07月](#) (3)
- [2012年12月](#) (4)
- [2012年10月](#) (4)

展开

阅读排行

- [Android Audio System 之](#) (39882)
- [Android Audio System 之](#) (25553)
- [Android Audio System 之](#) (25317)
- [Linux ALSA声卡驱动之一](#) (24001)
- [Linux ALSA声卡驱动之二](#) (18421)
- [Android SurfaceFlinger](#)

[投票赢好礼，周周有惊喜！](#)
[2014年4月微软MVP申请开始了！](#)
[消灭0回答，赢下载分](#)
[“我的2013”年度征文活动火爆进行中！](#)
[办公大师系列经典丛书 诚聘英才](#)

Linux ALSA声卡驱动之七：ASoC架构中的Codec

分类：[Linux设备驱动](#) [Linux音频子系统](#)
 2012-02-23 14:12
 13073人阅读
 评论(7)
 收藏
 举报

[codec](#)
[linux](#)
[struct](#)
[playback](#)
[audio](#)
[stream](#)

目录(?)

[-]

1. Codec简介
2. ASoC中对Codec的数据抽象
3. Codec的注册
4. mfd设备
5. Codec初始化
6. regmap-io
- 7.

1. Codec简介

在移动设备中，Codec的作用可以归结为4种，分别是：

- 对PCM等信号进行D/A转换，把数字的音频信号转换为模拟信号
- 对Mic、Linein或者其他输入源的模拟信号进行A/D转换，把模拟的声音信号转变CPU能够处理的数字信号
- 对音频通路进行控制，比如播放音乐，收听调频收音机，又或者接听电话时，音频信号在codec内的流通路线是不一样的
- 对音频信号做出相应的处理，例如音量控制，功率放大，EQ控制等等

ASoC对Codec的这些功能都定义好了一些列相应的接口，以方便地对Codec进行控制。ASoC对Codec驱动的一个基本要求是：驱动程序的代码必须要做到平台无关性，以方便同一个Codec的代码不经修改即可用在不同的平台上。以下的讨论基于wolfson的Codec芯片WM8994，kernel的版本3.3.x。

/*****
 声明：本博客内容均由<http://blog.csdn.net/droidphone>原创，转载请注明出处，谢谢！
 *****/

2. ASoC中对Codec的数据抽象

描述Codec的最主要的几个数据结构分别

是：snd_soc_codec, snd_soc_codec_driver, snd_soc_dai, snd_soc_dai_driver，其中的snd_soc_dai和snd_soc_dai_driver在ASoC的Platform驱动中也会使用到，Platform和Codec的DAI通过snd_soc_dai_link结构，在Machine驱动中进行绑定连接。下面我们先看看这几个结构的定义，这里我只贴出我要关注的字段，详细的定义请参照：/include/sound/soc.h。

snd_soc_codec:

```

[html]
01.  /* SoC Audio Codec device */
02.  struct snd_soc_codec {
03.      const char *name; /* Codec的名字*/
04.      struct device *dev; /* 指向Codec设备的指针 */
05.      const struct snd_soc_codec_driver *driver; /* 指向该codec的驱动的指针 */
06.      struct snd_soc_card *card; /* 指向Machine驱动的card实例 */
07.      int num_dai; /* 该Codec数字接口的个数，目前越来越多的Codec带有多多个I2S或者是PCM接口 */
08.      int (*volatile_register)(...); /* 用于判定某一寄存器是否是volatile */
09.      int (*readable_register)(...); /* 用于判定某一寄存器是否可读 */
    
```

(18248)
[Linux ALSA声卡驱动之三](#)
(17112)
[Android中的sp和wp指针](#)
(13786)
[Linux ALSA声卡驱动之七](#)
(13061)
[Android SurfaceFlinger](#)
(12550)

评论排行

[Android Audio System 之](#) (49)
[Linux ALSA声卡驱动之八](#) (30)
[Android SurfaceFlinger](#) (21)
[Linux ALSA声卡驱动之二](#) (18)
[Linux ALSA声卡驱动之三](#) (16)
[Android Audio System 之](#) (16)
[Linux中断（interrupt）子](#) (15)
[Android中的sp和wp指针](#) (13)
[Linux中断（interrupt）子](#) (12)
[Android SurfaceFlinger](#) (11)

推荐文章

* [SharePoint 2010/2013 使用 Javascript来判断权限的三种方法](#)
* [坚持前进的方向：总结 2013，规划2014](#)
* [创业者那些鲜为人知的事情](#)
* [ListView具有多种item布局——实现微信对话框](#)
* [实现自己的类加载时，重写方法loadClass与findClass的区别](#)
* [GDAL影像投影转换](#)

最新评论

[Linux输入子系统：多点触控协议 gocy123](#): 很有用，多谢分享

[ALSA声卡驱动中的DAPM详解之 wsc_168](#): 楼主,您好: 现在正在移植wm8962的驱动,遇到了一些问题,请教一下您.串口信息显示已经扫描...

[Linux ALSA声卡驱动之五：移动i slcsss](#): @DroidPhone:感谢您的回复,我是新手,想问下这个配置的具体位置在哪里?

[Linux ALSA声卡驱动之五：移动i DroidPhone](#): @u013222557:这种情况通常是你的codec中的音频路径把Mic至HP的路径被打开了,请仔细...

[Linux ALSA声卡驱动之五：移动i slcsss](#): 您好: 我是一名在读研究生,最近在ALSA架构下搞嵌入式音频程序开发,遇到了一个棘手的问题...

[ALSA声卡驱动中的DAPM详解之 DroidPhone](#): @u012389631:和电源管理和音频路径相关的control需要定义为dapm control (...)

[ALSA声卡驱动中的DAPM详解之 elliepfang](#): 大侠,你好! 这两天把您的文章1-7 看了一遍,关于control这个概念在您的文章中有提到过多次...

[ALSA声卡驱动中的DAPM详解之 elliepfang](#): @DroidPhone:因为这个是wm8962的machine上的现有代码,但如果我要porting...

```
10.     int (*writable_register)(...); /* 用于判定某一寄存器是否可写 */
11.
12.     /* runtime */
13.     .....
14.     /* codec IO */
15.     void *control_data; /* 该指针指向的结构用于对codec的控制,通常和read, write字段联合使用 */
16.     enum snd_soc_control_type control_type; /* 可以是SND_SOC_SPI, SND_SOC_I2C, SND_SOC_REGMAP中
   的一种 */
17.     unsigned int (*read)(struct snd_soc_codec *, unsigned int); /* 读取Codec寄存器的函数 */
18.     int (*write)(struct snd_soc_codec *, unsigned int, unsigned int); /* 写入Codec寄存器的函
   数 */
19.     /* dapm */
20.     struct snd_soc_dapm_context dapm; /* 用于DAPM控件 */
21.     };
```

snd_soc_codec_driver:

```
[html]
01. /* codec driver */
02. struct snd_soc_codec_driver {
03.     /* driver ops */
04.     int (*probe)(struct snd_soc_codec *); /* codec驱动的probe函数,由snd_soc_instantiate_card
   回调 */
05.     int (*remove)(struct snd_soc_codec *);
06.     int (*suspend)(struct snd_soc_codec *); /* 电源管理 */
07.     int (*resume)(struct snd_soc_codec *); /* 电源管理 */
08.
09.     /* Default control and setup, added after probe() is run */
10.     const struct snd_kcontrol_new *controls; /* 音频控件指针 */
11.     const struct snd_soc_dapm_widget *dapm_widgets; /* dapm部件指针 */
12.     const struct snd_soc_dapm_route *dapm_routes; /* dapm路由指针 */
13.
14.     /* codec wide operations */
15.     int (*set_sysclk)(...); /* 时钟配置函数 */
16.     int (*set_pll)(...); /* 锁相环配置函数 */
17.
18.     /* codec IO */
19.     unsigned int (*read)(...); /* 读取codec寄存器函数 */
20.     int (*write)(...); /* 写入codec寄存器函数 */
21.     int (*volatile_register)(...); /* 用于判定某一寄存器是否是volatile */
22.     int (*readable_register)(...); /* 用于判定某一寄存器是否可读 */
23.     int (*writable_register)(...); /* 用于判定某一寄存器是否可写 */
24.
25.     /* codec bias level */
26.     int (*set_bias_level)(...); /* 偏置电压配置函数 */
27.
28.     };
```

snd_soc_dai:

```
[html]
01. /*
02.  * Digital Audio Interface runtime data.
03.  *
04.  * Holds runtime data for a DAI.
05.  */
06. struct snd_soc_dai {
07.     const char *name; /* dai的名字 */
08.     struct device *dev; /* 设备指针 */
09.
10.     /* driver ops */
11.     struct snd_soc_dai_driver *driver; /* 指向dai驱动结构的指针 */
12.
13.     /* DAI runtime info */
14.     unsigned int capture_active:1; /* stream is in use */
15.     unsigned int playback_active:1; /* stream is in use */
16.
17.     /* DAI DMA data */
18.     void *playback_dma_data; /* 用于管理playback dma */
19.     void *capture_dma_data; /* 用于管理capture dma */
20.
21.     /* parent platform/codec */
22.     union {
23.         struct snd_soc_platform *platform; /* 如果是cpu dai,指向所绑定的平台 */
24.         struct snd_soc_codec *codec; /* 如果是codec dai指向所绑定的codec */
25.     };
```

Linux ALSA声卡驱动之六: ASoC
elliepfang: @DroidPhone:关于
snd_soc_dai_link结构中的代码
注释中struct snd_...

ALSA声卡驱动中的DAPM详解之
DroidPhone: @u012389631:这
种名字根据实际的意义自己定义
就好了, 只要符合常识即可。不
过通常还是会和co...

```
26.     struct snd_soc_card *card; /* 指向Machine驱动中的crad实例 */
27. };
```

snd_soc_dai_driver:

```
[html]
01. /*
02.  * Digital Audio Interface Driver.
03.  *
04.  * Describes the Digital Audio Interface in terms of its ALSA, DAI and AC97
05.  * operations and capabilities. Codec and platform drivers will register this
06.  * structure for every DAI they have.
07.  *
08.  * This structure covers the clocking, formatting and ALSA operations for each
09.  * interface.
10.  */
11. struct snd_soc_dai_driver {
12.     /* DAI description */
13.     const char *name; /* dai驱动名字 */
14.
15.     /* DAI driver callbacks */
16.     int (*probe)(struct snd_soc_dai *dai); /* dai驱动的probe函数, 由snd_soc_instantiate_card回
    调 */
17.     int (*remove)(struct snd_soc_dai *dai);
18.     int (*suspend)(struct snd_soc_dai *dai); /* 电源管理 */
19.     int (*resume)(struct snd_soc_dai *dai);
20.
21.     /* ops */
22.     const struct snd_soc_dai_ops *ops; /* 指向本dai的snd_soc_dai_ops结构 */
23.
24.     /* DAI capabilities */
25.     struct snd_soc_pcm_stream capture; /* 描述capture的能力 */
26.     struct snd_soc_pcm_stream playback; /* 描述playback的能力 */
27. };
```

snd_soc_dai_ops用于实现该dai的控制盒参数配置:

```
[html]
01. struct snd_soc_dai_ops {
02.     /*
03.      * DAI clocking configuration, all optional.
04.      * Called by soc_card drivers, normally in their hw_params.
05.      */
06.     int (*set_sysclk)(...);
07.     int (*set_pll)(...);
08.     int (*set_clkdiv)(...);
09.     /*
10.      * DAI format configuration
11.      * Called by soc_card drivers, normally in their hw_params.
12.      */
13.     int (*set_fmt)(...);
14.     int (*set_tdm_slot)(...);
15.     int (*set_channel_map)(...);
16.     int (*set_tristate)(...);
17.     /*
18.      * DAI digital mute - optional.
19.      * Called by soc-core to minimise any pops.
20.      */
21.     int (*digital_mute)(...);
22.     /*
23.      * ALSA PCM audio operations - all optional.
24.      * Called by soc-core during audio PCM operations.
25.      */
26.     int (*startup)(...);
27.     void (*shutdown)(...);
28.     int (*hw_params)(...);
29.     int (*hw_free)(...);
30.     int (*prepare)(...);
31.     int (*trigger)(...);
32.     /*
33.      * For hardware based FIFO caused delay reporting.
34.      * Optional.
35.      */
36.     snd_pcm_sframes_t (*delay)(...);
37. };
```

3. Codec的注册

因为Codec驱动的代码要做到平台无关性，要使得Machine驱动能够使用该Codec，Codec驱动的首要任务就是确定snd_soc_codec和snd_soc_dai的实例，并把它们注册到系统中，注册后的codec和dai才能为Machine驱动所用。以WM8994为例，对应的代码位置：/sound/soc/codecs/wm8994.c，模块的入口函数注册了一个platform driver：

```
[html]
01. static struct platform_driver wm8994_codec_driver = {
02.     .driver = {
03.         .name = "wm8994-codec",
04.         .owner = THIS_MODULE,
05.     },
06.     .probe = wm8994_probe,
07.     .remove = __devexit_p(wm8994_remove),
08. };
09.
10. module_platform_driver(wm8994_codec_driver);
```

有platform driver，必定会有相应的platform device，这个platform device的来源后面再说，显然，platform driver注册后，probe回调将会被调用，这里是wm8994_probe函数：

```
[html]
01. static int __devinit wm8994_probe(struct platform_device *pdev)
02. {
03.     return snd_soc_register_codec(&pdev->dev, &soc_codec_dev_wm8994,
04.         wm8994_dai, ARRAY_SIZE(wm8994_dai));
05. }
```

其中，soc_codec_dev_wm8994和wm8994_dai的定义如下（代码中定义了3个dai，这里只列出第一个）：

```
[html]
01. static struct snd_soc_codec_driver soc_codec_dev_wm8994 = {
02.     .probe = wm8994_codec_probe,
03.     .remove = wm8994_codec_remove,
04.     .suspend = wm8994_suspend,
05.     .resume = wm8994_resume,
06.     .set_bias_level = wm8994_set_bias_level,
07.     .reg_cache_size = WM8994_MAX_REGISTER,
08.     .volatile_register = wm8994_soc_volatile,
09. };
```

```
[html]
01. static struct snd_soc_dai_driver wm8994_dai[] = {
02.     {
03.         .name = "wm8994-aif1",
04.         .id = 1,
05.         .playback = {
06.             .stream_name = "AIF1 Playback",
07.             .channels_min = 1,
08.             .channels_max = 2,
09.             .rates = WM8994_RATES,
10.             .formats = WM8994_FORMATS,
11.         },
12.         .capture = {
13.             .stream_name = "AIF1 Capture",
14.             .channels_min = 1,
15.             .channels_max = 2,
16.             .rates = WM8994_RATES,
17.             .formats = WM8994_FORMATS,
18.         },
19.         .ops = &wm8994_aif1_dai_ops,
20.     },
21.     .....
22. }
```

可见，Codec驱动的第一个步骤就是定义snd_soc_codec_driver和snd_soc_dai_driver的实例，然后调用snd_soc_register_codec函数对Codec进行注册。进入snd_soc_register_codec函数看看：
首先，它申请了一个snd_soc_codec结构的实例：

```
[html]
01. codec = kzalloc(sizeof(struct snd_soc_codec), GFP_KERNEL);
```

确定codec的名字，这个名字很重要，Machine驱动定义的snd_soc_dai_link中会指定每个link的codec和dai的名字，进行匹配绑定时就是通过和这里的名字比较，从而找到该Codec的！

```
[html]
01. /* create CODEC component name */
02. codec->name = fmt_single_name(dev, &codec->id);
```

然后初始化它的各个字段，多数字段的值来自上面定义的snd_soc_codec_driver的实例soc_codec_dev_wm8994:

```
[html]
01. codec->write = codec_drv->write;
02. codec->read = codec_drv->read;
03. codec->volatile_register = codec_drv->volatile_register;
04. codec->readable_register = codec_drv->readable_register;
05. codec->writable_register = codec_drv->writable_register;
06. codec->dapm.bias_level = SND_SOC_BIAS_OFF;
07. codec->dapm.dev = dev;
08. codec->dapm.codec = codec;
09. codec->dapm.seq_notifier = codec_drv->seq_notifier;
10. codec->dapm.stream_event = codec_drv->stream_event;
11. codec->dev = dev;
12. codec->driver = codec_drv;
13. codec->num_dai = num_dai;
```

在做了一些寄存器缓存的初始化和配置工作后，通过snd_soc_register_dais函数对本Codec的dai进行注册:

```
[html]
01. /* register any DAIs */
02. if (num_dai) {
03.     ret = snd_soc_register_dais(dev, dai_drv, num_dai);
04.     if (ret < 0)
05.         goto fail;
06. }
```

最后，它把codec实例链接到全局链表codec_list中，并且调用snd_soc_instantiate_cards是函数触发Machine驱动进行一次匹配绑定操作:

```
[html]
01. list_add(&codec->list, &codec_list);
02. snd_soc_instantiate_cards();
```

上面的snd_soc_register_dais函数其实也是和snd_soc_register_codec类似，显示为每个snd_soc_dai实例分配内存，确定dai的名字，用snd_soc_dai_driver实例的字段对它进行必要初始化，最后把该dai链接到全局链表dai_list中，和Codec一样，最后也会调用snd_soc_instantiate_cards函数触发一次匹配绑定的操作。

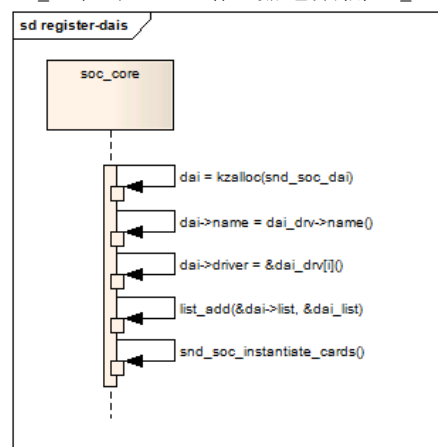


图3.1 dai的注册

关于snd_soc_instantiate_cards函数，请参阅另一篇博文：[Linux音频驱动之六：ASoC架构中的Machine](#)。

4. mfd设备

前面已经提到，codec驱动把自己注册为一个platform driver，那对应的platform device在哪里定义？答案是在以下代码文件中：/drivers/mfd/wm8994-core.c。

WM8994本身具备多种功能，除了codec外，它还有作为LDO和GPIO使用，这几种功能共享一些IO和中断资源，linux为这种设备提供了一套标准的实现方法：mfd设备。其基本思想是为这些功能的公共部分实现一个父设备，以便共享某些系统资源和功能，然后每个子功能实现为它的子设备，这样既共享了资源和代码，又能实现合理的设备层次结构，主要利用到的API就

是：mfd_add_devices(), mfd_remove_devices(), mfd_cell_enable(), mfd_cell_disable(), mfd_clone_cell()。

回到wm8994-core.c中，因为WM8994使用I2C进行内部寄存器的存取，它首先注册了一个I2C驱动：

```
[html]
01. static struct i2c_driver wm8994_i2c_driver = {
02.     .driver = {
03.         .name = "wm8994",
04.         .owner = THIS_MODULE,
05.         .pm = &wm8994_pm_ops,
06.         .of_match_table = wm8994_of_match,
07.     },
08.     .probe = wm8994_i2c_probe,
09.     .remove = wm8994_i2c_remove,
10.     .id_table = wm8994_i2c_id,
11. };
12.
13. static int __init wm8994_i2c_init(void)
14. {
15.     int ret;
16.
17.     ret = i2c_add_driver(&wm8994_i2c_driver);
18.     if (ret != 0)
19.         pr_err("Failed to register wm8994 I2C driver: %d\n", ret);
20.
21.     return ret;
22. }
23. module_init(wm8994_i2c_init);
```

进入wm8994_i2c_probe()函数，它先申请了一个wm8994结构的变量，该变量被作为这个I2C设备的driver_data使用，上面已经讲过，codec作为它的子设备，将会取出并使用这个driver_data。接下来，本函数利用regmap_init_i2c()初始化并获得一个regmap结构，该结构主要用于后续基于regmap机制的寄存器I/O，关于regmap我们留在后面再讲。最后，通过wm8994_device_init()来添加mfd子设备：

```
[html]
01. static int wm8994_i2c_probe(struct i2c_client *i2c,
02.                             const struct i2c_device_id *id)
03. {
04.     struct wm8994 *wm8994;
05.     int ret;
06.     wm8994 = devm_kzalloc(&i2c->dev, sizeof(struct wm8994), GFP_KERNEL);
07.     i2c_set_clientdata(i2c, wm8994);
08.     wm8994->dev = &i2c->dev;
09.     wm8994->irq = i2c->irq;
10.     wm8994->type = id->driver_data;
11.     wm8994->regmap = regmap_init_i2c(i2c, &wm8994_base_regmap_config);
12.
13.     return wm8994_device_init(wm8994, i2c->irq);
14. }
```

继续进入wm8994_device_init()函数，它首先为两个LDO添加mfd子设备：

```
[html]
01. /* Add the on-chip regulators first for bootstrapping */
02. ret = mfd_add_devices(wm8994->dev, -1,
03.                       wm8994_regulator_devs,
04.                       ARRAY_SIZE(wm8994_regulator_devs),
05.                       NULL, 0);
```

因为WM1811，WM8994，WM8958三个芯片功能类似，因此这三个芯片都使用了WM8994的代码，所以wm8994_device_init()接下来根据不同的芯片型号做了一些初始化动作，这部分的代码就不贴了。接着，从platform_data中获得部分配置信息：

```
[html]
01. if (pdata) {
02.     wm8994->irq_base = pdata->irq_base;
```



```

03.     wm8994->gpio_base = pdata->gpio_base;
04.
05.     /* GPIO configuration is only applied if it's non-zero */
06.     .....
07. }

```

最后，初始化irq，然后添加codec子设备和gpio子设备：

```

[html]
01. wm8994_irq_init(wm8994);
02.
03. ret = mfd_add_devices(wm8994->dev, -1,
04.     wm8994_devs, ARRAY_SIZE(wm8994_devs),
05.     NULL, 0);

```

经过以上这些处理后，作为父设备的I2C设备已经准备就绪，它的下面挂着4个子设备：ldo-0，ldo-1，codec，gpio。其中，codec子设备的加入，它将会和前面所讲codec的platform driver匹配，触发probe回调完成下面所说的codec驱动的初始化工作。

5. Codec初始化

Machine驱动的初始化，codec和dai的注册，都会调用snd_soc_instantiate_cards()进行一次声卡和codec，dai，platform的匹配绑定过程，这里所说的绑定，正如Machine驱动一文中所描述，就是通过3个全局链表，按名字进行匹配，把匹配的codec，dai和platform实例赋值给声卡每对dai的snd_soc_pcm_runtime变量中。一旦绑定成功，将会使得codec和dai驱动的probe回调被调用，codec的初始化工作就在该回调中完成。对于WM8994，该回调就是wm8994_codec_probe函数：

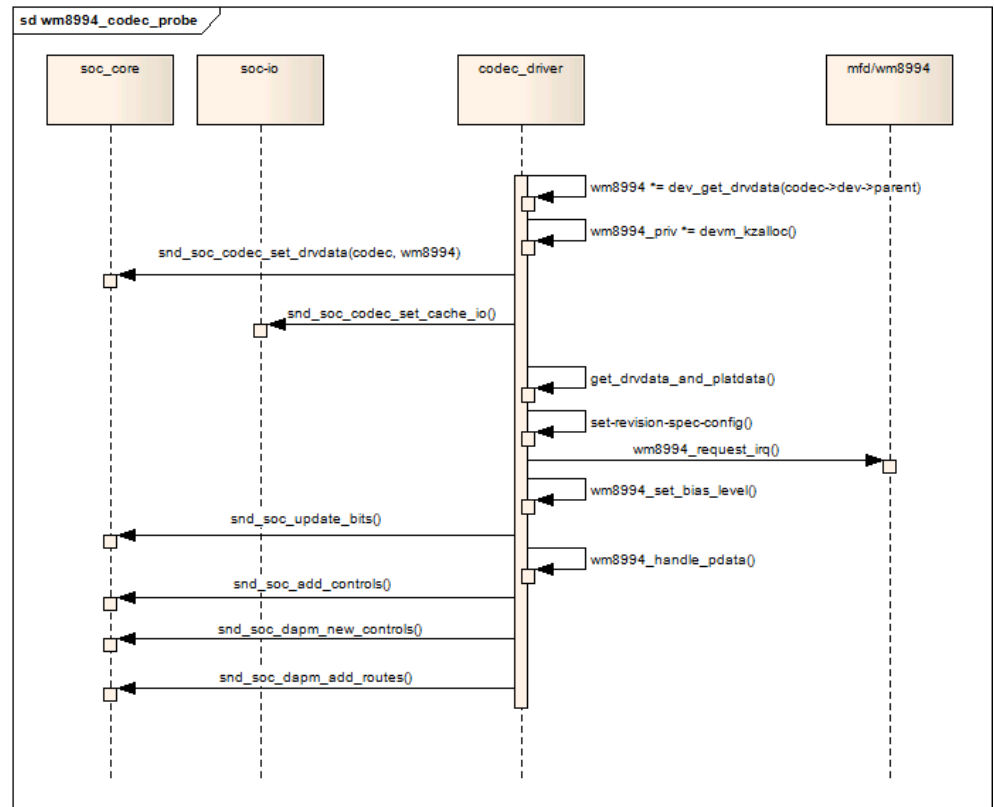


图5.1 wm8994_codec_probe

- 取出父设备的driver_data，其实就是上一节的wm8994结构变量，取出其中的regmap字段，复制到codec的control_data字段中；
- 申请一个wm8994_priv私有数据结构，并把它设为codec设备的driver_data；
- 通过snd_soc_codec_set_cache_io初始化regmap io，完成这一步后，就可以使用API：snd_soc_read()，snd_soc_write()对codec的寄存器进行读写了；
- 把父设备的driver_data（struct wm8994）和platform_data保存到私有结构wm8994_priv中；
- 因为要同时支持3个芯片型号，这里要根据芯片的型号做一些特定的初始化工作；
- 申请必要的几个中断；
- 设置合适的偏置电平；
- 通过snd_soc_update_bits修改某些寄存器；

- 根据父设备的platform_data，完成特定于平台的初始化配置；
- 添加必要的control，dapm部件进而dapm路由信息；

至此，codec驱动的初始化完成。

5. regmap-io

我们知道，要想对codec进行控制，通常都是通过读写它的内部寄存器完成的，读写的接口通常是I2C或者是SPI接口，不过每个codec芯片寄存器的比特位组成都有所不同，寄存器地址的比特位也有所不同。例如WM8753的寄存器地址是7bits，数据是9bits，WM8993的寄存器地址是8bits，数据也是16bits，而WM8994的寄存器地址是16bits，数据也是16bits。在kernel3.1版本，内核引入了一套regmap机制和相关的API，这样就可以用统一的操作来实现对这些多样的寄存器的控制。regmap使用起来也相对简单：

- 为codec定义一个regmap_config结构实例，指定codec寄存器的地址和数据位等信息；
- 根据codec的控制总线类型，调用以下其中一个函数，得到一个指向regmap结构的指针：
 - struct regmap *regmap_init_i2c(struct i2c_client *i2c, const struct regmap_config *config);
 - struct regmap *regmap_init_spi(struct spi_device *dev, const struct regmap_config *config);
- 把获得的regmap结构指针赋值给codec->control_data；
- 调用soc-io的api：snd_soc_codec_set_cache_io使得soc-io和regmap进行关联；

完成以上步骤后，codec驱动就可以使用诸如snd_soc_read、snd_soc_write、snd_soc_update_bits等API对codec的寄存器进行读写了。

更多 1

上一篇：[Linux ALSA声卡驱动之六：ASoC架构中的Machine](#)

下一篇：[Linux ALSA声卡驱动之八：ASoC架构中的Platform](#)

传感器

传感器

teamview

linux

游戏

传感器

查看评论

4楼 saltlight 2013-05-17 00:58发表



楼主你好！我在调试让我的片子wm8580支持24bits播放的时候，出现了underrun的现象，从网上找了些资料说是调整buffer-time就可以。于是我aplay --buffer-time=9000，可执行一会还是会出现underrun。而且声音听到的都是带杂音的(每个一会就有，非常规律)，但基本音轨可以听得到。不知楼主有什么建议么？期待回复 谢谢
软硬件环境(2.6.35.7+iis+wm8580+s5pv210)

3楼 天才2012 2012-06-25 16:25发表



在这里为何Codec和dai都要进行一次调用snd_soc_instantiate_cards函数触发一次匹配绑定的操作，这个函数里面会做probe的调用，那这样是不是会重复呢？

2楼 帅得不敢出门 2012-04-07 17:47发表



kernel是如何进入wm8994_i2c_probe的，光有init是无法自动调用probe的

Re: geekguy 2012-07-01 14:27发表



回复zmlovelx: 这个是linux驱动的平台设备的问题。目前大多数linux驱动都不直接在init中建立设备文件了，而是在probe中。这个函数需要将平台设备应编码到系统内核中，像rtc、lcd都是这么做的。系统检测到由响应的平台设备，就会调用probe函数了，然后就是一些列初始化动作。

Re: DroidPhone 2012-04-07 21:03发表



回复zmlovelx: 首先，你的板级代码中要先定义和注册一个wm8994的i2c设备，然后，wm8994_i2c_init(void)调用add_i2c_driver后触发i2c bus设备和驱动的匹配，匹配成功后wm8994_i2c_probe自然会被调用了。

1楼 hainei_ 2012-02-24 09:57发表



你好,我最近做的东西要用到ALSA,看了你的文章很有启发,能否加个QQ:568865992

Re: DroidPhone 2012-02-25 21:59发表



回复hainei_: QQ比较少用，博客的主页面上有我的联系方式，谢谢！

您还没有登录,请[\[登录\]](#)或[\[注册\]](#)

* 以上用户言论只代表其个人观点，不代表CSDN网站的观点或立场

专区推荐内容

使用 Thread 类创建与启动...
下载QNX白皮书助您完成卓越的嵌...
在HTML5中，如何从C#执行网...
如何从Javascript传送字...
Hadoop4Win+Eclip...
开启 Android 应用开发新...



更多招聘职位

我公司职位也要出现在这里

【广州友魄信息科技有限公司】Asp.net (C#) 开发工程师
【北京奥鹏远程教育中心有限公司】专职/兼职IT培训讲师
(Web前端开发, PHP, 3G开发方向)
【langoon】java架构师
【北京富邦天成信息技术有限公司】java高级工程师
【南京科莱斯企业管理咨询有限公司】Flex开发工程师

核心技术类目

全部主题 Java VPN Android iOS ERP IE10 Eclipse CRM JavaScript Ubuntu NFC
WAP jQuery 数据库 BI HTML5 Spring Apache Hadoop .NET API HTML SDK IIS
Fedora XML LBS Unity Splashtop UML components Windows Mobile Rails QEMU KDE
Cassandra CloudStack FTC coremail OPhone CouchBase 云计算 iOS6 Rackspace
Web App SpringSide Maemo Compuware 大数据 aptech Perl Tornado Ruby Hibernate
ThinkPHP Spark HBase Pure Solr Angular Cloud Foundry Redis Scala Django
Bootstrap

公司简介 | 招贤纳士 | 广告服务 | 银行汇款帐号 | 联系方式 | 版权声明 | 法律顾问 | 问题报告

QQ客服 微博客服 论坛反馈 联系邮箱: webmaster@csdn.net 服务热线: 400-600-2320

京 ICP 证 070598 号

北京创新乐知信息技术有限公司 版权所有

世纪乐知(北京)网络技术有限公司 提供技术支持

江苏乐知网络技术有限公司 提供商务支持

Copyright © 1999-2012, CSDN.NET, All Rights Reserved



DroidPhone的专栏

欢迎各位大虾交流，本人联系方式：droid.phx@gmail.com

目录视图 摘要视图 RSS 订阅

个人资料



DroidPhone



访问：382343次

积分：3434分

排名：第2116名

原创：46篇 转载：0篇

译文：4篇 评论：356条

文章搜索

文章分类

移动开发之Android (11)

Linux内核架构 (15)

Linux设备驱动 (16)

Linux电源管理 (3)

Linux音频子系统 (15)

Linux中断子系统 (5)

Linux时间管理系统 (8)

Linux输入子系统 (4)

文章存档

2013年11月 (4)

2013年10月 (3)

2013年07月 (3)

2012年12月 (4)

2012年10月 (4)

展开

阅读排行

Android Audio System 之 (38982)

Android Audio System 之 (25553)

Android Audio System 之 (25317)

Linux ALSA声卡驱动之一 (24001)

Linux ALSA声卡驱动之二 (18421)

Android SurfaceFlinger

投票赢好礼，周周有惊喜！ 2014年4月微软MVP申请开始了！ 消灭0回答，赢下载分 “我的2013”年度征文活动火爆进行中！ 办公大师系列经典丛书 诚聘英才

Linux ALSA声卡驱动之八：ASoC架构中的Platform

分类：Linux设备驱动 Linux音频子系统

2012-03-13 14:56

10644人阅读

评论(30)

收藏

举报

linux buffer codec playback struct stream

目录(?)

[-]

1. Platform驱动在ASoC中的作用
2. snd_soc_platform_driver的注册
3. cpu的snd_soc_dai driver驱动的注册
4. snd_soc_dai_driver中的ops字段
5. snd_soc_platform_driver中的ops字段
6. 音频数据的dma操作
 1. 申请dma buffer
 2. dma buffer管理

1. Platform驱动在ASoC中的作用

前面几章内容已经说过，ASoC被分为Machine，Platform和Codec三大部件，Platform驱动的主要作用是完成音频数据的管理，最终通过CPU的数字音频接口（DAI）把音频数据传送给Codec进行处理，最终由Codec输出驱动耳机或者是喇叭的音信号。在具体实现上，ASoC有把Platform驱动分为两个部分：snd_soc_platform_driver和snd_soc_dai_driver。其中，platform_driver负责管理音频数据，把音频数据通过dma或其他操作传送到cpu dai中，dai_driver则主要完成cpu一侧的dai的参数配置，同时也会通过一定的途径把必要的dma等参数与snd_soc_platform_driver进行交互。

/*****

声明：本博内容均由http://blog.csdn.net/droidphone原创，转载请注明出处，谢谢！

/*****

2. snd_soc_platform_driver的注册

通常，ASoC把snd_soc_platform_driver注册为一个系统的platform_driver，不要被这两个相像的术语所迷惑，前者只是针对ASoC子系统的，后者是来自Linux的设备驱动模型。我们要做的就是：

- 定义一个snd_soc_platform_driver结构的实例；
- 在platform_driver的probe回调中利用ASoC的API：snd_soc_register_platform()注册上面定义的实例；
- 实现snd_soc_platform_driver中的各个回调函数；

以kernel3.3中的/sound/soc/samsung/dma.c为例：

```
[cpp]
01. static struct snd_soc_platform_driver samsung_asoc_platform = {
02.     .ops      = &dma_ops,
03.     .pcm_new   = dma_new,
04.     .pcm_free  = dma_free_dma_buffers,
05. };
06.
07. static int __devinit samsung_asoc_platform_probe(struct platform_device *pdev)
08. {
09.     return snd_soc_register_platform(&pdev->dev, &samsung_asoc_platform);
10. }
11.
12. static int __devexit samsung_asoc_platform_remove(struct platform_device *pdev)
13. {
14.     snd_soc_unregister_platform(&pdev->dev);
15.     return 0;
```

Linux ALSA声卡驱动之三 (18248)

Android中的sp和wp指针 (17112)

Linux ALSA声卡驱动之七 (13786)

Android SurfaceFlinger (13061)

(12550)

评论排行

Android Audio System 之 (49)

Linux ALSA声卡驱动之八 (30)

Android SurfaceFlinger (21)

Linux ALSA声卡驱动之二 (18)

Linux ALSA声卡驱动之三 (16)

Android Audio System 之 (16)

Linux中断（interrupt）子 (15)

Android中的sp和wp指针 (13)

Linux中断（interrupt）子 (12)

Android SurfaceFlinger (11)

推荐文章

* SharePoint 2010/2013 使用 Javascript来判断权限的三种方法

* 坚持前进的方向：总结 2013，规划2014

* 创业者那些鲜为人知的事情

* ListView具有多种item布局——实现微信对话框

* 实现自己的类加载时，重写方法loadClass与findClass的区别

* GDAL影像投影转换

最新评论

Linux输入子系统：多点触控协议 gocy123: 很有用，多谢分享

ALSA声卡驱动中的DAPM详解之 wsc_168: 楼主,您好：现在正在移植wm8962的驱动，遇到了一些问题，请教一下您。串口信息显示已经扫描...

Linux ALSA声卡驱动之五：移动i slcsss: @DroidPhone:感谢您的回复，我是新手，想问下这个配置的具体位置在哪里？

Linux ALSA声卡驱动之五：移动i DroidPhone: @u013222557:这种情况通常是你的codec中的音频路径把Mic至HP的路径被打开了，请仔细...

Linux ALSA声卡驱动之五：移动i slcsss: 您好：我是一名在读研究生，最近在ALSA架构下搞嵌入式音频程序开发，遇到了一个棘手的问题...

ALSA声卡驱动中的DAPM详解之 DroidPhone: @u012389631:和电源管理和音频路径相关的control需要定义为dapm control (...)

ALSA声卡驱动中的DAPM详解之 elliepfang: 大侠，你好！这两天把您的文章1-7 看了一遍，关于control这个概念在您的文章中有提到过多次...

ALSA声卡驱动中的DAPM详解之 elliepfang: @DroidPhone:因为这个是wm8962的machine上的现有代码，但如果我要porting...

```
16.     }
17.
18.     static struct platform_driver asoc_dma_driver = {
19.         .driver = {
20.             .name = "samsung-audio",
21.             .owner = THIS_MODULE,
22.         },
23.
24.         .probe = samsung_asoc_platform_probe,
25.         .remove = __devexit_p(samsung_asoc_platform_remove),
26.     };
27.
28.     module_platform_driver(asoc_dma_driver);
```

snd_soc_register_platform() 该函数用于注册一个snd_soc_platform，只有注册以后，它才可以被Machine驱动使用。它的代码已经清晰地表达了它的实现过程：

- 为snd_soc_platform实例申请内存；
- 从platform_device中获得它的名字，用于Machine驱动的匹配工作；
- 初始化snd_soc_platform的字段；
- 把snd_soc_platform实例连接到全局链表platform_list中；
- 调用snd_soc_instantiate_cards，触发声卡的machine、platform、codec、dai等的匹配工作；

3. cpu的snd_soc_dai driver驱动的注册

dai驱动通常对应cpu的一个或几个I2S/PCM接口，与snd_soc_platform一样，dai驱动也是实现为一个platform driver，实现一个dai驱动大致可以分为以下几个步骤：

- 定义一个snd_soc_dai_driver结构的实例；
- 在对应的platform_driver中的probe回调中通过API：snd_soc_register_dai或者snd_soc_register_dais，注册snd_soc_dai实例；
- 实现snd_soc_dai_driver结构中的probe、suspend等回调；
- 实现snd_soc_dai_driver结构中的snd_soc_dai_ops字段中的回调函数；

snd_soc_register_dai 这个函数在上一篇介绍codec驱动的博文中已有介绍，请参考：[Linux ALSA声卡驱动之七：ASoC架构中的Codec](#)。

snd_soc_dai 该结构在snd_soc_register_dai函数中通过动态内存申请获得，简要介绍一下几个重要字段：

- driver 指向关联的snd_soc_dai_driver结构，由注册时通过参数传入；
- playback_dma_data 用于保存该dai播放stream的dma信息，例如dma的目标地址，dma传送单元大小和通道号等；
- capture_dma_data 同上，用于录音stream；
- platform 指向关联的snd_soc_platform结构；

snd_soc_dai_driver 该结构需要自己根据不同的soc芯片进行定义，关键字段介绍如下：

- probe、remove 回调函数，分别在声卡加载和卸载时被调用；
- suspend、resume 电源管理回调函数；
- ops 指向snd_soc_dai_ops结构，用于配置和控制该dai；
- playback_snd_soc_pcm_stream结构，用于指出该dai支持的声道数，码率，数据格式等能力；
- capture_snd_soc_pcm_stream结构，用于指出该dai支持的声道数，码率，数据格式等能力；

4. snd_soc_dai_driver中的ops字段

ops字段指向一个snd_soc_dai_ops结构，该结构实际上是一组回调函数的集合，dai的配置和控制几乎都是通过这些回调函数来实现的，这些回调函数基本可以分为3大类，驱动程序可以根据实际情况实现其中的一部分：

工作时钟配置函数 通常由machine驱动调用：

- set_sysclk 设置dai的主时钟；
- set_pll 设置PLL参数；
- set_clkdiv 设置分频系数；
- dai的格式配置函数 通常由machine驱动调用；
- set_fmt 设置dai的格式；

Linux ALSA声卡驱动之六: ASoC
elliepfang: @DroidPhone:关于
snd_soc_dai_link结构中的代码
注释中struct snd_...

ALSA声卡驱动中的DAPM详解之
DroidPhone: @u012389631:这
种名字根据实际的意义自己定义
就好了, 只要符合常识即可。不
过通常还是会和co...

- **set_tdm_slot** 如果dai支持时分复用, 用于设置时分复用的slot;
- **set_channel_map** 声道的时分复用映射设置;
- **set_tristate** 设置dai引脚的状态, 当与其他dai并联使用同一引脚时需要使用该回调;

标准的snd_soc_ops回调 通常由soc-core在进行PCM操作时调用:

- startup
- shutdown
- hw_params
- hw_free
- prepare
- trigger

抗pop, pop声 由soc-core调用:

- digital_mute

以下这些api通常被machine驱动使用, machine驱动在他的snd_pcm_ops字段中的hw_params回调中使用这些api:

- **snd_soc_dai_set_fmt()** 实际上会调用snd_soc_dai_ops或者codec driver中的set_fmt回调;
- **snd_soc_dai_set_pll()** 实际上会调用snd_soc_dai_ops或者codec driver中的set_pll回调;
- **snd_soc_dai_set_sysclk()** 实际上会调用snd_soc_dai_ops或者codec driver中的set_sysclk回调;
- **snd_soc_dai_set_clkdiv()** 实际上会调用snd_soc_dai_ops或者codec driver中的set_clkdiv回调;

snd_soc_dai_set_fmt(struct snd_soc_dai *dai, unsigned int fmt)的第二个参数fmt在这里特别说一下, ASoC目前只是用了它的低16位, 并且为它专门定义了一些宏来方便我们使用:

bit 0-3 用于设置接口的格式:

```
[cpp]
01. #define SND_SOC_DAIFMT_I2S      1 /* I2S mode */
02. #define SND_SOC_DAIFMT_RIGHT_J  2 /* Right Justified mode */
03. #define SND_SOC_DAIFMT_LEFT_J   3 /* Left Justified mode */
04. #define SND_SOC_DAIFMT_DSP_A    4 /* L data MSB after FRM LRC */
05. #define SND_SOC_DAIFMT_DSP_B    5 /* L data MSB during FRM LRC */
06. #define SND_SOC_DAIFMT_AC97     6 /* AC97 */
07. #define SND_SOC_DAIFMT_PDM      7 /* Pulse density modulation */
```

bit 4-7 用于设置接口时钟的开关特性:

```
[cpp]
01. #define SND_SOC_DAIFMT_CONT     (1 << 4) /* continuous clock */
02. #define SND_SOC_DAIFMT_GATED    (2 << 4) /* clock is gated */
```

bit 8-11 用于设置接口时钟的相位:

```
[cpp]
01. #define SND_SOC_DAIFMT_NB_NF   (1 << 8) /* normal bit clock + frame */
02. #define SND_SOC_DAIFMT_NB_IF   (2 << 8) /* normal BCLK + inv FRM */
03. #define SND_SOC_DAIFMT_IB_NF   (3 << 8) /* invert BCLK + nor FRM */
04. #define SND_SOC_DAIFMT_IB_IF   (4 << 8) /* invert BCLK + FRM */
```

bit 12-15 用于设置接口主从格式:

```
[cpp]
01. #define SND_SOC_DAIFMT_CBM_CFM (1 << 12) /* codec clk & FRM master */
02. #define SND_SOC_DAIFMT_CBS_CFM (2 << 12) /* codec clk slave & FRM master */
03. #define SND_SOC_DAIFMT_CBM_CFS (3 << 12) /* codec clk master & frame slave */
04. #define SND_SOC_DAIFMT_CBS_CFS (4 << 12) /* codec clk & FRM slave */
```

5. snd_soc_platform_driver中的ops字段

该ops字段是一个snd_pcm_ops结构, 实现该结构中的各个回调函数是soc platform驱动的主要工作, 他们基本都涉及dma操作以及dma buffer的管理等工作。下面介绍几个重要的回调函数:

ops.open

当应用程序打开一个pcm设备时，该函数会被调用，通常，该函数会使用snd_soc_set_runtime_hwparams()设置substream中的snd_pcm_runtime结构里面的hw_params相关字段，然后为snd_pcm_runtime的private_data字段申请一个私有结构，用于保存该平台的dma参数。

ops.hw_params

驱动的hw_params阶段，该函数会被调用。通常，该函数会通过snd_soc_dai_get_dma_data函数获得对应的dai的dma参数，获得的参数一般都会保存在snd_pcm_runtime结构的private_data字段。然后通过snd_pcm_set_runtime_buffer函数设置snd_pcm_runtime结构中的dma buffer的地址和大小等参数。要注意的是，该回调可能会被多次调用，具体实现时要小心处理多次申请资源的问题。

ops.prepare

正式开始数据传送之前会调用该函数，该函数通常会完成dma操作的必要准备工作。

ops.trigger

数据传送的开始，暂停，恢复和停止时，该函数会被调用。

ops.pointer

该函数返回传送数据的当前位置。

6. 音频数据的dma操作

soc-platform驱动的最主要功能就是要完成音频数据的传送，大多数情况下，音频数据都是通过dma来完成的。

6.1. 申请dma buffer

因为dma的特殊性，dma buffer是一块特殊的内存，比如有的平台规定只有某段地址范围的内存才可以进行dma操作，而多数嵌入式平台还要求dma内存的物理地址是连续的，以方便dma控制器对内存的访问。在ASoC架构中，dma buffer的信息保存在snd_pcm_substream结构的snd_dma_buffer *buf字段中，它的定义如下

```
[cpp]
01. struct snd_dma_buffer {
02.     struct snd_dma_device dev; /* device type */
03.     unsigned char *area; /* virtual pointer */
04.     dma_addr_t addr; /* physical address */
05.     size_t bytes; /* buffer size in bytes */
06.     void *private_data; /* private for allocator; don't touch */
07. };
```

那么，在哪里完成了snd_dam_buffer结构的初始化赋值操作呢？答案就在snd_soc_platform_driver的pcm_new回调函数中，还是以sound/soc/samsung/dma.c为例：

```
[cpp]
01. static struct snd_soc_platform_driver samsung_asoc_platform = {
02.     .ops = &dma_ops,
03.     .pcm_new = dma_new,
04.     .pcm_free = dma_free_dma_buffers,
05. };
06.
07. static int __devinit samsung_asoc_platform_probe(struct platform_device *pdev)
08. {
09.     return snd_soc_register_platform(&pdev->dev, &samsung_asoc_platform);
10. }
```

pcm_new字段指向了dma_new函数，dma_new函数进一步为playback和capture分别调用preallocate_dma_buffer函数，我们看看preallocate_dma_buffer函数的实现：

```
[cpp]
01. static int preallocate_dma_buffer(struct snd_pcm *pcm, int stream)
02. {
03.     struct snd_pcm_substream *substream = pcm->streams[stream].substream;
04.     struct snd_dma_buffer *buf = &substream->dma_buffer;
```



```

05.     size_t size = dma hardware.buffer_bytes_max;
06.
07.     pr_debug("Entered %s\n", __func__);
08.
09.     buf->dev.type = SNDRV_DMA_TYPE_DEV;
10.     buf->dev.dev = pcm->card->dev;
11.     buf->private_data = NULL;
12.     buf->area = dma_alloc_writecombine(pcm->card->dev, size,
13.                                       &buf->addr, GFP_KERNEL);
14.     if (!buf->area)
15.         return -ENOMEM;
16.     buf->bytes = size;
17.     return 0;
18. }

```

该函数先是获得事先定义好的buffer大小，然后通过dma_alloc_writecombine函数分配dma内存，然后完成substream->dma_buffer的初始化赋值工作。上述的pcm_new回调会在声卡的建立阶段被调用，调用的详细的过程请参考Linux ALSAs 声卡驱动之六：ASoC架构中的Machine中的图3.1。

在声卡的hw_params阶段，snd_soc_platform_driver结构的ops->hw_params会被调用，在该回调调用，通常会使用api: snd_pcm_set_runtime_buffer()把substream->dma_buffer的数值拷贝到substream->runtime的相关字段中(.dma_area, .dma_addr, .dma_bytes)，这样以后就可以通过substream->runtime获得这些地址和大小信息了。

dma buffer获得后，即是获得了dma操作的源地址，那么目的地址在哪里？其实目的地址当然是在dai中，也就是前面介绍的snd_soc_dai结构的playback_dma_data和capture_dma_data字段中，而这两个字段的值也是在hw_params阶段，由snd_soc_dai_driver结构的ops->hw_params回调，利用api: snd_soc_dai_set_dma_data进行设置的。紧随其后，snd_soc_platform_driver结构的ops->hw_params回调利用api: snd_soc_dai_get_dma_data获得这些dai的dma信息，其中就包括了dma的目的地址信息。这些dma信息通常还会被保存在substream->runtime->private_data中，以便在substream的整个生命周期中可以随时获得这些信息，从而完成对dma的配置和操作。

6.2 dma buffer管理

播放时，应用程序把音频数据源源不断地写入dma buffer中，然后相应platform的dma操作则不停地从该buffer中取出数据，经dai送往codec中。录音时则正好相反，codec源源不断地把A/D转换好的音频数据经过dai送入dma buffer中，而应用程序则不断地从该buffer中读走音频数据。

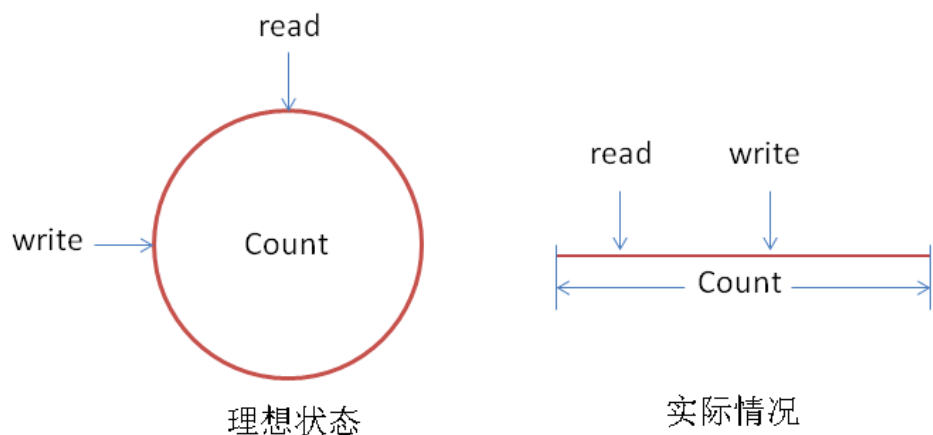


图6.2.1 环形缓冲区

环形缓冲区正好适用于这种情景的buffer管理，理想情况下，大小为Count的缓冲区具备一个读指针和写指针，我们期望他们都可以闭合地做环形移动，但是实际的情况确实：缓冲区通常都是一段连续的地址，他是有开始和结束两个边界，每次移动之前都必须进行一次判断，当指针移动到末尾时就必须人为地让他回到起始位置。在实际应用中，我们通常都会把这个大小为Count的缓冲区虚拟成一个大小为n*Count的逻辑缓冲区，相当于理想状态下的圆形绕了n圈之后，然后把这段总的距离拉平为一段直线，每一圈对应直线中的一段，因为n比较大，所以大多数情况下不会出现读写指针的换位的情况（如果不对buffer进行扩展，指针到达末端后，回到起始端时，两个指针的前后相对位置会发生互换）。扩展后的逻辑缓冲区在计算剩余空间可条件判断是相对方便。alsa driver也使用了该方法对dma buffer进行管理：

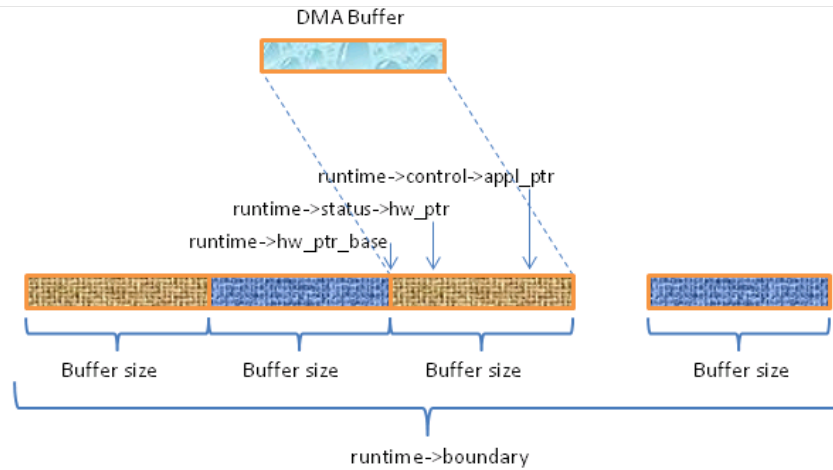


图6.2.2 alsa driver缓冲区管理

snd_pcm_runtime结构中，使用了四个相关的字段来完成这个逻辑缓冲区的管理：

- snd_pcm_runtime.hw_ptr_base 环形缓冲区每一圈的基地址，当读写指针越过一圈后，它按buffer size进行移动；
- snd_pcm_runtime.status->hw_ptr 硬件逻辑位置，播放时相当于读指针，录音时相当于写指针；
- snd_pcm_runtime.control->appl_ptr 应用逻辑位置，播放时相当于写指针，录音时相当于读指针；
- snd_pcm_runtime.boundary 扩展后的逻辑缓冲区大小，通常是 $(2^n) \times \text{size}$ ；

通过这几个字段，我们可以很容易地获得缓冲区的有效数据，剩余空间等信息，也可以很容易地把当前逻辑位置映射回真实的dma buffer中。例如，获得播放缓冲区的空闲空间：

```
[csharp]
01. static inline snd_pcm_uframes_t snd_pcm_playback_avail(struct snd_pcm_runtime *runtime)
02. {
03.     snd_pcm_uframes_t avail = runtime->status->hw_ptr + runtime->buffer_size - runtime->control->appl_ptr;
04.     if (avail < 0)
05.         avail += runtime->boundary;
06.     else if ((snd_pcm_uframes_t) avail >= runtime->boundary)
07.         avail -= runtime->boundary;
08.     return avail;
09. }
```

要想映射到真正的缓冲区位置，只要减去runtime->hw_ptr_base即可。下面的api用于更新这几个指针的当前位置：

```
[cpp]
01. int snd_pcm_update_hw_ptr(struct snd_pcm_substream *substream)
```

所以要想通过snd_pcm_playback_avail等函数获得正确的信息前，应该先要调用这个api更新指针位置。

以播放(playback)为例，我现在知道至少有3个途径可以完成对dma buffer的写入：

- 应用程序调用alsa-lib的snd_pcm_writei、snd_pcm_writen函数；
- 应用程序使用ioctl：SNDRV_PCM_IOCTL_WRITEI_FRAMES或SNDRV_PCM_IOCTL_WRITEN_FRAMES；
- 应用程序使用alsa-lib的snd_pcm_mmap_begin/snd_pcm_mmap_commit；

以上几种方式最终把数据写入dma buffer中，然后修改runtime->control->appl_ptr的值。

播放过程中，通常会配置成每一个period size生成一个dma中断，中断处理函数最重要的任务就是：

- 更新dma的硬件的当前位置，该数值通常保存在runtime->private_data中；
- 调用snd_pcm_period_elapsed函数，该函数会进一步调用snd_pcm_update_hw_ptr0函数更新上述所说的4个缓冲区管理字段，然后唤醒相应的等待进程；

```
[cpp]
01. <span style="font-family:Arial, Verdana, sans-serif;"><span style="white-space: normal;">
```

```

02.  }
03.      struct snd_pcm_runtime *runtime;
04.      unsigned long flags;
05.
06.      if (PCM_RUNTIME_CHECK(substream))
07.          return;
08.      runtime = substream->runtime;
09.
10.      if (runtime->transfer_ack_begin)
11.          runtime->transfer_ack_begin(substream);
12.
13.      snd_pcm_stream_lock_irqsave(substream, flags);
14.      if (!snd_pcm_running(substream) ||
15.          snd_pcm_update_hw_ptr0(substream, 1) < 0)
16.          goto _end;
17.
18.      if (substream->timer_running)
19.          snd_timer_interrupt(substream->timer, 1);
20.  _end:
21.      snd_pcm_stream_unlock_irqrestore(substream, flags);
22.      if (runtime->transfer_ack_end)
23.          runtime->transfer_ack_end(substream);
24.      kill_fasync(&runtime->fasync, SIGIO, POLL_IN);
25.  }
26. </pre>如果设置了transfer_ack_begin和transfer_ack_end回调，snd_pcm_period_elapsed还会调用这两个回
    调函数。<br>
27. <br>
28. <pre></pre>
29. <pre></pre>
30. <pre></pre>

```

7. 图说代码

最后，反正图也画了，好与不好都传上来供参考一下，以下这张图表达了 ASoC 中 Platform 驱动的几个重要数据结构之间的关系：

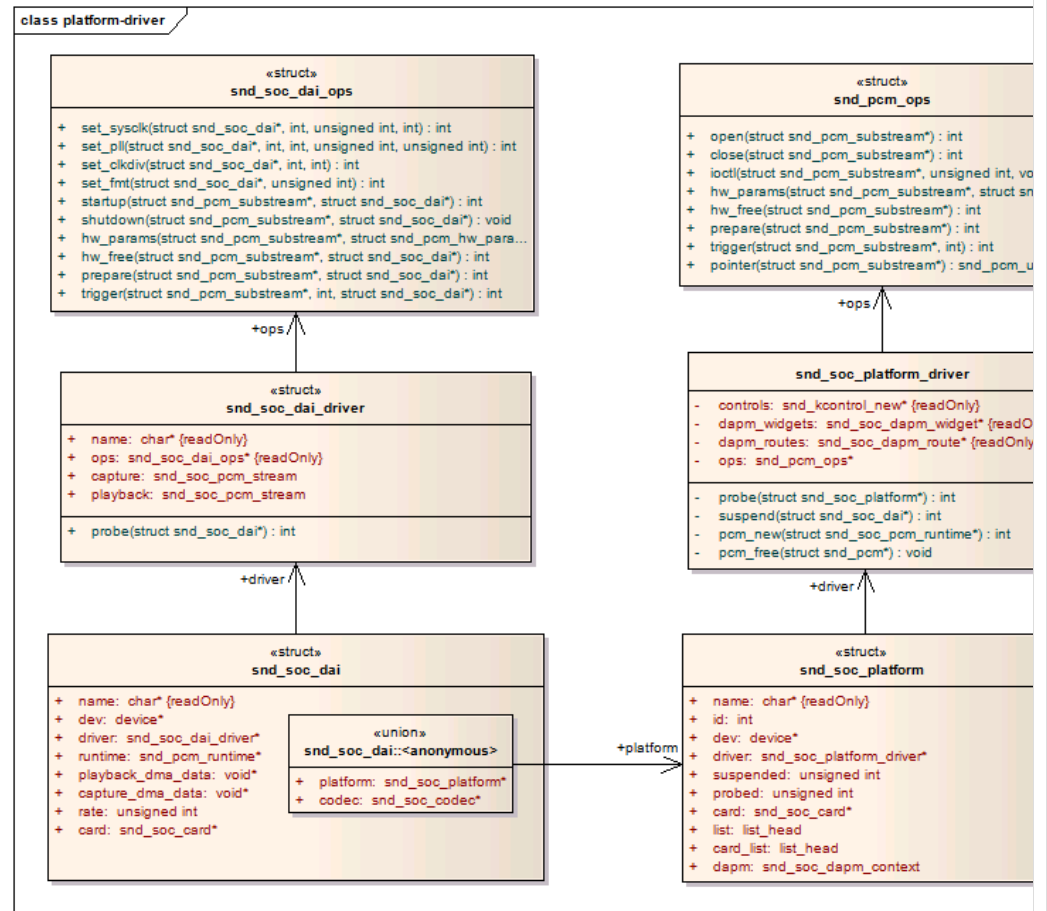


图7.1 ASoC Platform驱动

一堆的private_data，很重要但也很容易搞混，下面的图不知对大家有没有帮助：

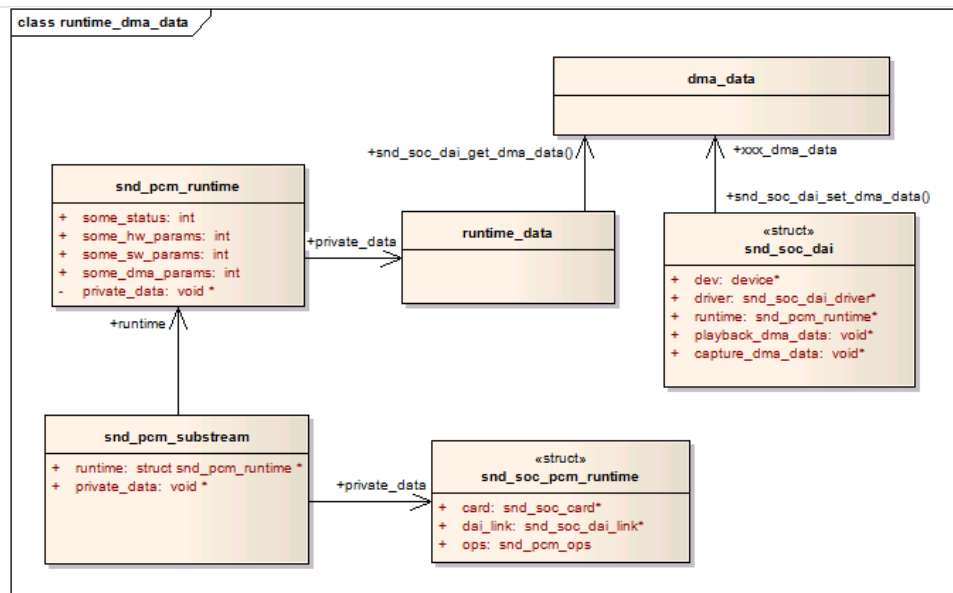


图7.2 private_data

更多 1

上一篇: [Linux ALSA声卡驱动之七: ASoC架构中的Codec](#)

下一篇: [自旋锁spin_lock和raw_spin_lock](#)

查看评论

22楼 [ljf69](#) 2013-11-21 09:10发表



楼主你好,

我在2416上面移植wm9713驱动, 移植后内核都认到声卡了, 但是没有声音, 不知道怎么回事。<http://bbs.csdn.net/topics/390647742>

这是具体移植过程。

希望楼主能帮我看下问题出现在哪里?

非常感谢!

21楼 [wo2581511](#) 2013-11-11 22:28发表



楼主, 你好! 请问一下, alsa架构具体的音量控制怎么体现?

Re: [DroidPhone](#) 2013-11-12 10:03发表



回复wo2581511: 通常codec都会有音量控制器, 把控制音量的寄存器定义成一个tlv形式的kcontrol, 用户空间控制该kcontrol就可以实现音量控制了。

Re: [wo2581511](#) 2013-11-13 10:44发表



回复DroidPhone: 谢谢! 我定义了这样一个kcontrol:

```

SOC_DOUBLE_R_SX_TLV("LineOut Analog Playback Volume", CS42L73_LOAAVOL,
CS42L73_LOBAVOL, 0, 0x41, 0x4B, hpaloa_tlv),
#define SOC_DOUBLE_R_SX_TLV(xname, xreg, xreg, xshift, xmin, xmax, tlv_array) \
{ .iface = SNDRV_CTL_ELEM_IFACE_MIXER, .name = (xname), \
.access = SNDRV_CTL_ELEM_ACCESS_TLV_READ | \
SNDRV_CTL_ELEM_ACCESS_READWRITE, \
.tlv.p = (tlv_array), \
.info = snd_soc_info_volsw, \
.get = snd_soc_get_volsw_sx, \
.put = snd_soc_put_volsw_sx, \
.private_value = (unsigned long)&(struct soc_mixer_control) \
{.reg = xreg, .reg = xreg, \
.shift = xshift, .rshift = xshift, \
.max = xmax, .min = xmin} }
  
```

调节音量大小, 比如音量减-1应该进入info或者put函数一次? 还是怎么弄的?

20楼 [maosuyun2009](#) 2013-09-05 18:19发表

楼主, 请问cpu的snd_soc_dai是在哪里添加进了dai_list链表了呢?



19楼 [hello_hui](#) 2013-07-10 23:28发表

请问如何得到IIS接口的数字音频数据，而不经DAC送到内置的codec里？



18楼 [lostman80](#) 2013-07-04 22:22发表

膜拜啊



17楼 [tian1112yong](#) 2013-06-29 16:31发表

读了两遍，还是有些地方不懂。多谢楼主



16楼 [泥瓦工](#) 2013-06-06 10:33发表

打印出来读上几遍，谢谢lz!



15楼 [Hens007](#) 2013-01-13 23:32发表

我的声卡也是用这个架构，但是没有声音，不知道是哪里出问题？或者说是出现这样的问题有哪些可能的原因？



14楼 [hellowxworld](#) 2012-12-12 11:09发表

分析的alsa框架非常清晰.



13楼 [guolie1989](#) 2012-10-16 10:07发表

第三次阅读，细读。越读越明朗。建议多读几遍。



12楼 [confidence321](#) 2012-10-11 17:18发表

hao,讲的很高深。谢谢分享



11楼 [zczhao2009](#) 2012-08-27 00:26发表

相见恨晚 谢谢



10楼 [Homage_Faith](#) 2012-08-20 13:56发表

你好！楼主，我有个问题想请教一下：我想在这个音频驱动架构里添加ADPCM 单元，以用来对PCM进行编解码，请问ADPCM应该是属于platform驱动这一边吧？如果把它加进来，那接口要怎样实现呢？



9楼 [haokaihaohe110](#) 2012-07-04 19:07发表

请问大侠，我使用snd_kcontrol_new建立了设置寄存器的控制接口，然后在asound.conf里面增加调用这个control的配置，但是通过打印信息发现这个control不能被调用，请问这是怎么回事啊？我使用的是android2.3。多谢大侠指点。



8楼 [天才2012](#) 2012-06-25 16:41发表

看完了，膜拜



7楼 [kobewylb](#) 2012-06-12 11:15发表

hi，博主，我现在碰到这么一个问题。

现象：

两个线程中一个需要不停地AudioRecord read，一个需要不停地AudioTrack write，现在发现有些设备上面AudioRecord read的数据明显有延时而且会越来越大，有时发现AudioTrack write的速度也会一下子变慢，但是CPU的占用率只有5%左右。

提问：

1.AudioRecord和AudioTrack有独占声卡这一说法吗，就是一个record 在read的时候，track 的write只能等吗？

2.还有AudioRecord read在有些设备上面还会有返回0的情况，每次返回0的时候都会等1s以上，而且下次独到的数据还是之前的，就是说他的延时是累积的，这个有解决办法吗？



6楼 [charmland](#) 2012-05-10 09:17发表

soc_platform的平台设备层

1.为soc_platform定义一个平台驱动实例：

```
static struct platform_driver xxx_pcm_driver = {
    .driver = {
        .name = "xxx-pcm-audio",
        .owner = THIS_MODULE,
    },
    .probe = xxx_soc_platform_probe,
```

```
.remove = __devexit_p(xxx_soc_platform_remove),
};
2.模块初始化module_init /exit中注册和移除xxx_pcm_driver;
3.实现probe/remove回调函数:
xxx_soc_platform_probe/remove: 注册和移除xxx_soc_platform
```

5楼 [dongquan360](#) 2012-04-26 10:49发表



你好,我正在做一个soc驱动,已经能够通过codec输出播放音乐,现在还要让spdif作音频输出,感觉spdif在asoc架构中像是一个codec的地位,如果驱动中有两个codec,怎么能做到切换codec输出呢

Re: [DroidPhone](#) 2012-04-27 11:01发表



回复dongquan360: 你列一下设备节点列表: ls /dev/snd, 看看有哪些名字, 如果没有我所说的pcmC0D1p,你就应该检查一下驱动。

Re: [dongquan360](#) 2012-04-27 11:15发表



```
回复DroidPhone: -bash-3.2# ls /dev/snd/
controlC0 pcmC0D0c pcmC0D0p pcmC0D1p seq timer
设备节点是有的,我可以直用aplay播放wav文件,指定--device参数时就不行。
我有两个设备,一个codec,一个spdif,默认的是codec输出音频
-bash-3.2# /usr/bin/aplay -l
**** List of PLAYBACK Hardware Devices ****
card 0: gw3000audio [gw3000-audio], device 0: S13V33 PCM i65gv25-0 []
Subdevices: 1/1
Subdevice #0: subdevice #0
card 0: gw3000audio [gw3000-audio], device 1: spdif DIT-1 []
Subdevices: 1/1
Subdevice #0: subdevice #0
```

Re: [DroidPhone](#) 2012-04-27 12:23发表



回复dongquan360: 你可以试试一下命令:
aplay -Dhw:0,1 xxxx.wav

Re: [DroidPhone](#) 2012-04-26 11:04发表



回复dongquan360: 设备节点的名称会不一样的啊,一个是pcmC0D0p,一个是pcmC0D1p。切换是用户空间应用的策略问题,与驱动无关。

Re: [dongquan360](#) 2012-04-27 10:32发表



```
回复DroidPhone: 我在应用层通过alsa的命令行工具aplay进行播放,加上--device=NAME选项选择设备,这个NAME我试过驱动中注册的name以及aplay -l列出的名字,都不行,提示类似于:
ALSA lib pcm.c:2212:(snd_pcm_open_noupdate) Unknown PCM $NAME(就是我通过命令行参数
--device传入的NAME)
aplay: main:682: audio open error: No such file or directory
不知道进行设备选择时名字有什么特殊格式要求还是怎么回事
```

4楼 [android_linux_2012](#) 2012-04-22 19:09发表



我的执行到hw_params就出现
asoc: machine hw_params failed
这是什么原因

3楼 [ljgmz](#) 2012-04-16 10:11发表



你好,我想问一下dai结构里面的那些.name成员有什么作用?上层是怎么调用dai的?希望楼主给解答一下,谢谢

Re: [DroidPhone](#) 2012-04-16 17:31发表



回复ljgmz: 请先看看这篇博文<http://blog.csdn.net/droidphone/article/details/7231605>中的struct snd_soc_dai_link中的定义, machine驱动会根据这些名字和dai结构中的名字匹配,匹配成功后machine驱动就可以操作和控制dai啦。

2楼 [hungyiyang](#) 2012-03-21 18:07发表



this is a very good article series I have ever seen. Thanks.

1楼 [cskyrain](#) 2012-03-21 11:36发表



目前为止看到分析alsa最好的文章, lz归纳的能力很强!

您还没有登录,请[登录](#)或[注册](#)

* 以上用户言论只代表其个人观点,不代表CSDN网站的观点或立场

专区推荐内容

超极本触摸屏一种全新的动手方式
[Intel CPU中的安全指令...
是时候结束数据中心冲突了?
什么是微服务器
优化、并行化、及向量化
如何从数据库调出数据显示到页面



更多招聘职位

我公司职位也要出现在这里

【南京科莱斯企业管理咨询有限公司】Flex开发工程师
(南京)
【深圳西洋镜网络技术有限公司】游戏服务端开发工程师
【博彦科技(上海)有限公司】oracle数据库开发
【长沙中科院文化创意与科技产业研究院】Java研发工程师

核心技术类目

全部主题 Java VPN Android iOS ERP IE10 Eclipse CRM JavaScript Ubuntu NFC
WAP jQuery 数据库 BI HTML5 Spring Apache Hadoop .NET API HTML SDK IIS
Fedora XML LBS Unity Splashtop UML components Windows Mobile Rails QEMU KDE
Cassandra CloudStack FTC coremail OPhone CouchBase 云计算 iOS6 Rackspace
Web App SpringSide Maemo Compuware 大数据 aptech Perl Tornado Ruby Hibernate
ThinkPHP Spark HBase Pure Solr Angular Cloud Foundry Redis Scala Django
Bootstrap

公司简介 | 招贤纳士 | 广告服务 | 银行汇款帐号 | 联系方式 | 版权声明 | 法律顾问 | 问题报告

QQ客服 微博客服 论坛反馈 联系邮箱: webmaster@csdn.net 服务热线: 400-600-2320

京 ICP 证 070598 号

北京创新乐知信息技术有限公司 版权所有

世纪乐知(北京)网络技术有限公司 提供技术支持

江苏乐知网络技术有限公司 提供商务支持

Copyright © 1999-2012, CSDN.NET, All Rights Reserved

