

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：1077193次
积分：8768
等级：
排名：第1448名
原创：51篇 转载：0篇
译文：4篇 评论：537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 (20)
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

2014年07月 (1)
2014年04月 (4)
2013年11月 (4)
2013年10月 (3)
2013年07月 (3)

展开

阅读排行

Linux ALSA声卡驱动之- (74187)
Android Audio System 之 (58758)
Linux ALSA声卡驱动之二 (46387)
Android Audio System 之 (42720)
Linux ALSA声卡驱动之三 (41553)
Linux ALSA声卡驱动之六

【公告】博客系统优化升级 Unity3D学习，离VR开发还有一步 伯乐招募开始啦 虚拟现实，一探究竟

Linux ALSA声卡驱动之六：ASoC架构中的Machine

标签：linux codec playback list 数据结构 struct

2012-02-03 19:09 37516人阅读 评论(16) 收藏 举报

分类：Linux音频子系统 (14) Linux设备驱动 (19)

版权声明：本文为博主原创文章，未经博主允许不得转载。

目录(?) [+]

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：

[cpp]

Linux时间子系统之六：ns (37505)
Android Audio System 之 (36411)
Linux ALSA声卡驱动之十 (36317)
Linux ALSA声卡驱动之四 (36146)
(31714)

评论排行

Android Audio System 之 (56)
Linux ALSA声卡驱动之三 (42)
Linux ALSA声卡驱动之九 (35)
Linux时间子系统之六：ns (25)
Linux中断（interrupt）子 (24)
Android SurfaceFlinger中 (21)
Linux ALSA声卡驱动之二 (19)
Android Audio System 之 (18)
Linux ALSA声卡驱动之十 (17)
Linux中断（interrupt）子 (17)

推荐文章

* 致JavaScript也将征服的物联网世界
* 从苏宁电器到卡巴斯基：难忘的三年硕士时光
* 作为一名基层管理者如何利用情商管理自己和团队（一）
* Android CircleImageView圆形ImageView
* 高质量代码的命名法则

最新评论

Linux输入子系统：输入设备编程u012839187: 看着没有问题。你要弄清楚的是新的内核有许多函数宏的变动。

Linux时间子系统之一：clock sourcezhqh100: 1.2 read回调函数时钟源本身不会产生中断，要获得时钟源的当前计数，只能通过主动调用它的rea...

Linux中断（interrupt）子系统之：liunix61: 大神！必须顶@！！

Linux时间子系统之三：时间的维Kevin_Smart: 学习了

Linux时间子系统之六：高精度定Kevin_Smart: 像楼主说的，原则上，hrtimer是利用一个硬件计数器来实现的，所以精度才可以做到ns级别。硬件的计...

Linux中断（interrupt）子系统之：12期-马金兴: 恩，虽然这么多字但是我要好好学习一下

已知二叉树的前序遍历和中序遍历重修月: 很喜欢楼主的文章，刚刚用豆约翰博客备份专家备份了您的全部博文。

Linux ALSA声卡驱动之三：PCV灿哥哥: 学习了

Android Audio System 之三：Alss0429: 楼主的文章写的很精炼，多谢分享~

Linux中断（interrupt）子系统之：KrisFei: 针对这句话有两个问题想讨论下：1. disable_irq()放在中断上半部会导致死锁。2. 如果...

```
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合设置工作后，调用了 card->late_probe(card)进行一些最后的初始化合设置工作，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Platform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：

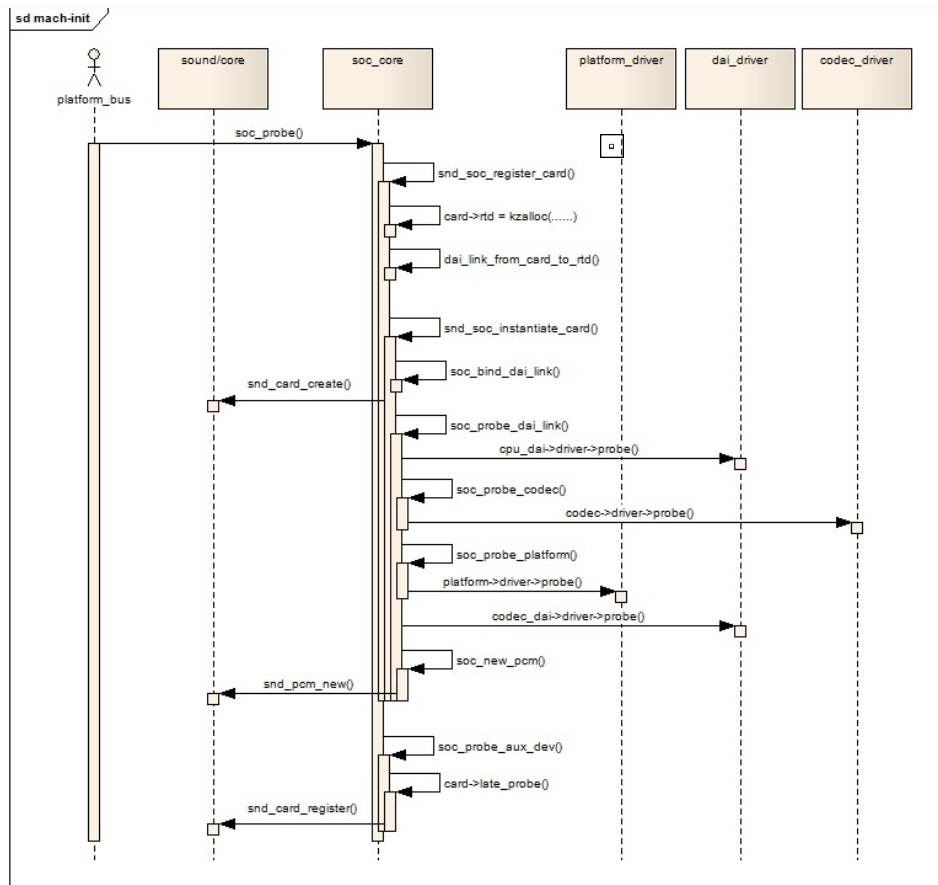


图3.1 基于3.0内核 soc_probe序列图

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：

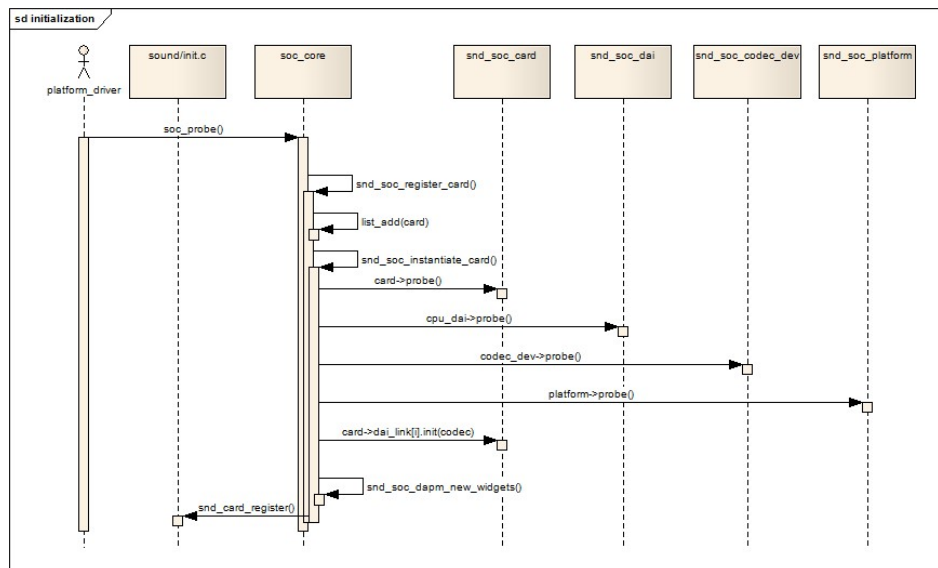


图3.2 基于2.6.35 soc_probe序列图

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](#)

我的同类文章

Linux音频子系统（14）	Linux设备驱动（19）
<ul style="list-style-type: none">• ALSA声卡驱动中的DAPM详... 2013-11-09 阅读 8979• ALSA声卡驱动中的DAPM详... 2013-11-04 阅读 12941• ALSA声卡驱动中的DAPM详... 2013-10-24 阅读 12568• ALSA声卡驱动中的DAPM详... 2013-10-18 阅读 17125• Linux ALSA声卡驱动之七：... 2012-02-23 阅读 36156	<ul style="list-style-type: none">• ALSA声卡驱动中的DAPM详... 2013-11-04 阅读 9950• ALSA声卡驱动中的DAPM详... 2013-11-01 阅读 10222• ALSA声卡驱动中的DAPM详... 2013-10-23 阅读 10804• Linux ALSA声卡驱动之八：... 2012-03-13 阅读 31315• Linux ALSA声卡驱动之五：... 2012-01-17 阅读 26715
更多文章	

参考知识库



算法与数据结构知识库


2034 关注 | 3500 收录

猜你在找

- | | |
|--|---|
| Linux嵌入式开发+驱动开发 | Linux ALSA声卡驱动之六ASoC架构中的Machine |
| Linux环境C语言编程基础 | Linux ALSA声卡驱动之六ASoC架构中的Machine |
| 企业级架构apache/PHP/tomcat架构应用实战 | Linux ALSA声卡驱动之六ASoC架构中的Machine |
| Linux设备驱动开发入门 | Linux ALSA声卡驱动之六ASoC架构中的Machine |
| Android底层技术：Linux驱动框架与开发 | Linux ALSA声卡驱动之六ASoC架构中的Machine |


查看评论

10楼 [qq_27935541](#) 2015-05-06 16:27发表




时序图太给力了。

9楼 [慢几步-深几度-前行](#) 2015-03-25 16:11发表




hello，这个流程图用什么软件画的，这种流程图很直观。

8楼 [qq721695846](#) 2014-03-29 23:35发表




我现在想弄个虚拟的(没有硬件)，把系统的声音截下再做我的事，不知道难不难

Re: [kangear](#) 2015-03-12 12:41发表



回复qq721695846：当然有虚拟的，比如HDMI就没有codec，就需要一个虚拟的。

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

Re: [cqcdscu999](#) 2014-02-25 14:36发表



回复u012389631：我也遇到相同的问题了。

我在流程图上面看到调用了这个dai_link_from_card_to_rtd () 函数，但我看代码的时候并没有看到该函数。自己先摸索了~

最后谢谢博主的文章!

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

Re: [kangear](#) 2015-03-12 12:40发表



回复lamaboy2012：USB接口的不存在不同的Machine之说，所以才用alsa就可以，目前的kernel中已经包含标准的Usb audio驱动的。

3楼 [天才2012](#) 2012-06-25 15:25发表



奇怪的是为何3.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，版本的区别你这里提到了，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

核心技术类目

[全部主题](#) [Hadoop](#) [AWS](#) [移动游戏](#) [Java](#) [Android](#) [iOS](#) [Swift](#) [智能硬件](#) [Docker](#) [OpenStack](#)
[VPN](#) [Spark](#) [ERP](#) [IE10](#) [Eclipse](#) [CRM](#) [JavaScript](#) [数据库](#) [Ubuntu](#) [NFC](#) [WAP](#) [jQuery](#)
[BI](#) [HTML5](#) [Spring](#) [Apache](#) [.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](#) [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](#) | [合作伙伴](#) | [论坛反馈](#)

[网站客服](#) [杂志客服](#) [微博客服](#) [webmaster@csdn.net](#) 400-600-2320 | 北京创新乐知信息技术有限公司 版权所有 | 江苏乐知网络技术有限公司 提供商务支持

京 ICP 证 09002463 号 | Copyright © 1999-2014, CSDN.NET, All Rights Reserved 