

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：1077188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声卡驱动之一 (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声卡驱动之四

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

Linux ALSA声卡驱动之七：ASoC架构中的Codec

标签：codec linux struct playback audio stream

2012-02-23 14:12 36156人阅读 评论(17) 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(?) [+]

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 */
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 */
```

Linux时间子系统之六: 高 (37505)
Android Audio System 之 (36411)
Linux ALSA声卡驱动之十 (36317)
Linux ALSA声卡驱动之四 (36146)
Linux ALSA声卡驱动之四 (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: 高 (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 so zhqh100: 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: 很喜欢博主的文章, 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: Al ss0429: 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...

```
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. };
```

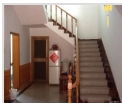
snd_soc_dai:

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

snd_soc_dai_driver:



pos机刷卡手续



个人急售二手房



上海专升本



呼叫中心系统

```
[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。

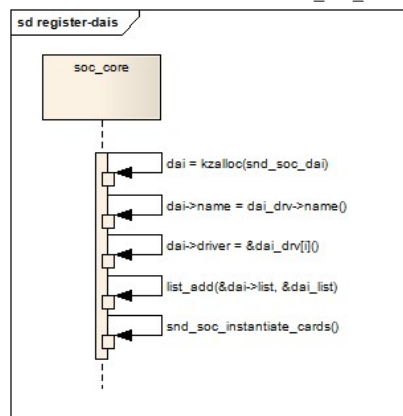


图3.1 dai的注册

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：

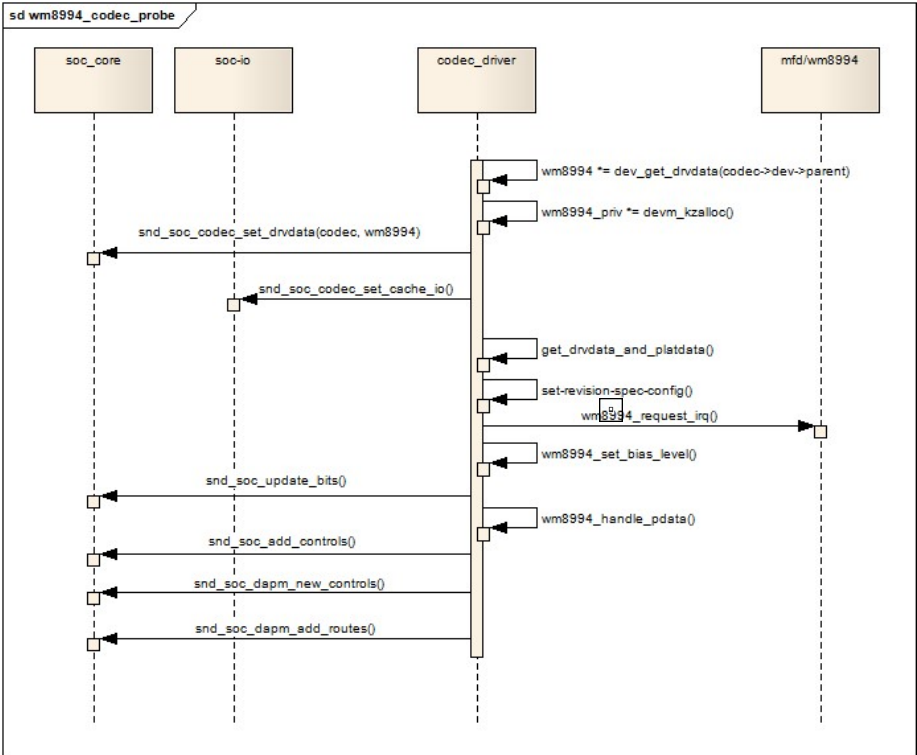


图5.1 wm8994_codec_probe

- 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。

顶

2

踩

0

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

我的同类文章

Linux音频子系统（14）Linux设备驱动（19）

• 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声卡驱动之六：... 2012-02-03 阅读 37515

• 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



大型网站架构知识库

2045 关注 | 532 收录


猜你在找

- 嵌入式Linux项目实战：三个大项目（数码相机、摄像头驱动Linux ALSA声卡驱动之七ASoC架构中的Codec
- Android底层技术：Linux驱动框架与开发
- Linux设备驱动开发入门
- linux嵌入式开发+驱动开发
- CentOS7 Linux系统管理实战视频课程
- Linux ALSA声卡驱动之七ASoC架构中的Codec
- Linux ALSA声卡驱动之七ASoC架构中的Codec
- Linux ALSA声卡驱动之七ASoC架构中的Codec
- Linux ALSA声卡驱动之七ASoC架构中的Codec
- Linux ALSA声卡驱动之七ASoC架构中的Codec




查看评论

10楼 [Bard_Zhang](#) 2016-03-03 16:59发表



博主您好，我现在需要解决苹果耳机麦克风不能使用的问题。
我的设备也是美标的耳机口，市面上常见的魅族，小米耳机都能录音，但是苹果耳机不能录音。请问有没有思路。

9楼 [sw8sw8](#) 2015-11-12 23:23发表



请问最终的读写DMA的函数是哪个。这个函数是自己需要写的吗。如果可以自己写。能不能在里面加上自己的音频处理代码。

8楼 [OSLVRO](#) 2014-08-07 16:13发表



good

7楼 [瑞瑞瑞](#) 2014-07-25 15:13发表



大神，我有这么个感觉，真正与平台无关部分，貌似，调试的时候，我们不需要动。
真正需要动的，只是与平台有关部分，比如i2c通道的选择和地址的设置？还有i2s管脚功能的设置？

Re: [DroidPhone](#) 2014-07-25 17:11发表



回复chenmeiceng：是的，了解与平台无关部分的原理，对调试和分析问题有较大帮助。

6楼 [Color_Fly](#) 2014-07-03 21:04发表



版主，请教个问题。这个ragmap机制，寄存器的操作大都是采用缓存的机制来处理的，内核什么时候才将真正的值回写到寄存器的，是什么触发了这个动作

5楼 [恒赌东道](#) 2014-06-28 16:18发表



楼主你好，这几天不断的学习，参考您的博客，受益匪浅，不过也会遇到一些问题，打扰您了，在文章的snd_soc_write这些函数我跟了一下，发现trace_snd_soc_reg_write这个没有定义的函数，那么就应该是codec->write实现了寄存器的读操作吧？codec的结构体为snd_soc_codec，我搜索源码，没有找到对应的write函数，有点迷惑，希望您讲解一下，非常感谢！

Re: [Color_Fly](#) 2014-07-03 21:02发表



回复WOAICJIA：这个貌似是内核调试用的，类似于打个桩，个人意见。

Re: [恒赌东道](#) 2014-07-04 11:52发表



回复u012498713：恩呢，非常感谢，通过差资料确实是内核调试用的，我走错分支了，在此阅读了你的博文，差不多理解了！

Re: [Color_Fly](#) 2014-07-04 19:21发表



回复WOAICJIA：你这次是真搞错了，你读的是博主的博文

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 [31843264群主](#) 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

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
BI	HTML5	Spring	Apache	.NET	API	HTML	SDK	IIS	Fedora	XML
Splashtop	UML	components	Windows Mobile	Rails	QEMU	KDE	Cassandra	CloudStack	LBS	Unity
coremail	OPhone	CouchBase	云计算	iOS6	Rackspace	Web App	SpringSide	Maemo	FTC	

[Compuware](#) [大数据](#) [apttech](#) [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 