个人资料

DroidPhone

访问： 1077191次
积分： 8768
等级： BLOG 6
排名： 第1448名

原创： 51篇 转载： 0篇
译文： 4篇 评论： 537条

Linux ALSA声卡驱动之八：ASoC架构中的Platform

标签： linux buffer codec playback struct stream

2012-03-13 14:56 31317人阅读 评论(35) 收藏 举报

分类： Linux设备驱动（19） Linux音频子系统（14）

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或其他操作传送至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。

/**/

/**/

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
static struct snd_soc_platform_driver samsung_asoc_platform = {
    .ops        = &dma_ops,
    .pcm_new    = dma_new,
    .pcm_free   = dma_free_dma_buffers,
};

static int __devinit samsung_asoc_platform_probe(struct platform_device *pdev)
{
    return snd_soc_register_platform(&pdev->dev, &samsung_asoc_platform);
}

static int __devexit samsung_asoc_platform_remove(struct platform_device *pdev)
{
    snd_soc_unregister_platform(&pdev->dev);
    return 0;
}

static struct platform_driver asoc_dma_driver = {
    .driver = {
        .name = "samsung-audio",
```

Linux时间子系统之六：高 (37505)
Android Audio System 之 (36411)
Linux ALSA声卡驱动之七 (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声卡驱动之八 (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声卡驱动之七 (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 sou
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
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：PCM
灿哥哥: 学习了
Android Audio System 之三： A
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()放在中断上半部有会导致死锁。2. 如果...

```
        .owner = THIS_MODULE,
    },

    .probe = samsung_asoc_platform_probe,
    .remove = __devexit_p(samsung_asoc_platform_remove),
};

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；
- 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
#define SND_SOC_DAIFMT_I2S      1 /* I2S mode */
#define SND_SOC_DAIFMT_RIGHT_J      2 /* Right Justified mode */
#define SND_SOC_DAIFMT_LEFT_J       3 /* Left Justified mode */
#define SND_SOC_DAIFMT_DSP_A        4 /* L data MSB after FRM LRC */
#define SND_SOC_DAIFMT_DSP_B        5 /* L data MSB during FRM LRC */
#define SND_SOC_DAIFMT_AC97     6 /* AC97 */
#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
#define SND_SOC_DAIFMT_CONT     (1 << 4) /* continuous clock */
#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
#define SND_SOC_DAIFMT_NB_NF        (1 << 8) /* normal bit clock + frame */
#define SND_SOC_DAIFMT_NB_IF        (2 << 8) /* normal BCLK + inv FRM */
#define SND_SOC_DAIFMT_IB_NF        (3 << 8) /* invert BCLK + nor FRM */
#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
#define SND_SOC_DAIFMT_CBM_CFM      (1 << 12) /* codec clk & FRM master */
#define SND_SOC_DAIFMT_CBS_CFM      (2 << 12) /* codec clk slave & FRM master */
#define SND_SOC_DAIFMT_CBM_CFS      (3 << 12) /* codec clk master & frame slave */
#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
struct snd_dma_buffer {
    struct snd_dma_device dev;  /* device type */
    unsigned char *area;    /* virtual pointer */
    dma_addr_t addr;    /* physical address */
    size_t bytes;       /* buffer size in bytes */
    void *private_data; /* private for allocator; don't touch */
};
```

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
static struct snd_soc_platform_driver samsung_asoc_platform = {
    .ops        = &dma_ops,
    .pcm_new    = dma_new,
    .pcm_free   = dma_free_dma_buffers,
};

static int __devinit samsung_asoc_platform_probe(struct platform_device *pdev)
{
    return snd_soc_register_platform(&pdev->dev, &samsung_asoc_platform);
}
```

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
static int preallocate_dma_buffer(struct snd_pcm *pcm, int stream)
{
    struct snd_pcm_substream *substream = pcm->streams[stream].substream;
    struct snd_dma_buffer *buf = &substream->dma_buffer;
    size_t size = dma_hardware.buffer_bytes_max;

    pr_debug("Entered %s\n", __func__);

    buf->dev.type = SNDRV_DMA_TYPE_DEV;
    buf->dev.dev = pcm->card->dev;
    buf->private_data = NULL;
    buf->area = dma_alloc_writecombine(pcm->card->dev, size,
                       &buf->addr, GFP_KERNEL);
    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_weitecombine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。

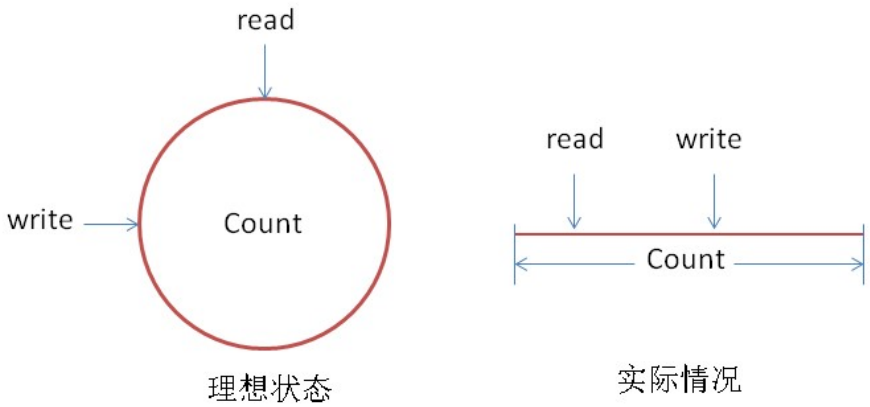

图6.2.1 环形缓冲区

环形缓冲区正好适合用于这种情景的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：

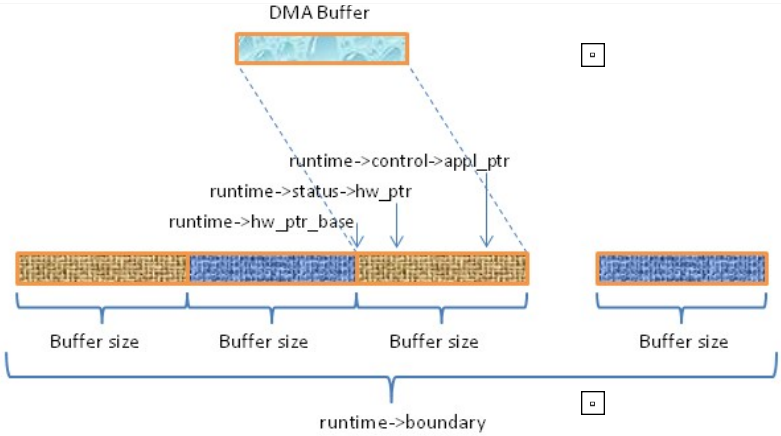

图6.2.2 alsa driver缓冲区管理

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)*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
static inline snd_pcm_uframes_t snd_pcm_playback_avail(struct snd_pcm_runtime *runtime)
{
    snd_pcm_sframes_t avail = runtime->status->hw_ptr + runtime->buffer_size - runtime->control->a
    if (avail < 0)
        avail += runtime->boundary;
    else if ((snd_pcm_uframes_t) avail >= runtime->boundary)
        avail -= runtime->boundary;
    return avail;
}
```

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
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
<span style="font-family:Arial, Verdana, sans-serif;"><span style="white-space: normal;"></span></span><pre class="cpp" name="code">void snd_pcm_period_elapsed(struct snd_pcm_substream *substream)
{
    struct snd_pcm_runtime *runtime;
```

```
    unsigned long flags;

    if (PCM_RUNTIME_CHECK(substream))
        return;
    runtime = substream->runtime;

    if (runtime->transfer_ack_begin)
        runtime->transfer_ack_begin(substream);

    snd_pcm_stream_lock_irqsave(substream, flags);
    if (!snd_pcm_running(substream) ||
        snd_pcm_update_hw_ptr0(substream, 1) < 0)
        goto _end;

    if (substream->timer_running)
        snd_timer_interrupt(substream->timer, 1);
 _end:
    snd_pcm_stream_unlock_irqrestore(substream, flags);
    if (runtime->transfer_ack_end)
        runtime->transfer_ack_end(substream);
    kill_fasync(&runtime->fasync, SIGIO, POLL_IN);
}
</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>
<br>
<pre></pre>
<pre></pre>
<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:

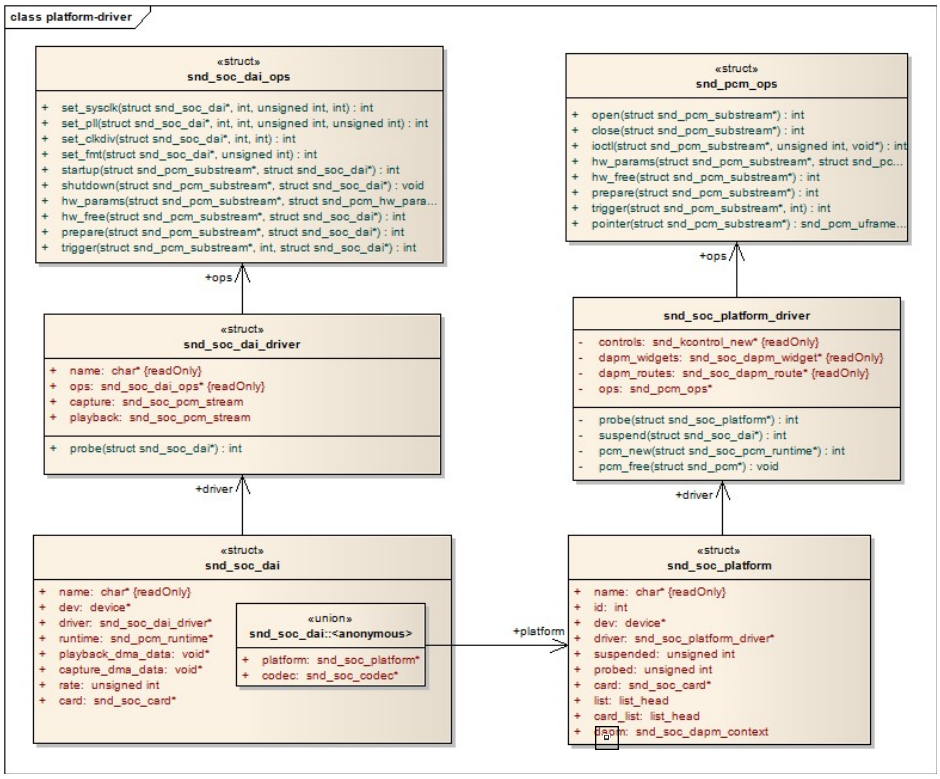

图7.1 ASoC Platform驱动

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：

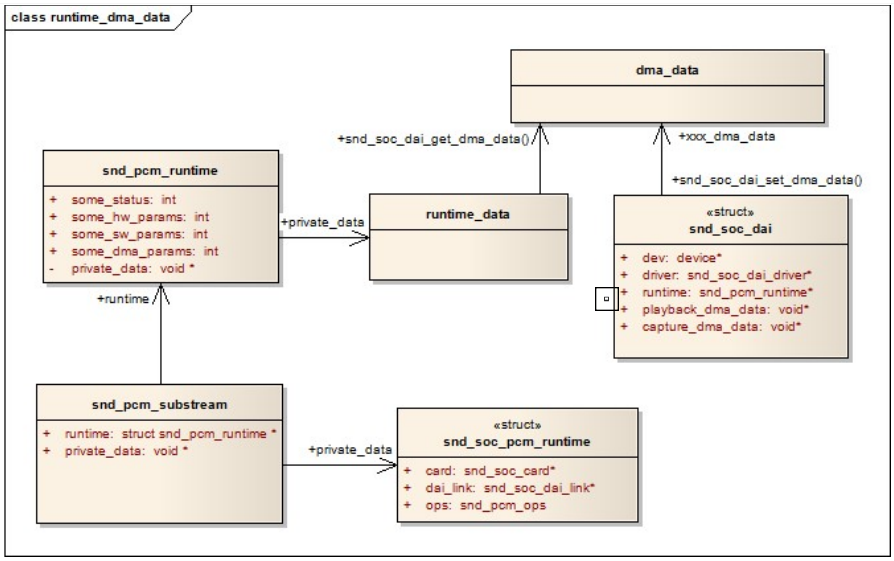

图7.2 private_data

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

我的同类文章

Linux设备驱动（19） Linux音频子系统（14）

- Linux SPI总线和设备驱动架… 2014-04-28 阅读 7370
- Linux SPI总线和设备驱动架… 2014-04-18 阅读 7936
- Linux中断（interrupt）子系… 2012-05-01 阅读 20964
- Linux中断（interrupt）子系… 2012-04-24 阅读 13768
- Linux中断（interrupt）子系… 2012-04-12 阅读 30031
- Linux SPI总线和设备驱动架… 2014-04-23 阅读 7352
- Linux SPI总线和设备驱动架… 2014-04-12 阅读 10804
- Linux中断（interrupt）子系… 2012-04-27 阅读 14327
- Linux中断（interrupt）子系… 2012-04-18 阅读 14472
- 自旋锁spin_lock和raw_spin… 2012-03-26 阅读 11876

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(数码相框、摄像头驱
Android底层技术：Linux驱动框架与开发
如何打造移动环境下满足业务场景的高可用架构
Linux设备驱动开发入门
linux嵌入式开发+驱动开发

Linux ALSA声卡驱动之八ASoC架构中的Platform
Linux ALSA声卡驱动之八ASoC架构中的Platform
Linux ALSA声卡驱动之八ASoC架构中的Platform
Linux ALSA声卡驱动之八ASoC架构中的Platform
Linux ALSA声卡驱动之八ASoC架构中的Platform

查看评论

24楼 No威_ 2015-07-22 17:20发表

好文章，牛

23楼 sinat_19703487 2014-10-23 15:21发表

博主，你好，我想问一下。platform_driver 怎么匹配设备的。

22楼 c553110519 2014-10-16 15:53发表

我现在在海思的芯片上集成这个ALSA，因为海思芯片内部有个声卡模块，不需要外界芯片，我想问下，这样的情况下，还可以移植ALSA吗，如何移植呢？

Re: jshmchenxi 2015-07-13 11:29发表

回复c553110519：楼上你好，我最近也在研究在海思的cpu上移植alsa，不知你的进展如何，留个联系方式交流一下？

Re: jshmchenxi 2015-07-13 11:28发表

回复c553110519：楼上你好，我最近也在研究在海思的cpu上移植alsa，不知你的进展如何，留个联系方式交流一下？

21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！

20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, xrreg, 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, .rreg = xrreg, \
.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？

19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？

18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，而不经过DAC送到内置的codec里？？

17楼 lostman80 2013-07-04 22:22发表

膜拜啊

16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

15楼 醉酒当歌人生几何 2013-06-06 10:33发表

打印出来读上几遍，谢谢lz！

14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，不知道是哪里出出问题？或者说是出现这样的问题有哪些可能的原因？

13楼 hellowxwworld 2012-12-12 11:09发表

分析的alsa框架非常清晰.

12楼 confidence321 2012-10-11 17:18发表

hao,讲的很高深。谢谢分享

11楼 tuituji_tuituji 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！