个人资料

DroidPhone

访问： 1077197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

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声卡驱动之六

ALSA声卡驱动中的DAPM详解之三：如何定义各种widget

标签： linux dapm widget audio driver

2013-10-24 21:01 12568人阅读 评论(10) 收藏 举报

分类： Linux音频子系统（14）

目录(?) [+]

上一节中，介绍了DAPM框架中几个重要的**数据结构**：snd_soc_dapm_widget，snd_soc_dapm_path，snd_soc_dapm_route。其中snd_soc_dapm_path无需我们自己定义，它会在注册snd_soc_dapm_route时动态地生成，但是对于系统中的widget和route，我们是需要自己进行定义的，另外，widget所包含的kcontrol与普通的kcontrol有所不同，它们的定义方法与标准的kcontrol也有所不同。本节的内容我将会介绍如何使用DAPM系统提供的一些辅助宏定义来定义各种类型的widget和它所用到的kcontrol。

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

定义widget

和普通的kcontrol一样，DAPM框架为我们提供了大量的辅助宏用来定义各种各样的widget控件，这些宏定义根据widget的类型，按照它们的电源所在的域，被分为了几个域，他们分别是：

- **codec域** 比如VREF和VMID等提供参考电压的widget，这些widget通常在codec的probe/remove回调中进行控制，当然，在工作中如果没有音频流时，也可以适当地进行控制它们的开启与关闭。
- **platform域** 位于该域上的widget通常是针对平台或板子的一些需要物理连接的输入/输出接口，例如耳机、扬声器、麦克风，因为这些接口在每块板子上都可能不一样，所以通常它们是在machine驱动中进行定义和控制，并且也可以由用户空间的应用程序通过某种方式来控制它们的打开和关闭。
- **音频路径域** 一般是指codec内部的mixer、mux等控制音频路径的widget，这些widget可以根据用户空间的设定连接关系，自动设定他们的电源状态。
- **音频数据流域** 是指那些需要处理音频数据流的widget，例如ADC、DAC等等。

codec域widget的定义

目前，DAPM框架只提供了定义一个codec域widget的辅助宏：

```cpp
#define SND_SOC_DAPM_VMID(wname) \
{       .id = snd_soc_dapm_vmid, .name = wname, .kcontrol_news = NULL, \
        .num_kcontrols = 0}
```

platform域widget的定义

DAPM框架为我们提供了多种platform域widget的辅助定义宏：

```cpp
#define SND_SOC_DAPM_SIGGEN(wname) \
{       .id = snd_soc_dapm_siggen, .name = wname, .kcontrol_news = NULL, \
```

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...

```
03.         .num_kcontrols = 0, .reg = SND_SOC_NOPM }
04. #define SND_SOC_DAPM_INPUT(wname) \
05. {       .id = snd_soc_dapm_input, .name = wname, .kcontrol_news = NULL, \
06.         .num_kcontrols = 0, .reg = SND_SOC_NOPM }
07. #define SND_SOC_DAPM_OUTPUT(wname) \
08. {       .id = snd_soc_dapm_output, .name = wname, .kcontrol_news = NULL, \
09.         .num_kcontrols = 0, .reg = SND_SOC_NOPM }
10. #define SND_SOC_DAPM_MIC(wname, wevent) \
11. {       .id = snd_soc_dapm_mic, .name = wname, .kcontrol_news = NULL, \
12.         .num_kcontrols = 0, .reg = SND_SOC_NOPM, .event = wevent, \
13.         .event_flags = SND_SOC_DAPM_PRE_PMU | SND_SOC_DAPM_POST_PMD}
14. #define SND_SOC_DAPM_HP(wname, wevent) \
15. {       .id = snd_soc_dapm_hp, .name = wname, .kcontrol_news = NULL, \
16.         .num_kcontrols = 0, .reg = SND_SOC_NOPM, .event = wevent, \
17.         .event_flags = SND_SOC_DAPM_POST_PMU | SND_SOC_DAPM_PRE_PMD}
18. #define SND_SOC_DAPM_SPK(wname, wevent) \
19. {       .id = snd_soc_dapm_spk, .name = wname, .kcontrol_news = NULL, \
20.         .num_kcontrols = 0, .reg = SND_SOC_NOPM, .event = wevent, \
21.         .event_flags = SND_SOC_DAPM_POST_PMU | SND_SOC_DAPM_PRE_PMD}
22. #define SND_SOC_DAPM_LINE(wname, wevent) \
23. {       .id = snd_soc_dapm_line, .name = wname, .kcontrol_news = NULL, \
24.         .num_kcontrols = 0, .reg = SND_SOC_NOPM, .event = wevent, \
25.         .event_flags = SND_SOC_DAPM_POST_PMU | SND_SOC_DAPM_PRE_PMD}
```

以上这些widget分别对应信号发生器，输入引脚，输出引脚，麦克风，耳机，扬声器，线路输入接口。其中的reg字段被设置为SND_SOC_NOPM（-1），表明这些widget是没有寄存器控制位来控制widget的电源状态的。麦克风，耳机，扬声器，线路输入接口这几种widget，还可以定义一个dapm事件回调函数wevent，从event_flags字段的设置可以看出，他们只会响应SND_SOC_DAPM_POST_PMU（上电后）和SND_SOC_DAPM_PMD（下电前）事件，这几个widget通常会在machine驱动中定义，而SND_SOC_DAPM_INPUT和SND_SOC_DAPM_OUTPUT则用来定义codec芯片的输出输入脚，通常在codec驱动中定义，最后，在machine驱动中增加相应的route，把麦克风和耳机等widget与相应的codec输入输出引脚的widget连接起来。

音频路径（path）域widget的定义

这种widget通常是对普通kcontrols控件的再封装，增加音频路径和电源管理功能，所以这种widget会包含一个或多个kcontrol，普通kcontrol的定义方法我们在ALSA声卡驱动中的DAPM详解之一：kcontrol中已经介绍过，不过这些被包含的kcontrol不能使用这种方法定义，它们需要使用dapm框架提供的定义宏来定义，详细的讨论我们后面有介绍。这里先列出这些widget的定义宏：

```cpp
[cpp]
#define SND_SOC_DAPM_PGA(wname, wreg, wshift, winvert,\
         wcontrols, wncontrols) \
{       .id = snd_soc_dapm_pga, .name = wname, .reg = wreg, .shift = wshift, \
        .invert = winvert, .kcontrol_news = wcontrols, .num_kcontrols = wncontrols}
#define SND_SOC_DAPM_OUT_DRV(wname, wreg, wshift, winvert,\
         wcontrols, wncontrols) \
{       .id = snd_soc_dapm_out_drv, .name = wname, .reg = wreg, .shift = wshift, \
        .invert = winvert, .kcontrol_news = wcontrols, .num_kcontrols = wncontrols}
#define SND_SOC_DAPM_MIXER(wname, wreg, wshift, winvert, \
         wcontrols, wncontrols)\
{       .id = snd_soc_dapm_mixer, .name = wname, .reg = wreg, .shift = wshift, \
        .invert = winvert, .kcontrol_news = wcontrols, .num_kcontrols = wncontrols}
#define SND_SOC_DAPM_MIXER_NAMED_CTL(wname, wreg, wshift, winvert, \
         wcontrols, wncontrols)\
{       .id = snd_soc_dapm_mixer_named_ctl, .name = wname, .reg = wreg, \
        .shift = wshift, .invert = winvert, .kcontrol_news = wcontrols, \
        .num_kcontrols = wncontrols}
#define SND_SOC_DAPM_MICBIAS(wname, wreg, wshift, winvert) \
{       .id = snd_soc_dapm_micbias, .name = wname, .reg = wreg, .shift = wshift, \
        .invert = winvert, .kcontrol_news = NULL, .num_kcontrols = 0}
#define SND_SOC_DAPM_SWITCH(wname, wreg, wshift, winvert, wcontrols) \
{       .id = snd_soc_dapm_switch, .name = wname, .reg = wreg, .shift = wshift, \
        .invert = winvert, .kcontrol_news = wcontrols, .num_kcontrols = 1}
#define SND_SOC_DAPM_MUX(wname, wreg, wshift, winvert, wcontrols) \
{       .id = snd_soc_dapm_mux, .name = wname, .reg = wreg, .shift = wshift, \
        .invert = winvert, .kcontrol_news = wcontrols, .num_kcontrols = 1}
#define SND_SOC_DAPM_VIRT_MUX(wname, wreg, wshift, winvert, wcontrols) \
{       .id = snd_soc_dapm_virt_mux, .name = wname, .reg = wreg, .shift = wshift, \
        .invert = winvert, .kcontrol_news = wcontrols, .num_kcontrols = 1}
#define SND_SOC_DAPM_VALUE_MUX(wname, wreg, wshift, winvert, wcontrols) \
{       .id = snd_soc_dapm_value_mux, .name = wname, .reg = wreg, \
        .shift = wshift, .invert = winvert, .kcontrol_news = wcontrols, \
        .num_kcontrols = 1}
```

可以看出，这些widget的reg和shift字段是需要赋值的，说明这些widget是有相应的电源控制寄存器的，DAPM框架在扫描和更新音频路径时，会利用这些寄存器来控制widget的电源状态，使得它们的供电状态是按需分配的，需要的时候（在有效的音频路径上）上电，不需要的时候（不再有效的音频路径上）下电。这些widget需要完成和之前介绍的mixer、mux等控件同样的功能，实际上，这是通过它们包含的kcontrol控件来完成的，这些kcontrol我们需要在定义widget前先定义好，然后通过wcontrols和num_kcontrols参数传递给这些辅助定义宏。
如果需要自定义这些widget的dapm事件处理回调函数，也可以使用下面这些带“_E”后缀的版本：

- SND_SOC_DAPM_PGA_E
- SND_SOC_DAPM_OUT_DRV_E
- SND_SOC_DAPM_MIXER_E
- SND_SOC_DAPM_MIXER_NAMED_CTL_E
- SND_SOC_DAPM_SWITCH_E
- SND_SOC_DAPM_MUX_E
- SND_SOC_DAPM_VIRT_MUX_E

音频数据流（stream）域widget的定义

这些widget主要包含音频输入/输出接口，ADC/DAC等等：

```cpp
#define SND_SOC_DAPM_AIF_IN(wname, stname, wslot, wreg, wshift, winvert) \
{       .id = snd_soc_dapm_aif_in, .name = wname, .sname = stname, \
        .reg = wreg, .shift = wshift, .invert = winvert }
#define SND_SOC_DAPM_AIF_IN_E(wname, stname, wslot, wreg, wshift, winvert, \
                              wevent, wflags)                   \
{       .id = snd_soc_dapm_aif_in, .name = wname, .sname = stname, \
        .reg = wreg, .shift = wshift, .invert = winvert, \
        .event = wevent, .event_flags = wflags }
#define SND_SOC_DAPM_AIF_OUT(wname, stname, wslot, wreg, wshift, winvert) \
{       .id = snd_soc_dapm_aif_out, .name = wname, .sname = stname, \
        .reg = wreg, .shift = wshift, .invert = winvert }
#define SND_SOC_DAPM_AIF_OUT_E(wname, stname, wslot, wreg, wshift, winvert, \
                               wevent, wflags)                  \
{       .id = snd_soc_dapm_aif_out, .name = wname, .sname = stname, \
        .reg = wreg, .shift = wshift, .invert = winvert, \
        .event = wevent, .event_flags = wflags }
#define SND_SOC_DAPM_DAC(wname, stname, wreg, wshift, winvert) \
{       .id = snd_soc_dapm_dac, .name = wname, .sname = stname, .reg = wreg, \
        .shift = wshift, .invert = winvert}
#define SND_SOC_DAPM_DAC_E(wname, stname, wreg, wshift, winvert, \
                           wevent, wflags)                  \
{       .id = snd_soc_dapm_dac, .name = wname, .sname = stname, .reg = wreg, \
        .shift = wshift, .invert = winvert, \
        .event = wevent, .event_flags = wflags}
#define SND_SOC_DAPM_ADC(wname, stname, wreg, wshift, winvert) \
{       .id = snd_soc_dapm_adc, .name = wname, .sname = stname, .reg = wreg, \
        .shift = wshift, .invert = winvert}
#define SND_SOC_DAPM_ADC_E(wname, stname, wreg, wshift, winvert, \
                           wevent, wflags)                  \
{       .id = snd_soc_dapm_adc, .name = wname, .sname = stname, .reg = wreg, \
        .shift = wshift, .invert = winvert, \
        .event = wevent, .event_flags = wflags}
#define SND_SOC_DAPM_CLOCK_SUPPLY(wname) \
{       .id = snd_soc_dapm_clock_supply, .name = wname, \
        .reg = SND_SOC_NOPM, .event = dapm_clock_event, \
        .event_flags = SND_SOC_DAPM_PRE_PMU | SND_SOC_DAPM_POST_PMD }
```

除了上面这些widget，还有另外三种widget没有提供显示的定义方法，它们的种类id分别是：

- snd_soc_dapm_dai_in
-
- snd_soc_dapm_dai_out
- snd_soc_dapm_dai_link

还记得我们在Linux ALSA声卡驱动之七：ASoC架构中的Codec中的snd_soc_dai结构吗？每个codec有多个dai，而cpu（通常就是指某个soc cpu芯片）也会有多个dai，dai注册时，dapm系统会为每个dai创建一个snd_soc_dapm_dai_in或snd_soc_dapm_dai_out类型的widget，通常，这两种widget会和codec中具有相同的stream name的widget进行连接。另外一种情况，当系统中具有多个音频处理器（比如多个codec）时，他们之间可能会通过某两个dai进行连接，当machine驱动确认有这种配置时（通过判断dai_links结构中的param字段），会为

他们建立一个dai link把他们绑定在一起，因为有连接关系，两个音频处理器之间的widget的电源状态就可以互相传递。

除了还有几个通用的widget，他们的定义方法如下：

```cpp
#define SND_SOC_DAPM_REG(wid, wname, wreg, wshift, wmask, won_val, woff_val) \
{       .id = wid, .name = wname, .kcontrol_news = NULL, .num_kcontrols = 0, \
        .reg = -((wreg) + 1), .shift = wshift, .mask = wmask, \
        .on_val = won_val, .off_val = woff_val, .event = dapm_reg_event, \
        .event_flags = SND_SOC_DAPM_PRE_PMU | SND_SOC_DAPM_POST_PMD}
#define SND_SOC_DAPM_SUPPLY(wname, wreg, wshift, winvert, wevent, wflags) \
{       .id = snd_soc_dapm_supply, .name = wname, .reg = wreg,  \
        .shift = wshift, .invert = winvert, .event = wevent, \
        .event_flags = wflags}
#define SND_SOC_DAPM_REGULATOR_SUPPLY(wname, wdelay, wflags)        \
{       .id = snd_soc_dapm_regulator_supply, .name = wname, \
        .reg = SND_SOC_NOPM, .shift = wdelay, .event = dapm_regulator_event, \
        .event_flags = SND_SOC_DAPM_PRE_PMU | SND_SOC_DAPM_POST_PMD, \
        .invert = wflags}
```

定义dapm kcontrol

上一节提到，对于音频路径上的mixer或mux类型的widget，它们包含了若干个kcontrol，这些被包含的kcontrol实际上就是我们之前讨论的mixer和mux等，dapm利用这些kcontrol完成音频路径的控制。不过，对于widget来说，它的任务还不止这些，dapm还要动态地管理这些音频路径的连结关系，以便可以根据这些连接关系来控制这些widget的电源状态，如果按照普通的方法定义这些kcontrol，是无法达到这个目的的，因此，dapm为我们提供了另外一套定义宏，由它们完成这些被widget包含的kcontrol的定义。

```cpp
#define SOC_DAPM_SINGLE(xname, reg, shift, max, invert) \
{       .iface = SNDRV_CTL_ELEM_IFACE_MIXER, .name = xname, \
        .info = snd_soc_info_volsw, \
        .get = snd_soc_dapm_get_volsw, .put = snd_soc_dapm_put_volsw, \
        .private_value =  SOC_SINGLE_VALUE(reg, shift, max, invert) }
#define SOC_DAPM_SINGLE_TLV(xname, reg, shift, max, invert, tlv_array) \
{       .iface = SNDRV_CTL_ELEM_IFACE_MIXER, .name = xname, \
        .info = snd_soc_info_volsw, \
        .access = SNDRV_CTL_ELEM_ACCESS_TLV_READ | SNDRV_CTL_ELEM_ACCESS_READWRITE,\
        .tlv.p = (tlv_array), \
        .get = snd_soc_dapm_get_volsw, .put = snd_soc_dapm_put_volsw, \
        .private_value =  SOC_SINGLE_VALUE(reg, shift, max, invert) }
#define SOC_DAPM_ENUM(xname, xenum) \
{       .iface = SNDRV_CTL_ELEM_IFACE_MIXER, .name = xname, \
        .info = snd_soc_info_enum_double, \
        .get = snd_soc_dapm_get_enum_double, \
        .put = snd_soc_dapm_put_enum_double, \
        .private_value = (unsigned long)&xenum }
#define SOC_DAPM_ENUM_VIRT(xname, xenum)                    \
{       .iface = SNDRV_CTL_ELEM_IFACE_MIXER, .name = xname, \
        .info = snd_soc_info_enum_double, \
        .get = snd_soc_dapm_get_enum_virt, \
        .put = snd_soc_dapm_put_enum_virt, \
        .private_value = (unsigned long)&xenum }
#define SOC_DAPM_ENUM_EXT(xname, xenum, xget, xput) \
{       .iface = SNDRV_CTL_ELEM_IFACE_MIXER, .name = xname, \
        .info = snd_soc_info_enum_double, \
        .get = xget, \
        .put = xput, \
        .private_value = (unsigned long)&xenum }
#define SOC_DAPM_VALUE_ENUM(xname, xenum) \
{       .iface = SNDRV_CTL_ELEM_IFACE_MIXER, .name = xname, \
        .info = snd_soc_info_enum_double, \
        .get = snd_soc_dapm_get_value_enum_double, \
        .put = snd_soc_dapm_put_value_enum_double, \
        .private_value = (unsigned long)&xenum }
#define SOC_DAPM_PIN_SWITCH(xname) \
{       .iface = SNDRV_CTL_ELEM_IFACE_MIXER, .name = xname " Switch", \
        .info = snd_soc_dapm_info_pin_switch, \
        .get = snd_soc_dapm_get_pin_switch, \
        .put = snd_soc_dapm_put_pin_switch, \
        .private_value = (unsigned long)xname }
```

可以看出，SOC_DAPM_SINGLE对应与普通控件的SOC_SINGLE，SOC_DAPM_SINGLE_TLV对应SOC_SINGLE_TLV.....，相比普通的kcontrol控件，dapm的kcontrol控件只是把info，get，put回调函数换掉了。dapm kcontrol的put回调函数不仅仅会更新控件本身的状态，他还会把这种变化传递到相邻的dapm kcontrol，相邻的dapm kcontrol又会传递这个变化到他自己相邻的dapm kcontrol，知道音频路径的末端，通过这种机制，只要改变其中一个widget的连接状态，与之相关的所有widget都会被扫描并测试一下自身是否还在有效的音频路径中，从而可以动态地改变自身的电源状态，这就是dapm的精髓所在。这里我只提一下这种概念，后续的章节会有较为详细的代码分析过程。

建立widget和route

上面介绍了一大堆的辅助宏，那么，对于一个实际的系统，我们怎么定义我们需要的widget？怎样定义widget的连接关系？下面我们还是以Wolfson公司的codec芯片WM8993为例子来了解这个过程。

第一步，利用辅助宏定义widget所需要的dapm kcontrol：

```cpp
static const struct snd_kcontrol_new left_speaker_mixer[] = {
SOC_DAPM_SINGLE("Input Switch", WM8993_SPEAKER_MIXER, 7, 1, 0),
SOC_DAPM_SINGLE("IN1LP Switch", WM8993_SPEAKER_MIXER, 5, 1, 0),
SOC_DAPM_SINGLE("Output Switch", WM8993_SPEAKER_MIXER, 3, 1, 0),
SOC_DAPM_SINGLE("DAC Switch", WM8993_SPEAKER_MIXER, 6, 1, 0),
};

static const struct snd_kcontrol_new right_speaker_mixer[] = {
SOC_DAPM_SINGLE("Input Switch", WM8993_SPEAKER_MIXER, 6, 1, 0),
SOC_DAPM_SINGLE("IN1RP Switch", WM8993_SPEAKER_MIXER, 4, 1, 0),
SOC_DAPM_SINGLE("Output Switch", WM8993_SPEAKER_MIXER, 2, 1, 0),
SOC_DAPM_SINGLE("DAC Switch", WM8993_SPEAKER_MIXER, 0, 1, 0),
};

static const char *aif_text[] = {
        "Left", "Right"
};

static const struct soc_enum aifinl_enum =
        SOC_ENUM_SINGLE(WM8993_AUDIO_INTERFACE_2, 15, 2, aif_text);


static const struct snd_kcontrol_new aifinl_mux =
        SOC_DAPM_ENUM("AIFINL Mux", aifinl_enum);


static const struct soc_enum aifinr_enum =
        SOC_ENUM_SINGLE(WM8993_AUDIO_INTERFACE_2, 14, 2, aif_text);


static const struct snd_kcontrol_new aifinr_mux =
        SOC_DAPM_ENUM("AIFINR Mux", aifinr_enum);
```

以上，我们定义了wm8993中左右声道的speaker mixer控件：left_speaker_mixer和right_speaker_mixer，同时还为左右声道各定义了一个叫做AIFINL Mux和AIFINR Mux的输入选择mux控件。

第二步，定义真正的widget，包含第一步定义好的dapm控件：

```cpp
static const struct snd_soc_dapm_widget wm8993_dapm_widgets[] = {
    ......
    SND_SOC_DAPM_AIF_IN("AIFINL", "Playback", 0, SND_SOC_NOPM, 0, 0),
    SND_SOC_DAPM_AIF_IN("AIFINR", "Playback", 1, SND_SOC_NOPM, 0, 0),
    ......
    SND_SOC_DAPM_MUX("DACL Mux", SND_SOC_NOPM, 0, 0, &aifinl_mux),
    SND_SOC_DAPM_MUX("DACR Mux", SND_SOC_NOPM, 0, 0, &aifinr_mux),

    SND_SOC_DAPM_MIXER("SPKL", WM8993_POWER_MANAGEMENT_3, 8, 0,
                       left_speaker_mixer, ARRAY_SIZE(left_speaker_mixer)),
    SND_SOC_DAPM_MIXER("SPKR", WM8993_POWER_MANAGEMENT_3, 9, 0,
                       right_speaker_mixer, ARRAY_SIZE(right_speaker_mixer)),
    ......
```

```
};
```

这一步，为左右声道各自定义了一个mux widget：DACL Mux和DACR Mux，实际的多路选择由dapm kcontrol：aifinl_mux和aifinr_mux，来完成，因为传入了SND_SOC_NOPM参数，这两个widget不具备电源属性，但是mux的切换会影响到与之相连的其它具备电源属性的电源状态。我们还为左右声道的扬声器各自定义了一个mixer widget：SPKL和SPKR，具体的mixer控制由上一步定义的left_speaker_mixer和right_speaker_mixer来完成，两个widget具备电源属性，所以，当这两个widget在一条有效的音频路径上时，dapm框架可以通过寄存器WM8993_POWER_MANAGEMENT_3的第8位和第9位控制它的电源状态。

第三步，定义这些widget的连接路径：

```cpp
static const struct snd_soc_dapm_route routes[] = {
        ......

        { "DACL Mux", "Left", "AIFINL" },
        { "DACL Mux", "Right", "AIFINR" },
        { "DACR Mux", "Left", "AIFINL" },
        { "DACR Mux", "Right", "AIFINR" },

        ......

        { "SPKL", "DAC Switch", "DACL" },
        { "SPKL", NULL, "CLK_SYS" },

        { "SPKR", "DAC Switch", "DACR" },
        { "SPKR", NULL, "CLK_SYS" },
};
```

通过第一步的定义，我们知道DACL Mux和DACR Mux有两个输入引脚，分别是

- Left
- Right

而SPKL和SPKR有四个输入选择引脚，分别是：

- Input Switch
- IN1LP Switch/IN1RP Switch
- Output Switch
- DAC Switch

所以，很显然，上面的路径定义的意思就是：

- AIFINL连接到DACL Mux的Left输入脚
- AIFINR连接到DACL Mux的Right输入脚
- AIFINL连接到DACR Mux的Left输入脚
- AIFINR连接到DACR Mux的Right输入脚
- DACL连接到SPKL的DAC Switch输入脚
- DACR连接到SPKR的DAC Switch输入脚

第四步，在codec驱动的probe回调中注册这些widget和路径：

```cpp
static int wm8993_probe(struct snd_soc_codec *codec)
{
        ......
        snd_soc_dapm_new_controls(dapm, wm8993_dapm_widgets,
                                  ARRAY_SIZE(wm8993_dapm_widgets));
        ......

        snd_soc_dapm_add_routes(dapm, routes, ARRAY_SIZE(routes));
        ......
}
```

在machine驱动中，我们可以用同样的方式定义和注册板子特有的widget和路径信息。

顶 踩
0 0

上一篇 ALSA声卡驱动中的DAPM详解之二：widget-具备路径和电源管理信息的kcontrol
下一篇 ALSA声卡驱动中的DAPM详解之四：在驱动程序中初始化并注册widget和route

我的同类文章

Linux音频子系统（14）

- 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 9950
- ALSA声卡驱动中的DAPM详… 2013-11-04 阅读 12941
- ALSA声卡驱动中的DAPM详… 2013-11-01 阅读 10222
- ALSA声卡驱动中的DAPM详… 2013-10-23 阅读 10804
- ALSA声卡驱动中的DAPM详… 2013-10-18 阅读 17125
- Linux ALSA声卡驱动之八：… 2012-03-13 阅读 31315
- Linux ALSA声卡驱动之七：… 2012-02-23 阅读 36156
- Linux ALSA声卡驱动之六：… 2012-02-03 阅读 37515
- 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

分布式资源管理系统的前世今生，深入剖析YARN资源调度 ALSA声卡驱动中的DAPM详解之五建立widget之间的连接
【直通华为HCNA/HCNP系列R篇3】路由器接口配置与管理 ALSA声卡驱动中的DAPM详解之五建立widget之间的连接
基于.NET三层架构+高级SQL开发图书借阅管理系统 ALSA声卡驱动中的DAPM详解之五建立widget之间的连接
嵌入式Linux项目实战：三个大项目(数码相框、摄像头驱 ALSA声卡驱动中的DAPM详解之二widget-具备路径和电源
Struts实战-使用SSH框架技术开发学籍管理系统 ALSA声卡驱动中的DAPM详解之五建立widget之间的连接

查看评论

5楼 恒赌东道 2014-08-12 22:05发表

你好，在您的博客的帮组下，我学了很多，现在是在调试阶段了！请教您，alsamixer指令出来的图形界面，对应的控件名字和内核代码是如何联系起来的？非常感谢

4楼 elliepfsang 2013-12-18 15:41发表

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，也写到了dapm kcontrol与普通control的区别，但有个疑问是在定义普通control和dapm control时，如何区分哪些用普通control来定义，哪些用dapm control来定义，您能举个例子帮我解释一下吗？

Re: DroidPhone 2013-12-18 19:31发表

回复u012389631：和电源管理和音频路径相关的control需要定义为dapm control（比如电源或时钟开关，混音器，多路器等），否则定义为普通的control（例如：音量，音效，某些特殊的特性等）。

3楼 elliepfsang 2013-12-16 18:56发表

大侠
---
struct snd_soc_dapm_widget wm8993_dapm_widgets[] = {

.....
SND_SOC_DAPM_AIF_IN("AIFINL", "Playback", 0, SND_SOC_NOPM, 0, 0),
SND_SOC_DAPM_AIF_IN("AIFINR", "Playback", 1, SND_SOC_NOPM, 0, 0),
---
中的"AIFINL", "Playback"等字符串的设置是参考哪些而设的啊，关于这点满疑惑的

Re: DroidPhone 2013-12-16 21:04发表

回复u012389631：这里的playback，是指stream name，请参看：ALSA声卡驱动中的DAPM详解之七：dapm事件机制（dapm event）（http://blog.csdn.net/droidphone/article/details/14548631）：“连接dai widget和stream widget”一节的内容

2楼 hustljh 2013-12-11 20:16发表

kcontrol控件的get和put会调用snd_soc_codec_driver的read、write函数，wm8994.c中好像没有实现这两个函数，请教下博主，这是什么情况?是在其他地方实现了还是不需要？

Re: DroidPhone 2013-12-11 20:37发表

回复hustljh：wm8994_codec_probe-->snd_soc_codec_set_cache_io-->codec->write = hw_write;

1楼 Color_Fly 2013-11-01 10:41发表

版主你好，这里有个问题请教一下：
static const struct soc_enum aifinl_enum =
SOC_ENUM_SINGLE(WM8993_AUDIO_INTERFACE_2, 15, 2, aif_text);

对于这句：我查看了WM8993_AUDIO_INTERFACE_2寄存器的15位有两个选择（0 ： Left DAC outputs left interface data ，1 ： Left DAC outputs right interface data），这里为什么在宏中的xmax项却填写了2，我的理解是最大值是1，不知道这个地方怎么解释，请版主给科普下，thankyou！！！

Re: DroidPhone 2013-11-01 17:09发表

回复u012498713：我也注意到这点了，应该是内核的代码上的一个bug。

Re: Color_Fly 2013-11-01 19:47发表

回复DroidPhone：我查看了好几个，都是这种情况，不明白！！！！

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