ARM linux 平台 ALSA 驱动实现

作者： 向仔州

1.在应用层 open 的时候会调用 soc_pcm_open 指定驱动 mahcine，cpu_dai， codec_dai， platform(dma)文件里面 open 或者 startup 函数

2.在应用层 Ioctl(fd， SNDRV_PCM_IOCTL_HW_PARAMS)的时候会调用 snd_pcm_hw_params 函数来指定驱动 mahcine，cpu_dai，codec_dai， platform(dma)文件里面 hw_params 函数

在 alsa 应用程序调用过程文档里面讲了 alsa 调用底层的过程

在打开 open /dev/snd/pcmC0D0p 播放节点

snd_pcm_playback_open

static int snd_pcm_open

snd_pcm_open_file

int snd_pcm_open_substream

substream->ops->open

substream->ops->open

在 soc_pcm_open 函数里面就是如何去调用驱动层程序的文件

static int soc_pcm_open (struct snd_pcm_substream *substream)

{

if (cpu_dai->driver->ops && cpu_dai->driver->ops->startup) //cpu_dai 里面如果有 startup 函数就调用 startup 函数，否则不调用。 在 s3c24xx-i2s.c 没有发现 startup 函数，所以这个回调函数接口不调用

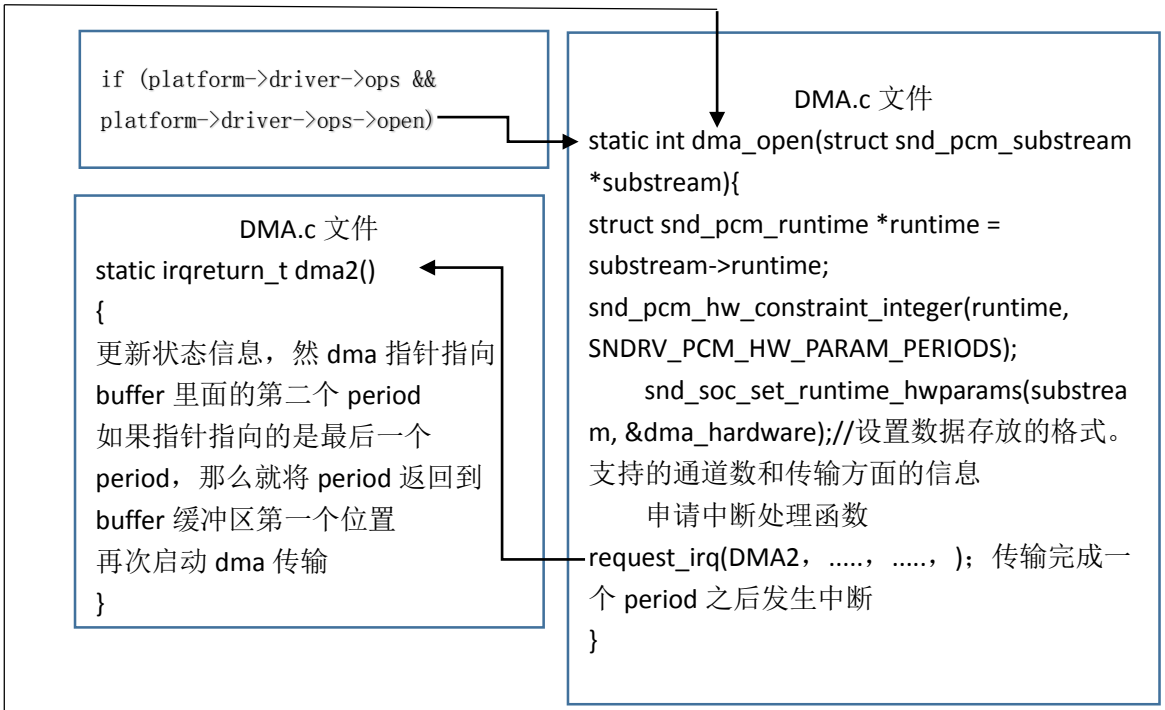
if (platform->driver->ops && platform->driver->ops->open) //DMA 里面如果有 open 函数就调用 open，否则不调用 在 DMA.c 文件里面有 open 函数，所以调用 open

if (codec_dai->driver->ops && codec_dai->driver->ops->startup) //codec_dai 里面如果有 startup 函数就调用 startup 函数，否则不调用

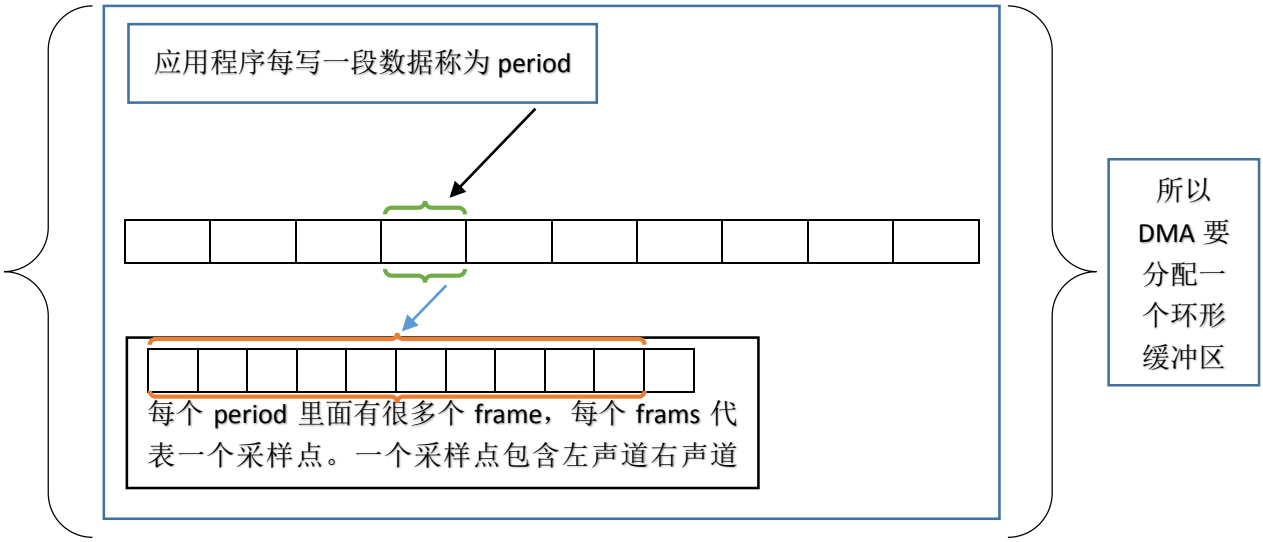
if (rtd->dai_link->ops && rtd->dai_link->ops->startup) //machine 文件里面如果有 startup 函数就调用 startup 函数，否则不调用。 在 mah cine 文件里面有 startup 函数，所以调用 startup

}

我们先来看看 open 函数去调用那些文件，我们发现 open 的时候只会去调用 dma 文件

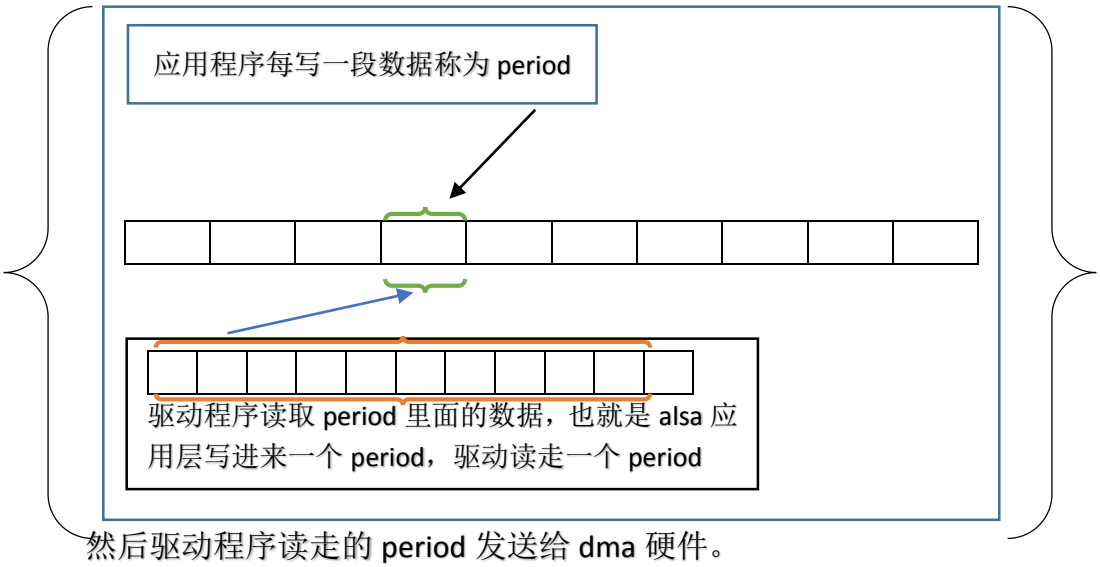


为什么要用缓冲区 buffer？



APP 调用 alsa 把一段数据写入这个 buffer，然后再写第二段，连续这样写

然后驱动程序也是同样的获取 buffer 里面的数据发送给 dma



然后驱动程序读走的 period 发送给 dma 硬件。

根据上述情况驱动程序就必须分配一个 buffer 给应用层和驱动层来相互读写

DMA.c 文件

```
static int dma_new(struct
snd_soc_pcm_runtime *rtd)
{
struct snd_pcm *pcm = rtd->pcm;
preallocate_dma_buffer(pcm,SNDRV_PCM_STR
EAM_PLAYBACK);//申请 DMAbuffer
}
```

DMA.c 文件

```
static void dma_free_dma_buffers(struct
snd_pcm *pcm)
{
dma_free_writecombine(pcm->card->dev,
buf->bytes,buf->area, buf->addr);//释放
DMAbuffer
}
```

DMA.c 文件

```
static struct snd_pcm_ops dma_ops = {
    .open           = dma_open,
    .close          = dma_close,
    .ioctl          = snd_pcm_lib_ioctl,
    .hw_params      = dma_hw_params,
    .hw_free        = dma_hw_free,
    .prepare        = dma_prepare,
    .trigger        = dma_trigger,
    .pointer        = dma_pointer,
    .mmap           = dma_mmap,
};
```

DMA.c 文件

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

DMA.c 文件

在 probe 函数中

samsung_asoc_dma_platform_register(struct device *dev)

{ 这个函数是在 I2S 文件中被调用的，所以在 machine 文件里面用的是.platform_name = "s3c24xx-iis",

return snd_soc_register_platform(dev, &samsung_asoc_platform);这是注册 DMA 的函数

}

DMA.c 文件

```
int dma_trigger(struct snd_pcm_substream
*substream, int cmd)
{
switch (cmd)
case SNDRV_PCM_TRIGGER_START://根据 cmd
启动 dma 传输
case SNDRV_PCM_TRIGGER_STOP://根据 cmd 停
止 dma 传输
}
static int dma_prepare(struct
snd_pcm_substream *substream)
{
准备 DMA 传输。
复位各种状态信息。

}
```

再来看看 Ioctl(fd， SNDRV_PCM_IOCTL_HW_PARAMS)的 hw_params

machine(s3c24xx-uda134x.c)文件里面有 hw_params 函数，所以要实现该函数

cpu_dai(s3c24xx-i2s.c)文件里面有 hw_params 函数，所以要实现该函数

DMA.c 文件里面有 hw_params 函数，所以要实现该函数

codec(uda134x.c)文件里面有 hw_params 函数，所以要实现该函数

Machine 文件(s3c24xx-uda134x.c)
s3c24xx_uda134x_hw_params(struct snd_pcm_substream *substream,struct snd_pcm_hw_params *params)
{
}

Machine 文件(s3c24xx-uda134x.c)

```
Static struct platform_device *s3c24xx_uda134x_snd_device;
static struct snd_soc_ops s3c24xx_uda134x_ops = {
        .startup = s3c24xx_uda134x_startup,
        .shutdown = s3c24xx_uda134x_shutdown,
        .hw_params = s3c24xx_uda134x_hw_params,
};static  struct  snd_soc_dai_link  s3c24xx_uda134x_dai_link  =  {
        .name = "UDA134X",
        .stream_name = "UDA134X",
        .codec_name = "uda134x-codec",
        .codec_dai_name = "uda134x-hifi",
        .cpu_dai_name = "s3c24xx-iis",
        .ops = &s3c24xx_uda134x_ops,
        .platform_name      = "s3c24xx-iis",
};
static struct snd_soc_card snd_soc_s3c24xx_uda134x = {
        .name = "S3C24XX_UDA134X",
        .owner = THIS_MODULE,
        .dai_link = &s3c24xx_uda134x_dai_link,
        .num_links = 1,
};
```

在 probe 函数中

```
s3c24xx_uda134x_snd_device=
platform_device_alloc("soc-audio", -1);
platform_set_drvdata(s3c24xx_uda134x_snd_
device,&snd_soc_s3c24xx_uda134x);
```

machine 声卡创建注册完成

codec 文件(uda134x.c)
codec 部分(uda134x.c)
static int uda134x_hw_params(struct snd_pcm_substream *substream,struct snd_pcm_hw_params *params,struct snd_soc_dai *dai)
{
    根据 params 的值设置 codec 芯片的寄存器
}

codec 文件(uda134x.c)
static struct snd_soc_codec_driver soc_codec_dev_uda134x =
{
.probe = uda134x_soc_probe,
.remove = uda134x_soc_remove,
.suspend = uda134x_soc_suspend,
.resume = uda134x_soc_resume,
.reg_cache_size = sizeof(uda134x_reg),
.reg_word_size = sizeof(u8),
.reg_cache_default = uda134x_reg,
.reg_cache_step = 1,
.read = uda134x_read_reg_cache,
.write = uda134x_write,
.set_bias_level = uda134x_set_bias_level,
.....
};
static struct platform_device uda134_dev = {
.name = "uda134x-codec"， 这个名字就是 machine 里面的.codec_name = "uda134x-codec",
}
static struct platform_driver uda134x_codec_driver =
{ .driver = {
.name = "uda134x-codec",
.owner = THIS_MODULE,
},
.probe = uda134x_codec_probe,
.remove = uda134x_codec_remove,
};
在 probe 函数里面
snd_soc_register_codec(&pdev->dev,&soc_codec_dev_uda134x, &uda134x_dai, 1);
注册 driver 和 dai

codec 文件(uda134x.c)
struct snd_soc_dai_ops uda134x_dai_ops = {
.startup = uda134x_startup, //这里又一个 startup 函数 这个函数是被 soc_pcm_open 里面 的 if (codec_dai->driver->ops && codec_dai->driver->ops->startup)调用
.shutdown = uda134x_shutdown,
.hw_params = uda134x_hw_params,
.digital_mute = uda134x_mute,
.set_sysclk = uda134x_set_dai_sysclk,
.set_fmt = uda134x_set_dai_fmt,
};

static struct snd_soc_dai_driver uda134x_dai = {
.name = "uda134x-hifi",
/* playback capabilities */
.playback = {
.stream_name = "Playback",
.channels_min = 1,
.channels_max = 2,
.rates = UDA134X_RATES,
.formats = UDA134X_FORMATS,
},
/* capture capabilities */
.capture = {
.stream_name = "Capture",
.channels_min = 1,
.channels_max = 2,
.rates = UDA134X_RATES,
.formats = UDA134X_FORMATS,
},
/* pcm operations */
.ops = &uda134x_dai_ops,
};

cpu_dai 部分(s3c24xx-i2s.c)文件

```
static int s3c24xx_i2s_hw_params(struct snd_pcm_substream *substream,struct snd_pcm_hw_params *params,struct snd_soc_dai *dai)
{
    根据 params 的值设置 i2s 控制器
    switch (params_format(params)) //获取音频采样位数
    {
    case SNDRV_PCM_FORMAT_S8:    //采样位数为 8 位
        设置 cpu  里面的 I2S 寄存器

    case SNDRV_PCM_FORMAT_S16_LE: //采样位数为 16 位
        设置 cpu  里面的 I2S 寄存器
    }

    fs=params_rate(params)；//获取采样率
}
```

cpu_dai 部分(s3c24xx-i2s.c)文件

```
static const struct snd_soc_dai_ops s3c24xx_i2s_dai_ops = {
    .trigger      = s3c24xx_i2s_trigger,触发传输
    .hw_params    = s3c24xx_i2s_hw_params,设置某个参数
    .set_fmt      = s3c24xx_i2s_set_fmt,
    .set_clkdiv   = s3c24xx_i2s_set_clkdiv,
    .set_sysclk   = s3c24xx_i2s_set_sysclk,
};
static const struct snd_soc_component_driver s3c24xx_i2s_component = {
.name             = "s3c24xx-i2s",//必须和 mahcine 里面的.cpu_dai_name = "s3c24xx-iis",相同
};
static struct snd_soc_dai_driver s3c24xx_i2s_dai = {
    .probe = s3c24xx_i2s_probe,
    .suspend = s3c24xx_i2s_suspend,
    .resume = s3c24xx_i2s_resume,
    .playback = {
        .............
    }
    .capture = {
        ..............
    }
    .ops = &s3c24xx_i2s_dai_ops,
};
```

在 probe 函数中

```
 snd_soc_register_component(&pdev->dev,    &s3c24xx_i2s_component,&s3c24xx_i2s_dai, 1);注册 I2Sdai
ret = samsung_asoc_dma_platform_register(&pdev->dev);注册 dma
```

根据以上的驱动分析方法我们就只需要记住这几个框架就行了

open /dev/snd/pcmC0D0p 的时候会去调用 dma.c 文件里面的 dma_open 函数，其余的文件不执行任何操作。

Ioctl(fd，SNDRV_PCM_IOCTL_HW_PARAMS) 的时候会去调用

machine(s3c24xx-uda134x.c)文件里面有 hw_params 函数

cpu_dai(s3c24xx-i2s.c)文件里面有 hw_params 函数

DMA.c 文件里面有 hw_params 函数

codec(uda134x.c)文件里面有 hw_params 函数