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

DMA.c文件

static int dma\_open(struct snd\_pcm\_substream \*substream){

struct snd\_pcm\_runtime \*runtime = substream->runtime;

snd\_pcm\_hw\_constraint\_integer(runtime, SNDRV\_PCM\_HW\_PARAM\_PERIODS);

snd\_soc\_set\_runtime\_hwparams(substream, &dma\_hardware);//设置数据存放的格式。支持的通道数和传输方面的信息

申请中断处理函数 request\_irq(DMA2，.....，.....，)；传输完成一个period之后发生中断

}

if (platform->driver->ops && platform->driver->ops->open)

DMA.c文件

static irqreturn\_t dma2()

{

更新状态信息，然dma指针指向buffer里面的第二个period

如果指针指向的是最后一个period，那么就将period返回到buffer缓冲区第一个位置

再次启动dma传输

}

为什么要用缓冲区buffer？

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  |  |  |  |  |  |  |  |  |  |

应用程序每写一段数据称为period

所以DMA要分配一个环形缓冲区

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  |  |  |  |  |  |  |  |  |  |  |

每个period里面有很多个frame，每个frams代表一个采样点。一个采样点包含左声道右声道rams

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

应用程序每写一段数据称为period

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  |  |  |  |  |  |  |  |  |  |  |

驱动程序读取period里面的数据，也就是alsa应用层写进来一个period，驱动读走一个period

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  |  |  |  |  |  |  |  |  |  |

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\_STREAM\_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\_soc\_platform\_driver samsung\_asoc\_platform = {

.ops = &dma\_ops,

.pcm\_new = dma\_new,

.pcm\_free = dma\_free\_dma\_buffers,

};

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

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)

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

Machine文件(s3c24xx-uda134x.c)

s3c24xx\_uda134x\_hw\_params(struct snd\_pcm\_substream \*substream,struct snd\_pcm\_hw\_params \*params)

{

}

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)

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,

};

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

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