Soc-core.c

module\_init(snd\_soc\_init)

snd\_soc\_init

soc\_probe

snd\_soc\_register\_card

snd\_soc\_instantiate\_cards

list\_for\_each\_entry(card, &card\_list, list)

snd\_soc\_instantiate\_card(card);

list\_for\_each\_entry(platform, &platform\_list, list)

card->platform == platform // s3c24xx-pcm.c

//pcm驱动进入音频核心层便是通过此函数以"插件"的 //形式插入音频子系统

//platform指向的是struct snd\_soc\_platform //s3c24xx\_soc\_platform:

ct snd\_soc\_platform s3c24xx\_soc\_platform = {

.name = "s3c24xx-audio",

.pcm\_ops = &s3c24xx\_pcm\_ops,

.pcm\_new = s3c24xx\_pcm\_new,

.pcm\_free = s3c24xx\_pcm\_free\_dma\_buffers,

};

list\_for\_each\_entry(dai, &dai\_list, list)

if (card->dai\_link[i].cpu\_dai == dai) // s3c24xx-i2s.c里 面初始化

//函数snd\_soc\_register\_dai()是音频子系统核心层的 //函数.DAI(Digital Audio Interface)便是通过此函数以" //插件"的形式插入音频子系统的核心层

dai指向的是struct snd\_soc\_dai s3c24xx\_i2s\_dai

===============================

struct snd\_soc\_dai s3c24xx\_i2s\_dai = {

.name = "s3c24xx-i2s",

.id = 0,

.probe = s3c24xx\_i2s\_probe,

.suspend = s3c24xx\_i2s\_suspend,

.resume = s3c24xx\_i2s\_resume,

.playback = {

.channels\_min = 2,

.channels\_max = 2,

.rates = S3C24XX\_I2S\_RATES,

.formats = SNDRV\_PCM\_FMTBIT\_S8 | SNDRV\_PCM\_FMTBIT\_S16\_LE,},

.capture = {

.channels\_min = 2,

.channels\_max = 2,

.rates = S3C24XX\_I2S\_RATES,

.formats = SNDRV\_PCM\_FMTBIT\_S8 | SNDRV\_PCM\_FMTBIT\_S16\_LE,},

.ops = &s3c24xx\_i2s\_dai\_ops,

};

=======================================

card->probe //未赋值

cpu\_dai->probe //平台相关的函数static int //s3c24xx\_i2s\_probe，平台相关的IIS总线 //配置

codec\_dev->probe //编解码器驱动

// sound/soc/codecs/uda134x.c,指向uda134x\_soc\_probe,

----------------------------------------------------------------------------------------------------------------------

uda134x\_soc\_probe

snd\_soc\_new\_pcms

//音频子系统的核心,实现音频控制的设备节点/dev/snd/controlCn,播放音频的设备节点/dev/snd/pcmCiDjp,录制音频的设备节点/dev/snd/pcmCiDjc都是在此函数生成,并在此函数在直接绑定了对应上层用户空间的系统调用//

向音频子系统注册一个声卡和PCM设备,主要完成AD/DA转换

snd\_card\_create

snd\_ctl\_create//创建控制音频接口的设备节点,如音量控制

//创建了/dev/snd/controlCn(n = 0,1,2,...)设备节点.并且在此设备节点上绑定了 //操作集snd\_ctl\_f\_ops,来最直接对应用户空间的系统调用,储如read、write、 //ioctl等

=====================

static struct snd\_device\_ops ops = {

.dev\_free = snd\_ctl\_dev\_free,

.dev\_register = snd\_ctl\_dev\_register,

.dev\_disconnect = snd\_ctl\_dev\_disconnect,

};

=========================

snd\_device\_new

list\_add(&dev->list,card)

//设备节点的生成及操作集的绑定都是在回调函数

//snd\_ctl\_dev\_register()里面完成

soc\_new\_pcm//生成了音频里面的两个设备节点(播音，录音)

/dev/snd/pcmCiDjp(i = 0,1,2,...; j = 0,1,2,...) //播音 /dev/snd/pcmCiDjc(i = 0,1,2,...; j = 0,1,2,...)//录音

//生成播音和放音这两个设备节点:/dev/snd/pcmCiDjp和/dev/snd/pcmCiDic.并绑定 //了最直接面向用户空间的系统调用的操作集snd\_pcm\_f\_ops.

        //和函数snd\_ctl\_create()相仿,内部都调用了函数snd\_device\_new()

snd\_pcm\_new

===============================

static struct snd\_device\_ops ops = {

.dev\_free = snd\_pcm\_dev\_free,

.dev\_register = snd\_pcm\_dev\_register,

.dev\_disconnect = snd\_pcm\_dev\_disconnect,

};

==============================

snd\_pcm\_new\_stream

snd\_device\_new

snd\_pcm\_set\_ops(pcm,SNDRV\_PCM\_STREAM\_PLAYBACK,

&soc\_pcm\_ops);

snd\_pcm\_set\_ops(pcm, SNDRV\_PCM\_STREAM\_CAPTURE,

&soc\_pcm\_ops);

snd\_soc\_init\_card

snd\_card\_register

snd\_device\_register\_all

dev->ops->dev\_register(dev)//该函数分别会调用ctrl和pcm的回调函数：

snd\_ctl\_dev\_register

snd\_pcm\_dev\_register

snd\_pcm\_dev\_register

case SNDRV\_PCM\_STREAM\_PLAYBACK:播放文件节点名

sprintf(str, "pcmC%iD%ip", pcm->card->number, pcm->device);

…………………最会会在/dev/snd下生成pcmC0D0p

break;

case SNDRV\_PCM\_STREAM\_CAPTURE:采集文件节点名

sprintf(str,"pcmC%iD%ic",pcm->card->numberpcm->device);

……………………最后会在/dev/snd下生成pcmC0D0c

break;

}

snd\_register\_device\_for\_dev(devtype, pcm->card,

pcm->device,

&snd\_pcm\_f\_ops[cidx],

pcm, str, dev);

//分配并初始化一个snd\_minor实例；

        //保存该snd\_minor实例到snd\_minors数组中；

      //调用device\_create()生成设备文件节点。

该回调函数建立了和用户空间应用程序（alsa-lib）通信所用的设备文件节点:/dev/snd/pcmCxxDxxp和/dev/snd/pcmCxxDxxc

* Playback(播放)    如何把用户空间的应用程序发过来的PCM数据，转化为人耳可以辨别的模拟音频//将数字信息模拟化发到硬件设备。
* Capture（采集）     把mic拾取到得模拟信号，经过采样、量化，转换为PCM信号送回给用户空间的应用程序//将硬件采集的模拟信息数字化发给上层应用。

const struct file\_operations snd\_pcm\_f\_ops[2] = {

----playback

{

.owner = THIS\_MODULE,

.write = snd\_pcm\_write,

.aio\_write = snd\_pcm\_aio\_write,

.open = snd\_pcm\_playback\_open,

.release = snd\_pcm\_release,

.llseek = no\_llseek,

.poll = snd\_pcm\_playback\_poll,

.unlocked\_ioctl = snd\_pcm\_playback\_ioctl,

.compat\_ioctl = snd\_pcm\_ioctl\_compat,

.mmap = snd\_pcm\_mmap,

.fasync = snd\_pcm\_fasync,

.get\_unmapped\_area = snd\_pcm\_get\_unmapped\_area,

},

{

------capture

.open = snd\_pcm\_capture\_open,

.unlocked\_ioctl = snd\_pcm\_capture\_ioctl,

……………………….}与上类似。

};以上为pcm的操作函数。

platform->probe //平台驱动

uda134x\_soc\_probe

snd\_soc\_new\_pcms

snd\_card\_create

soc\_new\_pcm

snd\_pcm\_new

snd\_pcm\_new\_stream

snd\_device\_new

snd\_pcm\_set\_ops (pcm, SNDRV\_PCM\_STREAM\_PLAYBACK, &soc\_pcm\_ops);

snd\_pcm\_set\_ops

// soc\_pcm\_ops.注意,这个操作集将是上层到IIS硬件层中间层的第二"逻辑层 //"(第一层是上述的snd\_pcm\_f\_ops).并且,在函数snd\_pcm\_set\_ops()暂存在 //动态分配的pcm的streams域

struct snd\_pcm\_substream \*substream

substream->ops = ops

二、static int snd\_pcm\_playback\_open(struct inode \*inode, struct file \*file)

OPEN:

以播放声音为例,当用户空间对设备节点/dev/sdn/pcmD0C0p进行open时,

第一响应的是snd\_pcm\_f\_ops.如下:

.open =         snd\_pcm\_playback\_open,

open函数里面一般来说其内容是最丰富的,比如通过file->private\_data来暂存一些设备相关信息,后续的read()、write()等动作可以通过file->private\_data来提取这些设备信息.函数

 snd\_pcm\_playback\_open

snd\_pcm\_open

snd\_pcm\_open\_file

snd\_pcm\_open\_substream

snd\_pcm\_attach\_substream

\*rsubstream = substream;

               pcm\_file-**>**substream = substream;

 file-**>**private\_data = pcm\_file;

//   pcm还是那个pcm,播音录音音频流的操作集(pcm->stream->ops)在函 //数snd\_pcm\_set\_ops()都被赋值为soc\_pcm\_ops.并且在open函数里面被/ //保存在file->private\_data

WRITE:

以write为例,看用户空间的音频数据流是如何流窜到硬件IIS的.大体流程如下

file\_operations snd\_pcm\_f\_ops

.write = snd\_pcm\_write

pcm\_file = file->private\_data

substream = pcm\_file->substream

snd\_pcm\_lib\_write(substream, buf, count)

snd\_pcm\_lib\_write1

snd\_pcm\_start

snd\_pcm\_action (&snd\_pcm\_action\_start,

substream,SNDRV\_PCM\_STATE\_RUNNING);

================

static struct action\_ops snd\_pcm\_action\_start = {

.pre\_action = snd\_pcm\_pre\_start,

.do\_action = snd\_pcm\_do\_start,

.undo\_action = snd\_pcm\_undo\_start,

.post\_action = snd\_pcm\_post\_start

};

==========================

snd\_pcm\_action\_single

ops->do\_action //即static int snd\_pcm\_do\_start

🡪 snd\_pcm\_do\_start

substream->ops->trigger//即static int soc\_pcm\_trigger

🡪soc\_pcm\_trigger

cpu\_dai->ops->trigger //调用了s3c24xx-i2s.c

🡪 s3c24xx\_i2s\_trigger

写音频数据到音频接口IIS流程：

UserSpace ->

snd\_pcm\_f\_ops -> snd\_pcm\_action\_start -> soc\_pcm\_ops -> s3c24xx\_i2s\_trigger

主要是由上层写数据到s3c2440主控的硬件iiS接口处。

声卡编解码器驱动：

uda134x\_soc\_probe

snd\_soc\_new\_pcms

snd\_soc\_add\_controls

snd\_ctl\_add(card, snd\_soc\_cnew(control, codec, NULL));

snd\_soc\_cnew

kctl.put = ncontrol->put;

//ncontrol即为uda1340\_snd\_controls

================================

static const struct snd\_kcontrol\_new uda1341\_snd\_controls[] = {

SOC\_SINGLE("Master Playback Volume", UDA134X\_DATA000, 0, 0x3F, 1),

SOC\_SINGLE("Capture Volume", UDA134X\_EA010, 2, 0x07, 0),

SOC\_SINGLE("Analog1 Volume", UDA134X\_EA000, 0, 0x1F, 1),

……}

宏展开：

#define SOC\_SINGLE(xname, reg, shift, max, invert) \

{ .iface = SNDRV\_CTL\_ELEM\_IFACE\_MIXER, .name = xname, \

.info = snd\_soc\_info\_volsw, .get = snd\_soc\_get\_volsw,\

.put = snd\_soc\_put\_volsw, \

.private\_value = SOC\_SINGLE\_VALUE(reg, shift, max, invert) }

控制声音播放音量流程：

==> /dev/snd/controlC 绑定 snd\_ctl\_f\_ops //从设备节点到底层寄存器

->static const struct file\_operations snd\_ctl\_f\_ops

->snd\_ctl\_ioctl()

->snd\_ctl\_elem\_write\_user()

->snd\_ctl\_elem\_write()

->result = kctl->put(kctl, control);??疑点在这,kctl是如何跟uda1341\_snd\_controls

++++++++++++++++++++++++++++++++++++++

snd\_soc\_add\_controls

snd\_ctl\_add

SOC\_SINGLE("Analog1 Volume", UDA134X\_EA000, 0, 0x1F, 1)

+++++++++++++++++++++++++++++++++++++++

->put = snd\_soc\_put\_volsw()

->snd\_soc\_update\_bits()

->snd\_soc\_write()

->codec->write()

->codec->write = uda134x\_write()

大体的执行流程:

UserSpace-->snd\_ctl\_f\_ops-->[kctl->put]-->uda1340\_snd\_controls->put

主要是上层应用通过音频芯片的操作函数，进行写数据。