linux2.6内核ALSA框架分析和imx6-wm8960移植

machine部分：

1. 定义结构体static struct snd\_soc\_card snd\_soc\_s3c24xx\_uda134x
2. 定义平台的结构体static struct platform\_device \*s3c24xx\_uda134x\_snd\_device;
3. platform\_set\_drvdata(s3c24xx\_uda134x\_snd\_device,&snd\_soc\_s3c24xx\_uda134x);设置结构体私有数据
4. 注册mahcine 里面的设platform\_device\_add(s3c24xx\_uda134x\_snd\_device);

I2S部分：

1. 定义结构体static struct snd\_soc\_dai\_driver s3c24xx\_i2s\_dai
2. 定义平台static const struct snd\_soc\_component\_driver s3c24xx\_i2s\_component
3. snd\_soc\_register\_component(&pdev->dev, &s3c24xx\_i2s\_component,&s3c24xx\_i2s\_dai, 1);

//这个snd\_soc\_register\_component里面包含了snd\_soc\_register\_dai

DMA部分

1. 定义结构static struct snd\_soc\_platform\_driver samsung\_asoc\_platform
2. 注册snd\_soc\_register\_platform(dev, &samsung\_asoc\_platform);

Codec部分：

1. 定义结构static struct snd\_soc\_codec\_driver soc\_codec\_dev\_uda134x
2. 定义I2S结构static struct snd\_soc\_dai\_driver uda134x\_dai
3. snd\_soc\_register\_codec(&pdev->dev,&soc\_codec\_dev\_uda134x, &uda134x\_dai, 1);//注册

以上就是S3C2440平台和音频芯片的框架功能实现。

Machine就是snd\_soc\_card这个结构

CPU的I2S就是snd\_soc\_dai\_driver这个结构体

DMA就是snd\_soc\_platform\_driver这个结构

Codec芯片是snd\_soc\_codec\_driver这个结构

codec的I2S总线是snd\_soc\_dai\_driver这个结构

以上结构的实现

Machine部分的实现

static struct snd\_soc\_card snd\_soc\_s3c24xx\_uda134x = {

.name = "S3C24XX\_UDA134X",

.owner = THIS\_MODULE,

.dai\_link = &s3c24xx\_uda134x\_dai\_link,//主要就是这个

.num\_links = 1,

};

static struct snd\_soc\_dai\_link s3c24xx\_uda134x\_dai\_link = {

.name = "UDA134X",//名字无所谓

.stream\_name = "UDA134X",//名字无所谓

.codec\_name = "uda134x-codec",//音频芯片驱动uda134x.c里面//platform\_driver的名字

.codec\_dai\_name = "uda134x-hifi",//音频芯片驱动uda134x.c里面的I2S接口

//snd\_soc\_dai\_driver，dai里面的名字

.cpu\_dai\_name = "s3c24xx-iis",//s3c24xx-iis.c里面platform\_driver的名字

.ops = &s3c24xx\_uda134x\_ops,// struct snd\_soc\_ops s3c24xx\_uda134x\_ops单独实现

.platform\_name = "DMA",//这里应该是DMA.C里面DMA的名字

};

static struct snd\_soc\_ops s3c24xx\_uda134x\_ops = {

.startup = s3c24xx\_uda134x\_startup,

.shutdown = s3c24xx\_uda134x\_shutdown,

.hw\_params = s3c24xx\_uda134x\_hw\_params,

};

I2S部分实现：

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

.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,

};

static const struct snd\_soc\_dai\_ops s3c24xx\_i2s\_dai\_ops = {

.trigger = s3c24xx\_i2s\_trigger,

.hw\_params = s3c24xx\_i2s\_hw\_params,

.set\_fmt = s3c24xx\_i2s\_set\_fmt,

.set\_clkdiv = s3c24xx\_i2s\_set\_clkdiv,

.set\_sysclk = s3c24xx\_i2s\_set\_sysclk,

};

DMA部分实现：

static struct snd\_soc\_platform\_driver samsung\_asoc\_platform = {

.ops = &dma\_ops,

.pcm\_new = dma\_new,

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

};

Uda1341部分实现：

static const struct snd\_soc\_dai\_ops uda134x\_dai\_ops = {

.startup = uda134x\_startup,

.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,

};

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,

.dapm\_widgets = uda134x\_dapm\_widgets,

.num\_dapm\_widgets = ARRAY\_SIZE(uda134x\_dapm\_widgets),

.dapm\_routes = uda134x\_dapm\_routes,

.num\_dapm\_routes = ARRAY\_SIZE(uda134x\_dapm\_routes),

};

以上就是四个部分的结构体实现过程，结构体里面的函数和参数都要另外实现这里就不多讲了。

在linux内核里面配置声卡驱动make menuconfig

I2s驱动在：

-->Device Drivers

-->Sound card support

--> Advanced Linux Sound Architecture

-->ALSA for SoC audio support

在这个下面就要支持2440的I2S总线，如果是IMX6平台就是Synopsys I2S Device Driver

Machine驱动在：

-->Device Drivers

-->Sound card support

--> Advanced Linux Sound Architecture

-->ALSA for SoC audio support

Machine文件2440在这个文件下面，如果是imx6-wm8962就在SoC Audio for Freescale CPUs --->这个里面.

Machine 常用的四个snd\_soc函数API

snd\_soc\_dai\_set\_fmt()

snd\_soc\_dai\_set\_pll()

snd\_soc\_dai\_set\_sysclk()

snd\_soc\_dai\_set\_clkdiv()

snd\_soc\_dai\_set\_fmt设置I2S的格式

if (data->is\_codec\_master)

fmt = SND\_SOC\_DAIFMT\_I2S | SND\_SOC\_DAIFMT\_NB\_NF |

SND\_SOC\_DAIFMT\_CBM\_CFM;

else

fmt = SND\_SOC\_DAIFMT\_I2S | SND\_SOC\_DAIFMT\_NB\_NF |

SND\_SOC\_DAIFMT\_CBS\_CFS;

snd\_soc\_dai\_set\_fmt(cpu\_dai, fmt);调用snd\_soc\_dai\_set\_fmt函数将格式注册进系统

snd\_soc\_dai\_set\_sysclk设置编解码芯片ADC和DAC系统时钟

if (!(params\_rate(params) % 11025))

snd\_soc\_dai\_set\_sysclk(codec\_dai, WM8994\_SYSCLK\_MCLK1, 11289600,SND\_SOC\_CLOCK\_IN);

else

snd\_soc\_dai\_set\_sysclk(codec\_dai, WM8994\_SYSCLK\_MCLK1, 12288000,SND\_SOC\_CLOCK\_IN);

snd\_soc\_dai\_set\_pll()设置Codec的PLL频率：11289600

snd\_soc\_dai\_set\_pll(codec\_dai,1,0,priv->sysclk/2,pll\_out)

/\* 设置Codec的系统时钟：11289600\*/

 snd\_soc\_dai\_set\_clkdiv( codec\_dai, WM8960\_SYSCLKDIV, sysclk\_div );

    /\* 设置采样频率（LRCK）的时钟，lrckclk= sysclk/1\*256=44100 \*/

    rsnd\_soc\_dai\_set\_clkdiv( codec\_dai, WM8960\_DACDIV, dacdiv);

    /\* 设置D类放大器时钟频率:dclk= sysclk/16 \*/

    snd\_soc\_dai\_set\_clkdiv( codec\_dai, WM8960\_DCLKDIV,dclk);

    /\* 设置BCLK类放大器时钟频率:bitclk=sysclk/4 = 2822400 \*/

     snd\_soc\_dai\_set\_clkdiv( codec\_dai, WM8960\_BCLKDIV, bclk);

讲述IMX6平台移植WM8960音频芯片方法

首先imx6移植WM8960只需要修改machine文件和设备树文件就可以了。

Machine文件是imx-wm8960.c 但是这个文件有好几个版本，我在linux3.14.28的内核上用了老版本内核里面的imx-wm8960.c，就会出现有几个函数编译不过

所以我在linux3.14.28内核上用的是linux3.14.52或者linux4.0内核版本里面的imx-wm8960.c文件。我将文件改成了imx6-wm8960.c所以拷贝到linux3.14.28内核后将imx6-wm8960.c改为imx-wm8960.c，然后再Makefile和Kconfig里面加上配置参数。然后编译会出现一个错误sound/soc/fsl/imx-wm8960.c:399:39: error: 'struct fsl\_sai' has no member named 'is\_stream\_opened'。

这个错误的修改方式是因为linux3.14.28的fsl\_sai.h没有定义bool is\_stream\_opened[2]; 这个整形。这个整形在linux3.14.52或者linux4.0版本中的fsl\_sai.h里面定义了的，所以将高版本的这个整形变量拷贝到linux3.14.28里面的fsl\_sai.h就可以了。

设备树移植是按照IMX6ul里面内核的方法来移植，如果imx6solo要将wm8962换成wm8960的话，建议下载一个linux3.14.52或者linux4.0内核，里面有imx6ul的内核，也有imx6，imx7的内核，这样就很方便。

但是我已经在文件夹里面存放了imx6在linux3.14.28要移植wm8960的文件，所以也可以不用下载新版本的内核。

以上wm8960移植方法有许多地方有许多问题

所以请看下面新的移植方法

首先imx6移植WM8960只需要修改machine文件和设备树文件就可以了。

这个套路和上面相似

但是machine文件要下载linux-3.10.17-imx文件里面的imx-wm8960.c

所以最好去github里面下载，地址为：<https://github.com/xuhuashan/imx6q/tree/47a7ef6a73667bd855af3ae4b14f370800983de2/linux-3.10.17-imx/arch/arm/boot>

然后dts设备树不要linux-3.10.17-imx目录里面的标准IM6qdl-sabresd.c设备树来，其他linux版本也是一样，所以要按照linux-3.10.17-imx目录里面的**imx6q-tqe9.dts 天嵌的版本是最好的。**

**这样machine和dts都已经移植好了，但是在编译的时候会报错。**

**1：报machine文件里面的函数错误codec\_dev = of\_find\_i2c\_device\_by\_node(codec\_np);**

**if (!codec\_dev || !codec\_dev->driver) {**

**dev\_err(&pdev->dev, "failed to find codec platform device\n");**

**ret = -EINVAL;**

**goto fail;**

**}**

**这个错误是因为codec\_dev->driver是linux3.10.17版本里面支持的，linux3.14.28版本必须要这样codec\_dev->dev.driver 也就是在driver前面加dev，因为该结构又被封装了。**

**2：报错头文件问题，头文件i2c是include<linux/of\_i2c.h>,但是linux3.14.28在i2c头文件里面已经封装了of函数了所有不需要of\_i2c.h，直接i2c.h就行。**

3：aplay音乐的时候I2S总线只有MCLK有时钟，其他IO没有时钟，比如BCLK, DACLRCLK,没有时钟输出。还有ADDAT,DACDAT没有数据输出。这个问题是因为没有打开线性pcm开关命令

打开amixer numid=41,iface=MIXER,name='Right Output Mixer PCM Playback Switch'

打开amixer numid=44,iface=MIXER,name='Left Output Mixer PCM Playback Switch'

这样你就可以听到耳机输出正常的声音了。

在aplay播放的时候，驱动程序会进入machine文件里面执行hw\_params回调函数，同时也要进入codec文件里面执行set\_pll回调函数, set\_fmt回调函数，hw\_params回调函数。

在arecord录音的时候，驱动程序会进入machine文件里面执行hw\_params回调函数，同时也要进入codec文件里面执行set\_pll回调函数，set\_fmt回调函数，hw\_params回调函数。

所以主要修改这两个回调函数里面的寄存器设置

所以在wm8960.c里面

static const struct snd\_soc\_dai\_ops wm8960\_dai\_ops = {

.hw\_params = wm8960\_hw\_params, aplay和arecord会执行该函数

.digital\_mute = wm8960\_mute,

.set\_fmt = wm8960\_set\_dai\_fmt, aplay和arecord会执行该函数

.set\_clkdiv = wm8960\_set\_dai\_clkdiv,

.set\_pll = wm8960\_set\_dai\_pll, aplay和arecord会执行该函数

};

在imx-wm8960.c里面

static struct snd\_soc\_ops imx\_hifi\_ops = {

.startup = imx\_hifi\_startup, 执行aplay和arecord会第一个执行该函数

.shutdown = imx\_hifi\_shutdown, ctrl+C或者自动音乐播放完会执行该函数

.hw\_params = imx\_hifi\_hw\_params, aplay和arecord会执行该函数

.hw\_free = imx\_hifi\_hw\_free,

};

但是我加入了TQ的imx-wm8960.c可以播放声音为什么无法录音呢,这个问题，我自己没有找到，但是找方案商解决了这个问题，将目录下方案商修改好的imx-wm8960.c文件用来替换内核里面现有的imx-wm8960.c文件，路径fsl-linux/sound/soc/fsl

然后将方案商改好的wm8960.c文件替换内核现有的wm8960.c文件，路径fsl-linux/sound/soc/codecs

然后再修改设备树文件

这是设备树在内核的路径



然后make，将编译好的内核放入开发板，然后启动。

numid=41,iface=MIXER,name='Right Output Mixer PCM Playback Switch'

打开这个右边耳机的开关，aplay的音乐才能从右耳机播放出来

numid=44,iface=MIXER,name='Left Output Mixer PCM Playback Switch'

打开这个左边耳机的开关，aplay的音乐才能从左耳机播放出来

numid=9,iface=MIXER,name='Headphone Playback Volume'设置耳机输出音量大小

然后就可以用aplay播放音乐了。

下面我们来看看录音

arecord [-Dplughw:0,0] -r 44100 -f S16\_LE -c 2 -d 5 record.wav

录音成功，能分别播放左右声道的声音

但是感觉录音的时候出现了很多背景噪音

numid=25,iface=MIXER,name='ALC Max Gain'

将ALC Max Gain降到4档，录音底噪就要下降很多

然后我们测试下直通，就是麦克风录入声音，然后从耳机直接输出出来。

numid=46,iface=MIXER,name='Left Output Mixer Boost Bypass Switch'

左声道麦克风和耳机直通

numid=43,iface=MIXER,name='Right Output Mixer Boost Bypass Switch'

右声道麦克风和耳机直通

arecord [-Dplughw:0,0] -r 44100 -f S16\_LE -c 2 -d 5 record.wav

直通功能光是打开开关还不行，还要执行arecod，然后直通才能启动，arecod执行多久直通就只能执行这么久

numid=35,iface=MIXER,name='Left Output Mixer Boost Bypass Volume'

左声道麦克风直通录音声音大小

numid=37,iface=MIXER,name='Right Output Mixer Boost Bypass Volume'

右声道麦克风直通录音声音大小

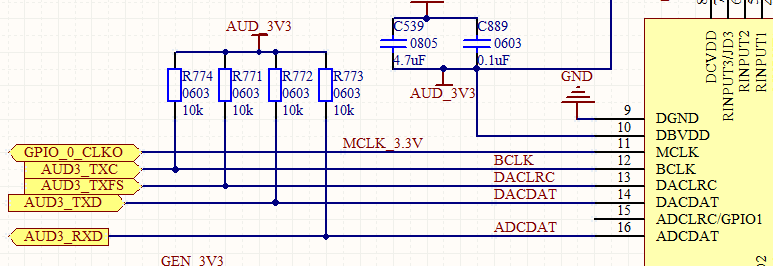
**WM8960硬件设计注意事项**

WM8960芯片DVDD引脚如果接的3.3V

那么I2S总线和MCLK必须接3.3V，有一个没有接3.3V接的是1.8V就会出现aplay一会能播放，一会播放不了。最好使用上拉电阻接3.3V

IMX6公版的WM8962是将CPU的MCLK3.3V，用芯片降到了1.8V，导致我以为I2S接3.3V，然后MCLK接的1.8V。所以也出现了aplay不能正常播放。

一定要注意DVDD如果接的3.3V，那么I2S总线和MCLK必须接3.3V



因为IMX6芯片clk输出电压默认是3.3V，所以mclk我没有接上拉电阻，记住MCLK一定要和I2S总线电压一样，我就是以为I2S总线接3.3V就可以了，MCLK不用接3.3V，导致音乐播放和录音不稳定，后来将MCLK改成3.3V就OK了，所以一定要谨记。

上面原理图是我正确的接法。

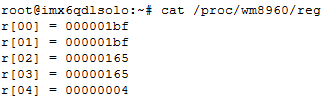
echo "1a 061" > /proc/wm8960/reg

这种echo是直接把061的值写在1a地址上，1a是wm8960寄存器地址，061是值



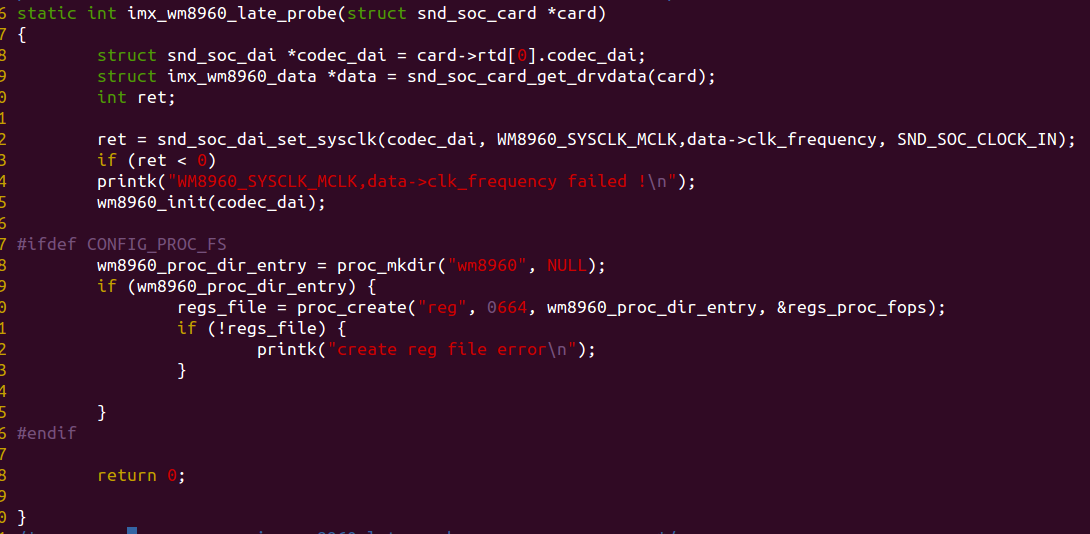
cat /proc/wm8960/reg 用于跟踪寄存器的状态，按Ctrl+c结束

我们看看刚才写入1a寄存器的值对不对

一定要手速快赶紧退出否则就刷屏了



这样就很好调试WM8960寄存器了，这是怎么做到的？



**这就是/proc/wm8960产生的代码**

在IMX6-wm8960.c初始化程序里加上这段代码，proc\_mkdir是创建/proc/wm8960目录的

如果没有proc\_mkdir这一段代码你向wm8960写寄存器就会出现问题。





文件没有创建，所以proc\_create函数是关键



**后面这个是实现&regs\_proc\_fops这个回调函数结构体的字符设备驱动**

