Linux ALSA声卡驱动开发最佳实践

※本实践是在《Linux 基础培训(2)-驱动开发最佳实践》基础上进行的

2014-1-2

目 录

2014-1-2

目 录

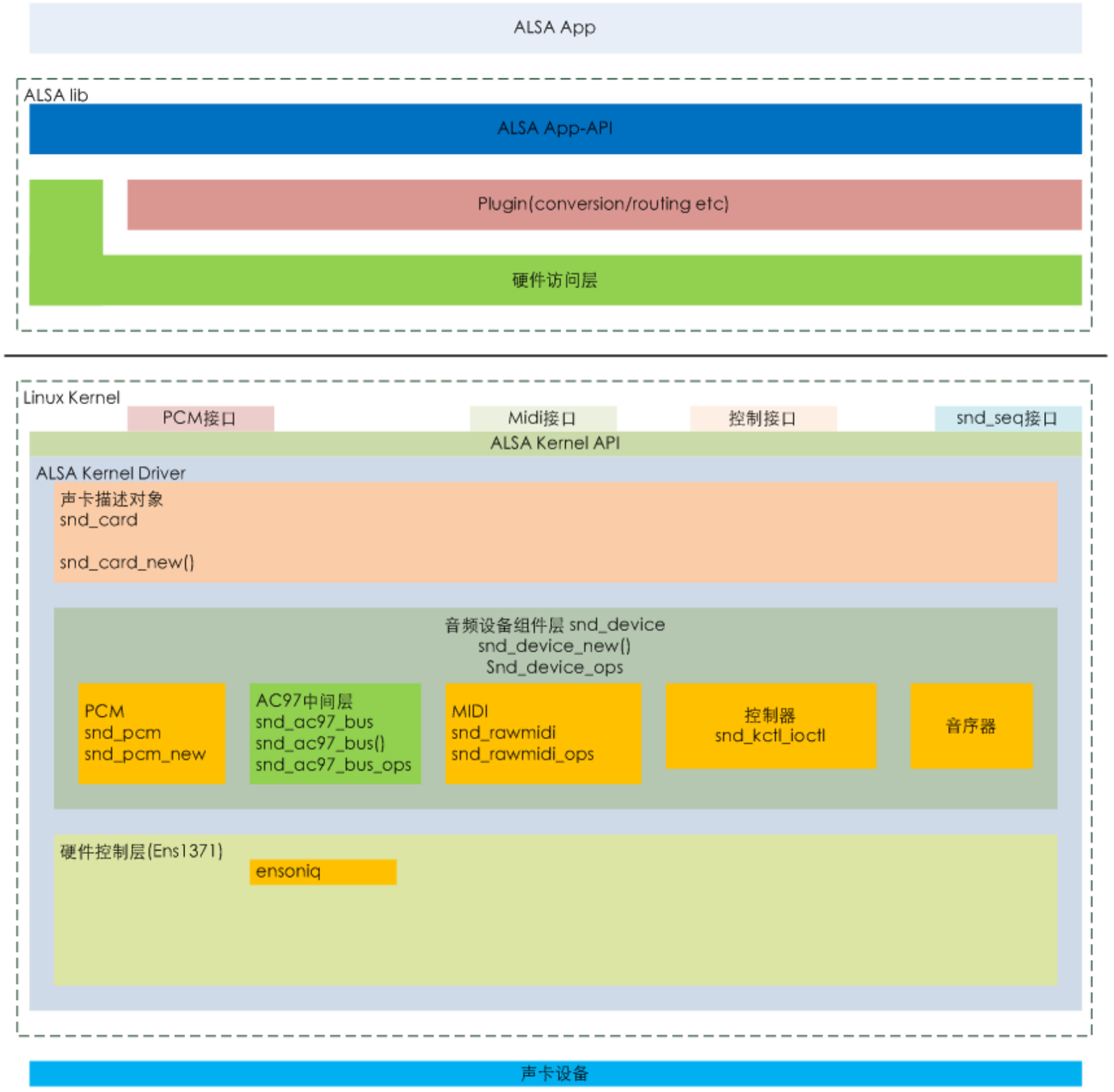
2014-1-2

一、ALSA架构简介

1. ALSA架构特点

- *支持多种声卡设备*
- *模块化的内核驱动程序*
- *支持SMP和多线程*
- *提供应用程序开发库*
- *兼容OSS应用程序开发*

一、ALSA架构简介



1. ALSA是Advanced Linux Sound Architecture，高级Linux声音架构的简称,它在Linux操作系统上提供了音频和MIDI（Musical Instrument Digital Interface，音乐设备数字化接口）的支持。它包含API 库和工具、内核驱动集合， 对 Linux 声音进行支持。 ALSA 包含一系列内核驱动对不同的声卡进行支持，还提供了 libasound 的 API 库。用这些进行写程序不需要打开设备等操作，所以编程人员在写程序的时候不会被底层的东西困扰。

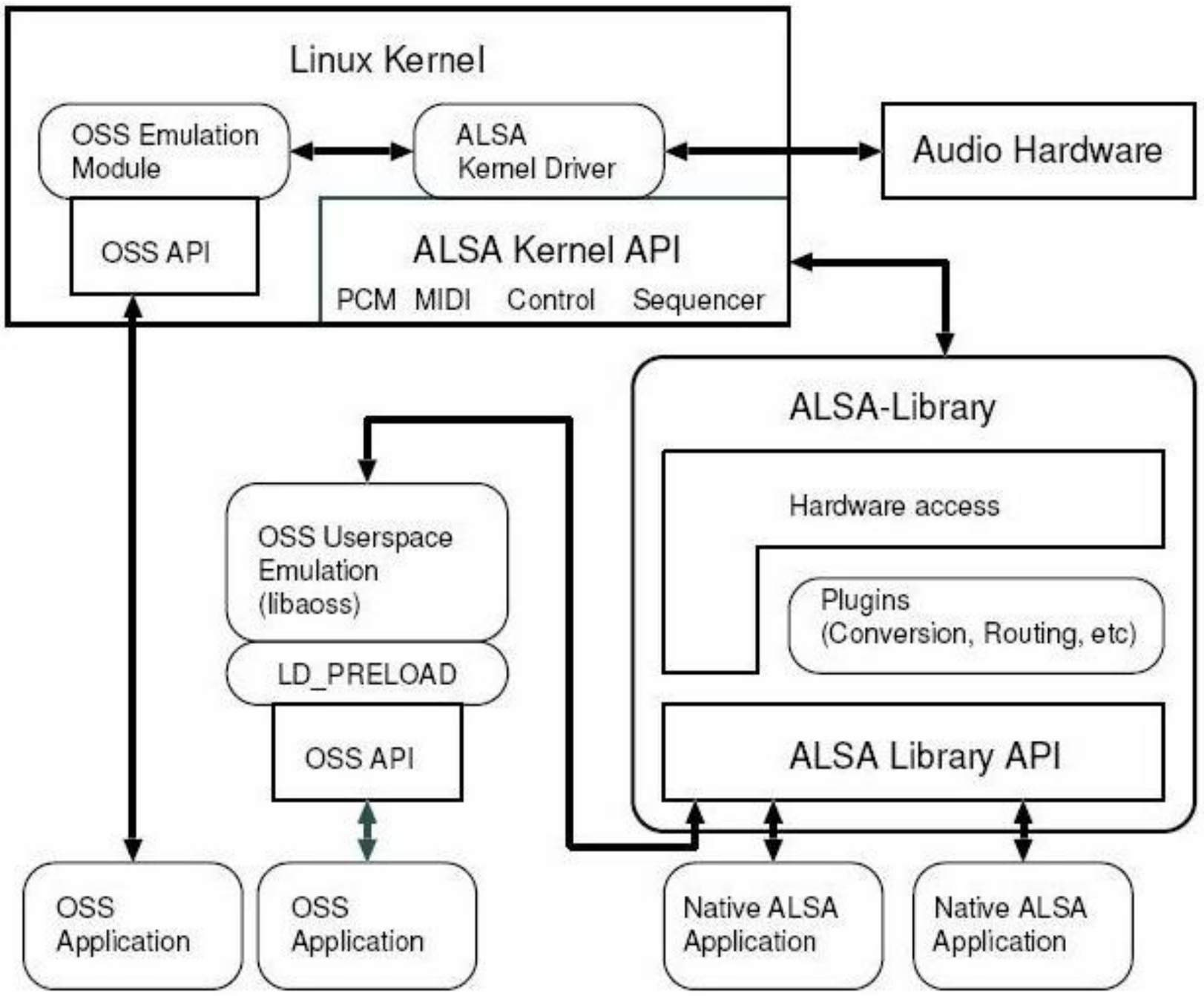
2. ALSA自带的应用程序是alsa-utils工具包，包括aplay、alsamixer等。aplay用于在ASLA上播放音频。alsamixer用于改变音频信号的音量。

3. alsa-lib是用户空间的函数库，提供了libasound.so给应用程序使用，应用程序应包含头文件asoundlib.h。这个库通过提供封装函数(ALSA-API)，使ALSA应用程序编写起来更容易。alsa-lib中有control，timer，dmix，pcm等，都是以插件(plugin)的形式存在的。alsa-lib通过硬件访问层的系统调用与内核层进行交互。

4. alsa-driver是音频设备的alsa内核部分的驱动。集成在内核里面，大多是以模块的方式存在。可分为三层。
   (1)最底层是硬件操控层，负责实现硬件操纵访问的功能，这也是声卡驱动程序中用户需实现的主要部分；
   (2)中间层是ASLA驱动的核心部分，它由各种功能的音频设备组件构成，为用户提供了一些预定义组件（如PCM、AC97、音序器和控制器等），另外用户也可以自行定义设备组件；
   (3)驱动的最上层是声卡对象描述层，它是声卡硬件的抽象描述，内核通过这些描述可以得知该声卡硬件的功能、设备组件和操作方法等。

2014-1-2

一、ALSA架构简介

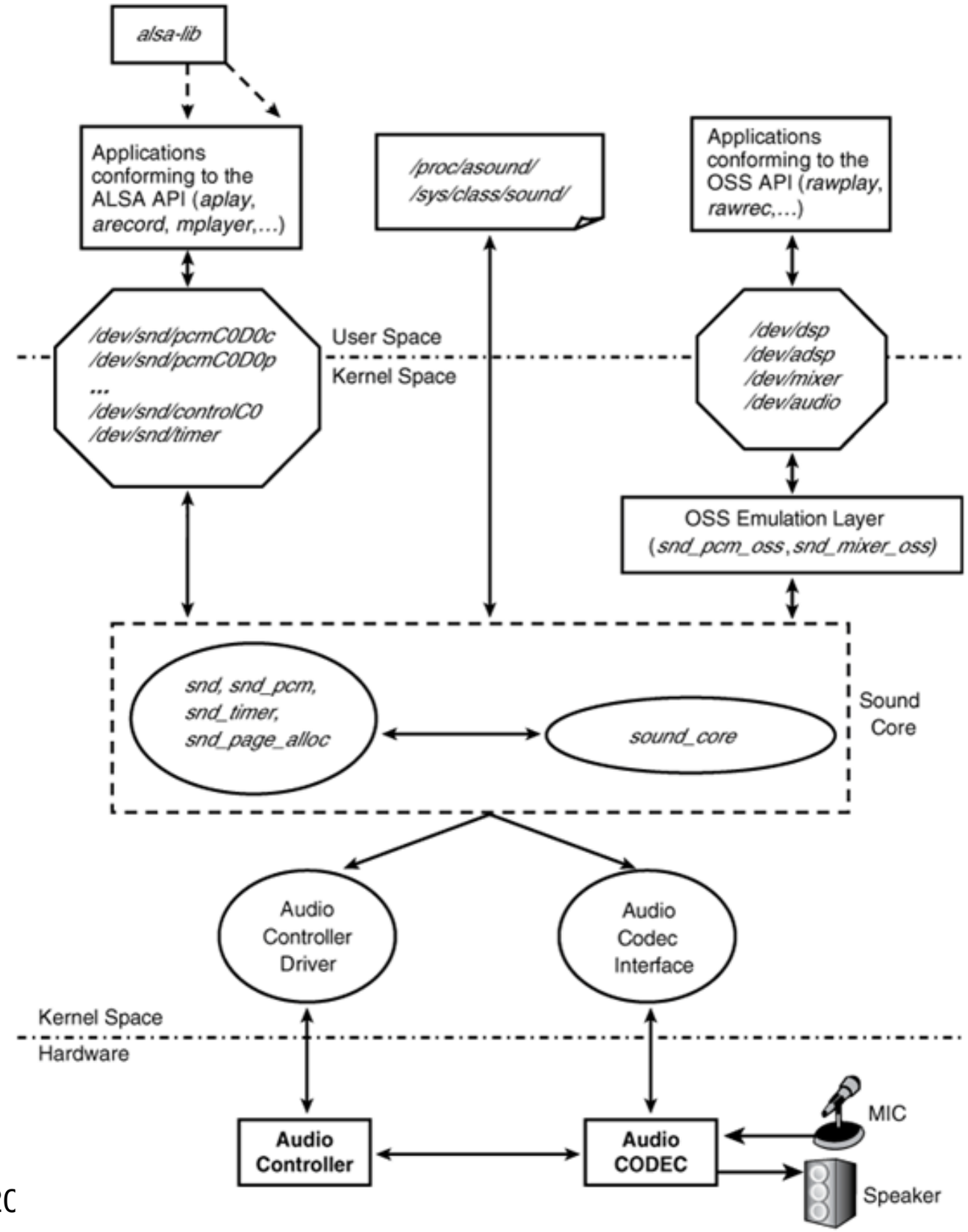


这是另一个ASLA架构图。与上一个图大同小异。从内核driver层、lib层到应用层勾画出了彼此之间的关系。

图中左下角OSS相关部分是为了兼容OSS驱动模型而存在的。不是本实践的相关部分。

2014-1-2

一、ALSA架构简介



左图是从代码的角度体现了alsa-lib和alsa-driver及hardware的交互关系。用户层的alsa-lib通过操作alsa-driver创建的设备文件/dev/snd/pcmC0D0p等对内核层进行访问。内核层的alsa-drivier驱动再经由sound core对硬件声卡芯片进行访问。从而实现了app alsa-lib alsa-driver hardware的操作。

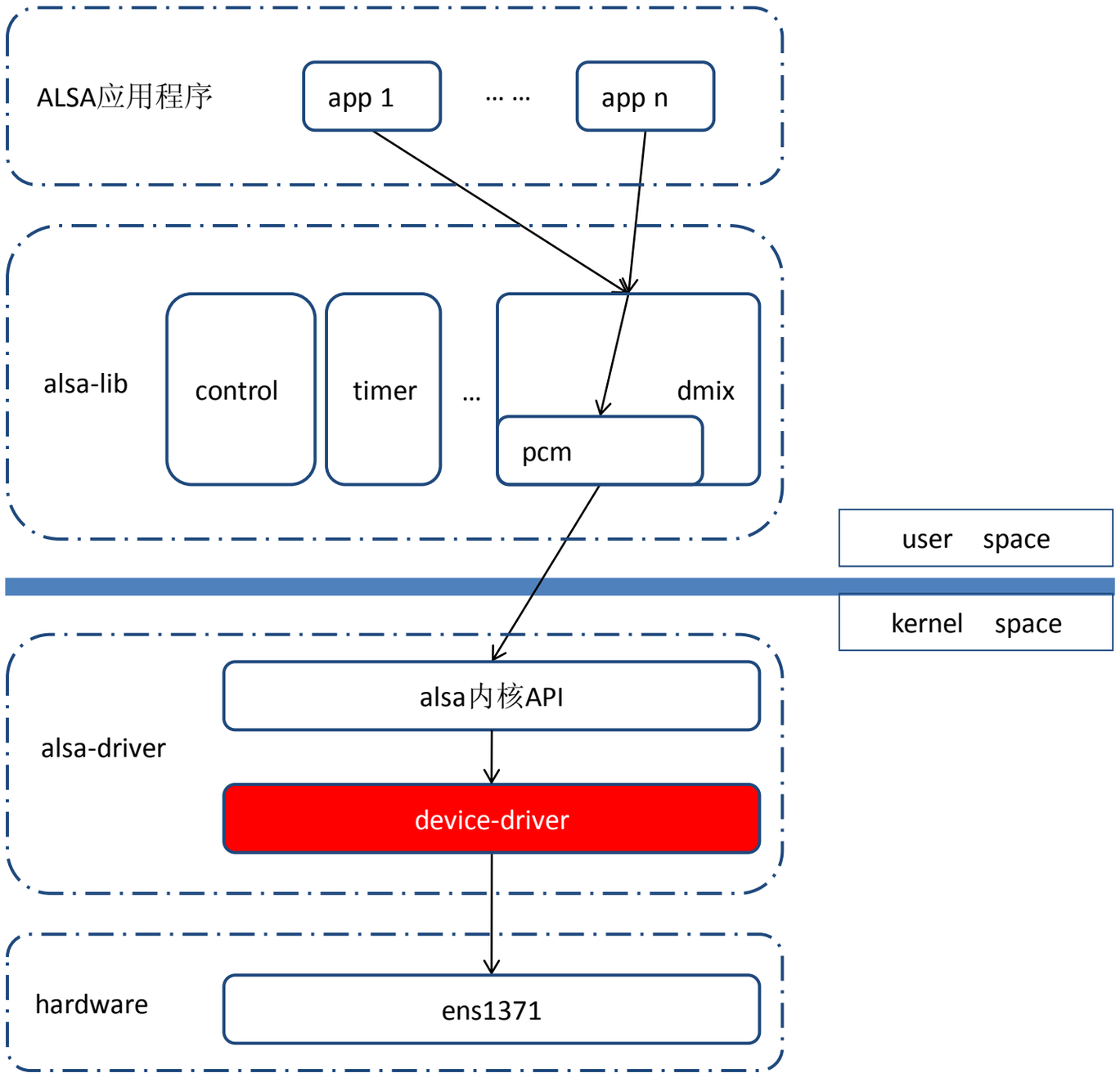
图中右上角OSS相关部分是为了兼容OSS驱动模型而存在的。不是本实践的相关部分。

目 录

2014-1-2

二、最佳实践的目标、目的、方法

ALSA整体架构



在alsa的整体架构中，alsa-driver的硬件操控层，负责实现硬件操纵访问，这也是声卡驱动程序中用户需实现的主要部分。

我们要实现的是针对特定声卡芯片ens1371的控制在这里把它叫做device-driver(设备驱动)， alsa-driver的其它部分叫做alsa-driver(alsa驱动)。

目标：	实现ens1371芯片的最小化驱动。 ***包括insmod，remod，以及playback 和stop 的过程。使ens1371可以正常加载、卸载、播放及停止。其他部分如录音(capture)等，不考虑。***
目的：	通过实现ens1371芯片的最小化驱动代码，理解ALSA架构的声卡驱动程序的开发。
方法：	~~最终目标是从ens1371的核心代码拷贝实现~~

目 录

2014-1-2

三、具体步骤

1. 准备工作。

(1) 使用alsa自带的aplay工具或其它应用程序播放音频，测试开发环境是否能够正常播放音频。如果不能播放，可以用alsamixer等工具查看音量是否被静音了。总之，如果正常环境如果都不能播放音频，那我们后续的工作也就不能检验正确与否了。

aplay工具shell中命令如下：

apt-get install alsa-utils //初次使用安装alsa-utils工具，以后直接执行下一步就可以了。

aplay /opt/test_files/pcm.wav //aplay后面的文件路径根据自己的实际情况填写。

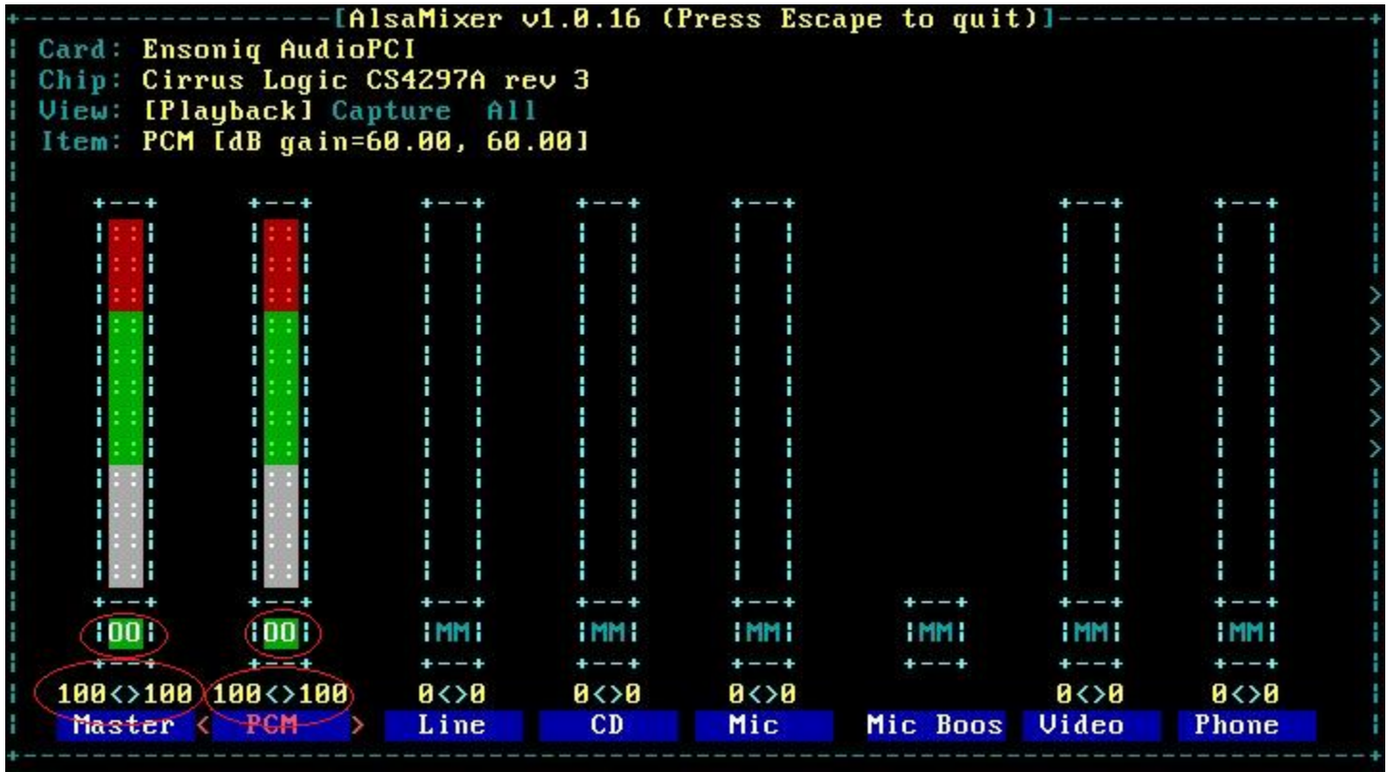
※红色文字表示shell中输入的命令。

alsamixer工具shell中命令如下：

apt-get install alsa-utils //初次使用安装alsa-utils工具，以后直接执行下一步就可以了。

alsamixer。

alsamixer中， 、 方向键可以调整焦点，↑、↓方向键可以调整音量，esc键退出。也可以用数字或+、-键调整音量。



(2) 检查使用的PC的声卡是否是ENS1371，shell中命令如下：

lspci |grep “1371”

ENS1371声卡会显示类似如下字样

#02:02.0 Multimedia audio controller: Ensoniq ES1371 [AudioPCI-97] (rev 02)

如果不是ENS1371，下文可作为参考根据自己的实际情况进行。

三、具体步骤

1. 准备工作。

(2) 新建工作目录如alsa-ens1371-dev-driver，在工作目录中新建文件ens1371-playback.c。

(3) 制作makefile。新建文件Makefile。参照《Linux 基础培训(2)-驱动开发最佳实践-1.pptx》第5页编写Makefile。Makefile内容如下：

```
ifneq ($(KERNELRELEASE),)
obj-m := ens1371-playback.o
else

KBUILD := /lib/modules/`uname -r`/build
modules:
		make -C $(KBUILD) M=$(shell pwd) modules
clean:
		rm -f *.o *~ core .depend .*.cmd *.ko *.mod.c *.order *.symvers
		rm -rf .tmp_versions
endif
```

2. 查看内核linux-source-2.6.26/sound/pci/ens1371.c的代码，内容如下：

```
#define CHIP1371
#include "ens1370.c“
```

即在定义宏CHIP1371的基础上，使用ens1370.c。

三、具体步骤

3. 抽取ens1370.c（位于linux-source-2.6.26/sound/pci）中Linux通用驱动程序核心，添加到ens1371-playback.c中并make，测试。可以先添加框架，再向框架中逐步添加函数。可以打印适当的log信息，不会影响ens1371的流程。Linux通用驱动程序代码如下：

```
#define CHIP1371
#include <linux/moduleparam.h>

#define ENS1371_DEBUG
#ifdef ENS1371_DEBUG
#define FUNC_LOG()  printk(KERN_ERR "FUNC_LOG: [%d][%s()]\n", __LINE__, __FUNCTION__)
#endif

static int __init alsa_card_ens137x_init(void)
{
    FUNC_LOG();
}
static void __exit alsa_card_ens137x_exit(void)
{
    FUNC_LOG();
}
module_init(alsa_card_ens137x_init)
module_exit(alsa_card_ens137x_exit)
```

4.根据ens1370.c的代码，向init和exit中添加实现函数。根据调用关系实现一个被调用函数的最小化定义（如下页图中的snd_audiopci_probe()和snd_audiopci_remove()），以填充调用函数，并将无关部分注释掉，减轻代码量。注意注释无关部分时需要注意，有些不能确定有没有用，可以先留着，另外，除了函数外，其他部分的最好用注释，不要直接删除，以便在后面检查是否删除了有用的部分。

定义最小化的函数后，代码是可以编译，并可以加载到内核的，再逐步添写被调用函数，以此类推，直到所有的调用关系全部被添加进代码。如向init和exit中添加实现函数：

三、具体步骤

```
#define CHIP1371

#include <linux/moduleparam.h>
#include <linux/pci.h>

#define ENS1371_DEBUG
#ifdef ENS1371_DEBUG
#define FUNC_LOG() printk(KERN_ERR "FUNC_LOG: [%d:][%s()]\n", __LINE__, __FUNCTION__)
#endif

#define DRIVER_NAME "ENS1371“
static struct pci_device_id snd_audiopci_ids[] = {
//#ifdef CHIP1370
//          { 0x1274, 0x5000, PCI_ANY_ID,  PCI_ANY_ID,  0, 0, 0, },              /* ES1370 */
//#endif
#ifdef CHIP1371
          { 0x1274, 0x1371, PCI_ANY_ID,  PCI_ANY_ID,  0, 0, 0, },              /* ES1371 */
          { 0x1274, 0x5880, PCI_ANY_ID,  PCI_ANY_ID,  0, 0, 0, },              /* ES1373 - CT5880 */
          { 0x1102, 0x8938, PCI_ANY_ID,  PCI_ANY_ID,  0, 0, 0, },              /* Ectiva EV1938 */
#endif
          { 0, }
};

static int __devinit snd_audiopci_probe(struct pci_dev *pci,  const struct pci_device_id *pci_id)
{
          FUNC_LOG();
          return 0;
}
static void __devexit snd_audiopci_remove(struct pci_dev *pci)
{
          FUNC_LOG();
}

static struct pci_driver driver = {
          .name = DRIVER_NAME,
          .id_table = snd_audiopci_ids,
          .probe = snd_audiopci_probe,
          .remove = __devexit_p(snd_audiopci_remove),
#ifdef CONFIG_PM
          //.suspend = snd_ensoniq_suspend,
          //.resume = snd_ensoniq_resume,
#endif
};

static int __init alsa_card_ens137x_init(void)
{
          FUNC_LOG();
          return pci_register_driver(&driver);
}
static void __exit alsa_card_ens137x_exit(void)
{
          FUNC_LOG();
          pci_unregister_driver(&driver);
}

module_init(alsa_card_ens137x_init)
module_exit(alsa_card_ens137x_exit)
```

三、具体步骤

5. make、insmod、rmmod 驱动模块示例：

```
db55:/mnt/share/alsa-ens1371-dev-driver# make
make: Warning: File `Makefile' has modification time 1.3e+04 s in the future
make -C /lib/modules/`uname -r`/build M=/mnt/share/alsa-ens1371-dev-driver modul
es
make[1]: Entering directory `/usr/src/linux-source-2.6.26'
make[2]: Warning: File `/mnt/share/alsa-ens1371-dev-driver/Makefile' has modific
ation time 1.3e+04 s in the future
  CC [M]  /mnt/share/alsa-ens1371-dev-driver/ens1371-playback.o
make[2]: warning:  Clock skew detected.  Your build may be incomplete.
  Building modules, stage 2.
make[2]: Warning: File `/mnt/share/alsa-ens1371-dev-driver/Makefile' has modific
ation time 1.3e+04 s in the future
  MODPOST 1 modules
  CC      /mnt/share/alsa-ens1371-dev-driver/ens1371-playback.mod.o
  LD [M]  /mnt/share/alsa-ens1371-dev-driver/ens1371-playback.ko
make[2]: warning:  Clock skew detected.  Your build may be incomplete.
make[1]: Leaving directory `/usr/src/linux-source-2.6.26'
make: warning:  Clock skew detected.  Your build may be incomplete.
db55:/mnt/share/alsa-ens1371-dev-driver#
db55:/mnt/share/alsa-ens1371-dev-driver#
db55:/mnt/share/alsa-ens1371-dev-driver# lsmod |grep "ens1371"
snd_ens1371            19236  0
gameport               10828  1 snd_ens1371
snd_rawmidi            18944  2 snd_ens1371,snd_seq_midi
snd_ac97_codec         91396  1 snd_ens1371
snd_pcm                64068  2 snd_ens1371,snd_ac97_codec
snd                    46648  7 snd_ens1371,snd_rawmidi,snd_ac97_codec,snd_pcm,s
nd_seq,snd_timer,snd_seq_device
db55:/mnt/share/alsa-ens1371-dev-driver# rmmod snd_ens1371
db55:/mnt/share/alsa-ens1371-dev-driver# lsmod |grep "ens1371"
db55:/mnt/share/alsa-ens1371-dev-driver#
db55:/mnt/share/alsa-ens1371-dev-driver# insmod ens1371-playback.ko
[ 3819.893531] FUNC_LOG: [53:][alsa_card_ens137x_init()]
[ 3819.893799] FUNC_LOG: [30:][snd_audiopci_probe()]
db55:/mnt/share/alsa-ens1371-dev-driver# lsmod |grep "ens1371"
ens1371_playback        2144  0
db55:/mnt/share/alsa-ens1371-dev-driver# rmmod ens1371-playback
[ 3848.189611] FUNC_LOG: [59:][alsa_card_ens137x_exit()]
[ 3848.189839] FUNC_LOG: [36:][snd_audiopci_remove()]
db55:/mnt/share/alsa-ens1371-dev-driver# _
```

※注意insmod ens1371-playback.ko前，要先卸载系统中原有的ens1371模块——snd_ens1371。否则，在定义了DRIVER_NAME后，insmod时就会出错。

图中，红色是shell中输入的命令，黄色是系统打印的信息，而绿色是我们的模块打印的log。

另外，驱动模块的单步调试可以参照《Linux 基础培训(2)-驱动开发最佳实践-1.pptx》。

三、具体步骤

6. 仿照snd_audiopci_probe()和 snd_audiopci_remove()的填充与alsa_card_ens137x_init()和alsa_card_ens137x_exit()类似，如此一级一级的填充下去，直至模块可以播放出声音。 对于snd_audiopci_probe()这样比较复杂的函数，要一行一行的添加，边添加边调试。如遇到变量等应该在前面函数中赋值而还没有实现的部分，可以先用if宏等方法暂时注释掉，如下页代码中的红色部分。添加的代码如下：

```
#define CHIP1371

#include <linux/pci.h>
#include <linux/moduleparam.h>

#include <sound/core.h>
#include <sound/initval.h>

#define ENS1371_DEBUG
#ifdef ENS1371_DEBUG
#define FUNC_LOG() printk(KERN_ERR "FUNC_LOG: [%d:][%s()]\n", __LINE__, __FUNCTION__)
#endif

#define DRIVER_NAME "ENS1371“
static struct pci_device_id snd_audiopci_ids[] = {
        ……
};

static int index[SNDRV_CARDS] = SNDRV_DEFAULT_IDX;          /* Index 0-MAX */
static char *id[SNDRV_CARDS] = SNDRV_DEFAULT_STR;           /* ID for this card */
static int enable[SNDRV_CARDS] = SNDRV_DEFAULT_ENABLE_PNP;  /* Enable switches */
static int spdif[SNDRV_CARDS];
static int lineio[SNDRV_CARDS];

struct ensoniq {
……
};

MODULE_DEVICE_TABLE(pci, snd_audiopci_ids);

static int __devinit snd_ensoniq_1371_mixer(struct ensoniq *ensoniq,    int has_spdif, int has_line)
{
        FUNC_LOG();
        return(0);
}

static int __devinit snd_ensoniq_pcm(struct ensoniq * ensoniq, int device, struct snd_pcm ** rpcm)
{
        FUNC_LOG();
        return 0;
}

static int __devinit snd_ensoniq_create(struct snd_card *card, struct pci_dev *pci, struct ensoniq ** rensoniq)
{
        FUNC_LOG();
        return 0;
}
```

三、具体步骤

6. 代码续：

```
static int __devinit snd_audiopci_probe(struct pci_dev *pci,  const struct pci_device_id *pci_id)
{
        static int dev;
        struct snd_card *card;
        struct ensoniq *ensoniq;
        int err, pcm_devs[2];

        FUNC_LOG();

        if (dev >= SNDRV_CARDS)
                return -ENODEV;
        if (!enable[dev]) {
                dev++;
                return -ENOENT;
        }

        card = snd_card_new(index[dev], id[dev], THIS_MODULE, 0);
        if (card == NULL)
                return -ENOMEM;

        if ((err = snd_ensoniq_create(card, pci, &ensoniq)) < 0) {
                snd_card_free(card);
                return err;
        }
        card->private_data = ensoniq;

        pcm_devs[0] = 0; pcm_devs[1] = 1;
//#ifdef CHIP1370
//      if ((err = snd_ensoniq_1370_mixer(ensoniq)) < 0) {
//              snd_card_free(card);
//              return err;
//      }
//#endif
//#ifdef CHIP1371
        if ((err = snd_ensoniq_1371_mixer(ensoniq, spdif[dev], lineio[dev])) < 0) {
                snd_card_free(card);
                return err;
        }
//#endif
        if ((err = snd_ensoniq_pcm(ensoniq, 0, NULL)) < 0) {
                snd_card_free(card);
                return err;
        }
//      if ((err = snd_ensoniq_pcm2(ensoniq, 1, NULL)) < 0) {
//              snd_card_free(card);
//              return err;
//      }
//      if ((err = snd_ensoniq_midi(ensoniq, 0, NULL)) < 0) {
//              snd_card_free(card);
//              return err;
//      }

//      snd_ensoniq_create_gameport(ensoniq, dev);
        strcpy(card->driver, DRIVER_NAME);

        strcpy(card->shortname, "Ensoniq AudioPCI");
#ifdef ENS1371_RESERVED
        sprintf(card->longname, "%s %s at 0x%lx, irq %i",
                        card->shortname,
                        card->driver,
                        ensoniq->port,
                        ensoniq->irq);
#endif
        if ((err = snd_card_register(card)) < 0) {
                snd_card_free(card);
                return err;
        }

        pci_set_drvdata(pci, card);
        dev++;
        return 0;

}

static void __devexit snd_audiopci_remove(struct pci_dev *pci)
{
        FUNC_LOG();
        snd_card_free(pci_get_drvdata(pci));
        pci_set_drvdata(pci, NULL);
}

static struct pci_driver driver = {
.name = DRIVER_NAME,
.id_table = snd_audiopci_ids,
.probe = snd_audiopci_probe,
.remove = __devexit_p(snd_audiopci_remove),
#ifdef CONFIG_PM
//.suspend = snd_ensoniq_suspend,
//.resume = snd_ensoniq_resume,
#endif
};


static int __init alsa_card_ens137x_init(void)
{
        FUNC_LOG();
        return pci_register_driver(&driver);
}
static void __exit alsa_card_ens137x_exit(void)
{
        FUNC_LOG();
        pci_unregister_driver(&driver);
}

module_init(alsa_card_ens137x_init)
module_exit(alsa_card_ens137x_exit)
```

三、具体步骤

7. 如此类推再实现snd_ensoniq_create()、snd_ensoniq_pcm()、snd_ensoniq_1371_mixer()函数，再此不再详述。直至播放出声音，并且能够正确停止，加载、卸载模块，即最小化的驱动程序能够正常工作，也就完成了我们的目标。

2014-1-2

目 录

2014-1-2

四、总结

ens1371的声卡核心驱动可以分为三个主要部分：

(1) 模块insmod时初始化和rmmod时退出部分：

初始化：

alsa_card_ens137x_init()

snd_audiopci_probe()

snd_ensoniq_create()

snd_ensoniq_chip_init()

snd_ensoniq_1371_mixer()

snd_ensoniq_pcm()

退出：

alsa_card_ens137x_exit()

snd_audiopci_remove()

snd_ensoniq_free()

snd_ensoniq_mixer_free_ac97()

(2) 播放以及停止部分：

播放：

snd_ensoniq_playback1_open()

snd_ensoniq_hw_params()

snd_ensoniq_playback1_prepare()

snd_ensoniq_trigger()

停止：

snd_ensoniq_trigger()

snd_ensoniq_hw_free()

snd_ensoniq_playback1_close()

(3) 中断触发部分：

snd_audiopci_interrupt()

snd_ensoniq_playback1_pointer()

其中有的部分又包含若干子部分。硬件无关部分会使用alsa-driver中的函数。

2014-1-2