Melis RTOS 音频
开发指南

版本号: 1.0
发布日期: 2020-10-22

版本历史

版本号	日期	制/修订人	内容描述
1.0	2020.08.13	AW1612	Melis_RTOS_ 音频 _ 开发指南

目　录

1 概述

1.1 编写目的

介绍 Melis RTOS 平台音频模块的使用方法、AudioCodec 的支持、应用接口、指导如何配置、使用、测试等。

1.2 适用范围

Allwinner 软件平台 Melis RTOS。

Allwinner 硬件平台 V833。

1.3 相关人员

Melis RTOS 下进行音频模块开发的工程师。

1.4 相关术语

表 1-1: audio 相关术语

术语	解释说明
ALSA	Advanced Linux Sound Architecture
DMA	直接内存存取, 指数据不经 cpu, 直接在设备和内存, 内存和内存, 设备和设备之间传输
ASoc	ALSA System on Chip
样本长度 sample	样本是记录音频数据最基本的单位，常使用 16 位
通道数 channel	该参数为 1 表示单声道，2 则是立体声
帧 frame	帧记录了一个声音单元，其长度为样本长度与通道数的乘积
采样率 rate	每秒钟采样次数，该次数是针对帧而言
周期 period	音频设备一次处理所需要的帧数，对于音频设备的数据访问以及音频数据的存储，都是以此为单位

术语	解释说明
audiocodec	芯片内置音频接口
daudio	数字音频接口，可配置成 I2S/PCM 标准音频接口

2 模块介绍

2.1 驱动框架

RTOS 上面的音频驱动框架与 Linux 上 ASOC 框架的思想比较类似，分为 codec,platform 模型。core 核心层实现具体的 codec 驱动加载、platform 驱动的选择、数据的搬运、dma 指针的更新等，能够将 codec 驱动与 SoC CPU 解耦合, 方便添加外挂 codec，加快适配新的 SoC。

音频驱动的大致框架如下：

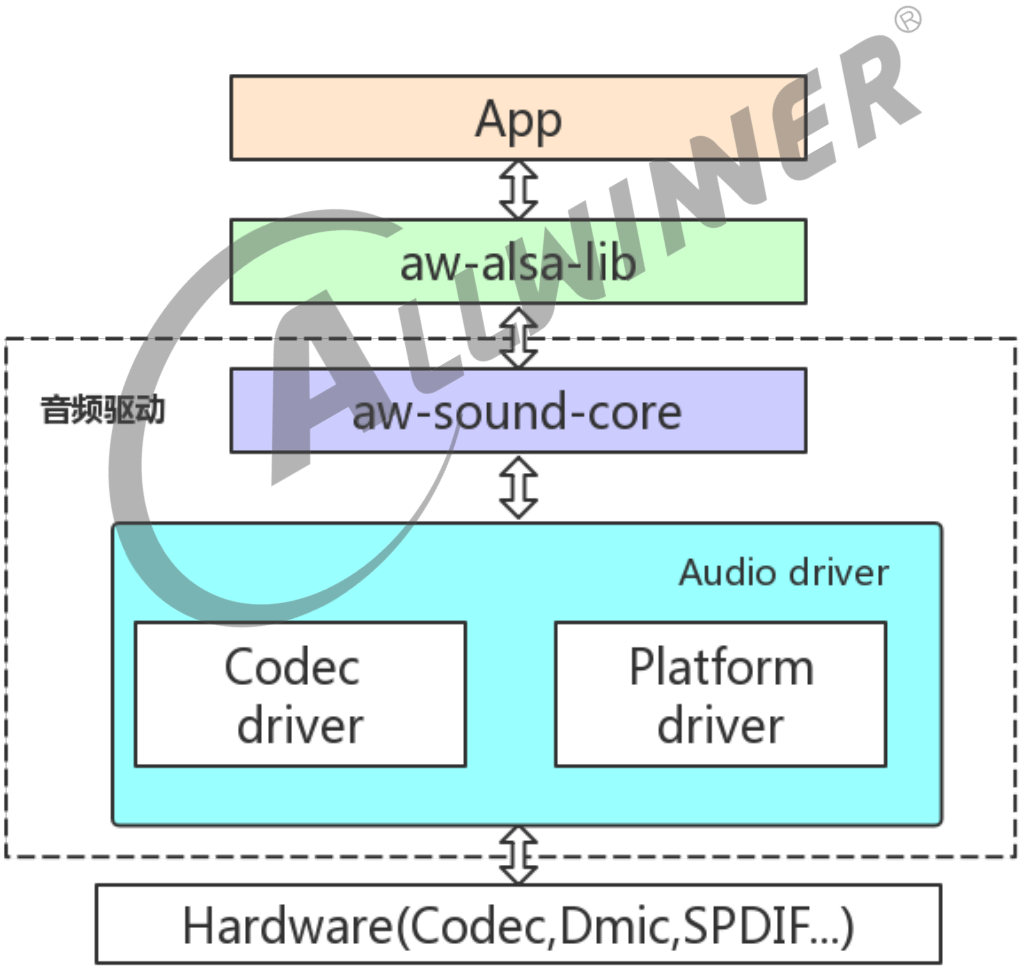

图 2-1: V833 音频驱动

2.2 V833 音频模块介绍

2.2.1 硬件资源

V833 RTOS 上目前支持 audiocodec,daudio。

2.2.2 时钟源

Melis V833 中音频模块的时钟源来自 pll_audio。

pll_audio 可以输出 24.576M 或者 22.5792M 的时钟，分别支持 48k 系列和 44.1k 系列的播放录音。

2.2.3 代码结构

```
drivers/hal/sound/
├── component/              //aw-alsa-utils（aplay、arecord） 、 aw-alsa-lib代码
├── card_default.c                  // 声卡加载配置文件
├── core/                           // 音频驱动核心层代码
├── sunxi-codecs/                     // codec驱动代码
│   ├── sun8iw19-codec.c        // V833 audiocodec驱动
└── sunxi-i2s/                       // platform驱动代码
    ├── dummy_codec.c               // dummy codec,用于加载没有实际硬件codec的声卡
    └── sun8iw19-daudio.c             // Allwinner i2s驱动
```

2.2.4 Audiocodec

硬件特性

- 一路 DAC
  - 支持 16bit,24bit 采样精度
  - 支持 8KHz~192KHz 采样率
- 一路 ADC
  - 支持 16bit,24bit 采样精度
  - 支持 8KHz~48KHz 采样率
- 一路模拟输出：一路差分输出 lineoutP/N 或单端 lineout 输出
- 二路模拟输入：phonein、linein

- 支持同时 playback 和 record(全双工模式)
- 支持 ADC、DAC 的 DRC 功能
- DAC FIFO 长度 128*24bits, ADC FIFO 长度 128*24bits

2.2.5 Daudio

硬件特性

- 2 套 I2S（I2S0 数据采集，I2S1 扩展）
- 支持主从模式
- 支持 Left-justified,Right-justified,Standar mode I2S,PCM mode
- 支持 PCM 模式、支持 8-bit u-Law、8-bit a-Law
- 支持 mono 和 stereo 模式，最高支持 16 通道
- 支持同时 palyback 和 record(全双工模式)
- 支持 8~192KHz 采样率
- 支持 8,16,24,32bit 采样精度
- 2 套 I2S 支持两个 data_out 或两个 data_in 或者一个 data_out 一个 data_in

3 音频接口库

Melis RTOS 提供了 aw-alsa-lib 音频接口去操作声卡，接口定义及使用与 alsa-lib 比较类似，除了常用的 pcm 操作接口，control 接口外，还支持 alsa plugin。具体插件的配置方法请查看 4.4 章节。

接口头文件在 ekernel/drivers/include/hal/aw-alsa-lib 目录中:

```
├── control.h      // 控件操作相关的接口
└── pcm.h          // pcm设备操作相关的接口
```

下面分别简单介绍下这两类接口。

3.1 pcm 设备操作接口

声卡打开、关闭接口:

```
打开声卡得到pcm设备。
int snd_pcm_open(snd_pcm_t **pcm,
        const char *name,
        snd_pcm_stream_t stream,
        int mode)
参数:
    pcm:    返回pcm操作句柄
    name:   pcm名称,即alsa plugin配置文件中定义的名称
    stream: 数据流, playback或者capture
    mode:   (unused)
返回值:
    成功打开则返回0,否则返回error code
例子:
    参考ekernel/drivers/hal/source/sound/component/aw-alsa-utils/aplay.c
```

```
关闭pcm设备
int snd_pcm_close(snd_pcm_t *pcm)
参数:
    pcm:    pcm操作句柄
返回值:
    成功打开则返回0,否则返回error code
例子:
    参考ekernel/drivers/hal/source/sound/component/aw-alsa-utils/aplay.c
```

hw_params 设置相关接口:

```
创建hw_params的宏:
snd_pcm_hw_params_alloca()
```

```
填充hw_param配置
int snd_pcm_hw_params_any(snd_pcm_t *pcm, snd_pcm_hw_params_t *params)
参数:
    pcm:    pcm操作句柄
    params: hw_param配置
返回值:
    成功打开则返回0,否则返回error code
```

```
设置采样格式
int snd_pcm_hw_params_set_format(snd_pcm_t *pcm, snd_pcm_hw_params_t *params,
    snd_pcm_format_t format)
参数:
    pcm:    pcm操作句柄
    params: hw_param配置
    format: 采样格式
返回值:
    成功打开则返回0,否则返回error code
```

```
设置通道数
int snd_pcm_hw_params_set_channels(snd_pcm_t *pcm, snd_pcm_hw_params_t *params, unsigned
    int val)
参数:
    pcm:    pcm操作句柄
    params: hw_param配置
    val:    通道数
返回值:
    成功打开则返回0,否则返回error code
```

```
设置采样率
int snd_pcm_hw_params_set_rate(snd_pcm_t *pcm, snd_pcm_hw_params_t *params, unsigned int
    val, int *dir)
参数:
    pcm:    pcm操作句柄
    params: hw_param配置
    val:    采样率
    dir:    (unused)
返回值:
    成功打开则返回0,否则返回error code
```

```
设置period_size
int snd_pcm_hw_params_set_period_size(snd_pcm_t *pcm, snd_pcm_hw_params_t *params,
    snd_pcm_uframes_t val, int *dir)
参数:
    pcm:    pcm操作句柄
    params: hw_param配置
    val:    period_size
    dir:    (unused)
返回值:
    成功打开则返回0,否则返回error code
```

```
设置buffer_size
int snd_pcm_hw_params_set_buffer_size(snd_pcm_t *pcm, snd_pcm_hw_params_t *params,
    snd_pcm_uframes_t val)
参数:
    pcm:    pcm操作句柄
    params: hw_param配置
    val:    buffer_size
    dir:    (unused)
返回值:
    成功打开则返回0,否则返回error code
```

```
安装hw_params到声卡中
int snd_pcm_hw_params(snd_pcm_t *pcm, snd_pcm_hw_params_t *params)
参数:
    pcm:    pcm操作句柄
    params: hw_param配置
返回值:
    成功打开则返回0,否则返回error code
```

sw_params 设置相关接口:

```
创建sw_params的宏:
snd_pcm_sw_params_alloca()
```

```
填充sw_param配置
int snd_pcm_sw_params_current(snd_pcm_t *pcm, snd_pcm_sw_params_t *params)
参数:
    pcm:    pcm操作句柄
    params: sw_param配置
返回值:
    成功打开则返回0,否则返回error code
```

```
设置start_threshold
```

```
对于playback,决定写入多少帧数据后开始播放(建议start_threshold设定为buffer size)。
对于capture,要采集的帧数大于start_threshold后才开始录音(建议start_threshold设定为1)。
int snd_pcm_sw_params_set_start_threshold(snd_pcm_t *pcm, snd_pcm_sw_params_t *params,
    snd_pcm_uframes_t val)
参数:
    pcm:    pcm操作句柄
    params: sw_param配置
    val:    start threshold,单位为帧
返回值:
    成功打开则返回0,否则返回error code
```

```
设置stop_threshold
对于playback,可写入pcm数据的空间大于等于stop_threshold时，触发underrun(建议stop_threshold设定为
    buffer size)。
对于capture,可读出来的pcm数据量大于等于stop_threshold时,触发overrun(建议stop_threshold设定为buffer
    size)。
int snd_pcm_sw_params_set_stop_threshold(snd_pcm_t *pcm, snd_pcm_sw_params_t *params,
    snd_pcm_uframes_t val)
参数:
    pcm:    pcm操作句柄
    params: sw_param配置
    val:    stop threshold,单位为帧
返回值:
    成功打开则返回0,否则返回error code
```

```
设置avail_min(建议avail_min设置为period size)
对于playback,可写入pcm数据的空间大于avail_min时，唤醒应用write任务将pcm数据写入。
对于capture,可读出来的pcm数据量大于avail_min时，唤醒应用read任务将pcm数据读到应用buffer中。
int snd_pcm_sw_params_set_avail_min(snd_pcm_t *pcm, snd_pcm_sw_params_t *params,
    snd_pcm_uframes_t val)
参数:
    pcm:    pcm操作句柄
    params: sw_param配置
    val:    avail_min,单位为帧
返回值:
    成功打开则返回0,否则返回error code
```

```
安装sw_params到声卡中
int snd_pcm_sw_params(snd_pcm_t *pcm, snd_pcm_sw_params_t *params)
参数:
    pcm:    pcm操作句柄
    params: sw_param配置
返回值:
    成功打开则返回0,否则返回error code
```

pcm 数据读写接口:

```
写入size帧pcm数据
snd_pcm_sframes_t snd_pcm_writei(snd_pcm_t *pcm, const void *buffer, snd_pcm_uframes_t size
```

```
    )
参数:
    pcm:    pcm操作句柄
    buffer: pcm数据buffer地址
    size:   要写入的帧数
返回值:
    成功返回已写入的帧数,否则返回error code
    -EBADFD   表示pcm设备目前的state不允许当前操作
    -EPIPE    表示出现underrun
```

```
读出size帧pcm数据
snd_pcm_sframes_t snd_pcm_readi(snd_pcm_t *pcm, void *buffer, snd_pcm_uframes_t size)
参数:
    pcm:    pcm操作句柄
    buffer: pcm数据buffer地址
    size:   要读出来的帧数
返回值:
    成功返回已读出来的帧数,否则返回error code
    -EBADFD   表示pcm设备目前的state不允许当前操作
    -EPIPE    表示出现overrun
```

pcm 设备控制接口:

```
将pcm设备设置为prepare状态
int snd_pcm_prepare(snd_pcm_t *pcm)
参数:
    pcm:    pcm操作句柄
返回值:
    成功则返回0,否则返回error code
```

```
停止pcm设备,并丢掉pending frames
int snd_pcm_drop(snd_pcm_t *pcm)
参数:
    pcm:    pcm操作句柄
返回值:
    成功则返回0,否则返回error code
```

```
停止pcm设备,并会把pending frames耗尽
int snd_pcm_drain(snd_pcm_t *pcm)
参数:
    pcm:    pcm操作句柄
返回值:
    成功则返回0,否则返回error code
```

```
暂停/恢复pcm设备
int snd_pcm_pause(snd_pcm_t *pcm, int enable)
```

```
参数:
    pcm:    pcm操作句柄
    enable: 0-恢复,1-暂停
返回值:
    成功则返回0,否则返回error code
```

```
从异常状态(xrun)恢复
int snd_pcm_recover(snd_pcm_t *pcm, int err, int silent)
参数:
    pcm:    pcm操作句柄
    err:    error code
    silent: 0-输出xrun等打印, 1-不打印任何信息
    enable: -恢复,1-暂停
返回值:
    成功则返回0,否则返回error code
```

```
得到当前pcm状态
snd_pcm_state_t snd_pcm_state(snd_pcm_t *pcm)
参数:
    pcm:    pcm操作句柄
返回值:
    返回当前pcm状态,如
    SND_PCM_STATE_OPEN
    SND_PCM_STATE_SETUP
    SND_PCM_STATE_PREPARED
    SND_PCM_STATE_RUNNING
```

```
对于播放，可以得到当前剩余播放数据量,单位为帧。
对于录音，可以得到当前剩余可录制数据量,单位为帧。
int snd_pcm_delay(snd_pcm_t *pcm, snd_pcm_sframes_t *delayp)
参数:
    pcm:    pcm操作句柄
    delayp: 返回delay值
返回值:
    成功则返回0,否则返回error code
```

3.2 control 控件操作接口

```
获取指定声卡的控件个数
int snd_ctl_num(const char *name)
参数:
    name:    声卡名称
返回值:
    成功则返回控件个数,否则返回error code
```

```
获取指定声卡、指定控件名称的信息
int snd_ctl_get(const char *name, const char *elem, snd_ctl_info_t *info)
参数:
    name:    声卡名称
    elem:    控件名称
    info:    返回的控件信息
返回值:
    成功则返回0,否则返回error code
```

```
获取指定声卡、指定控件id的信息
int snd_ctl_get_bynum(const char *name, const unsigned int elem_num, snd_ctl_info_t *info)
参数:
    name:    声卡名称
    elem_num:控件id
    info:    返回的控件信息
返回值:
    成功则返回0,否则返回error code
```

```
设定指定声卡、指定控件名称的值
int snd_ctl_set(const char *name, const char *elem, unsigned int val)
参数:
    name:    声卡名称
    elem:    控件名称
    val:     设定的值
返回值:
    成功则返回0,否则返回error code
```

```
设定指定声卡、指定控件id的值
int snd_ctl_set_bynum(const char *name, const unsigned int elem_num, unsigned int val)
参数:
    name:    声卡名称
    elem_num:控件id
    val:     设定的值
返回值:
    成功则返回0,否则返回error code
```

4 配置方法

4.1 audiocodec 配置

audiocodec 配置位于代码中, 路径如下：

```
ekernel/drivers/hal/source/sound/sunxi-codec/sun8iw19-codec.c

在该结构中定义：

static struct sunxi_codec_param default_param = {
    .digital_vol    = 0x0,
    .lineout_vol    = 0x1f,
    .mic1gain   = 0x1f,
    .lineingain = 0x0,
    .gpio_spk   = GPIOH(4),
    .pa_msleep_time = 160,
    .pa_level   = 1,
    .adcdrc_cfg     = 0,
    .adchpf_cfg     = 1,
    .dacdrc_cfg     = 0,
    .dachpf_cfg     = 0,
};
```

表 4-1: codec 配置

codec 配置	codec 配置说明
digital_vol	初始化 digital volume，可设定范围 0~0x3f, 表示 0~-73.08dB, -1.16dB/step
lineout_vol	lineout volume，可设定范围 0~0x1f, 表示-43.5dB~0dB, 1.5dB/step
mic1gain	mic1 增益，可设定范围 0~0x1f,0:0dB, 1~0x3:6dB, 0x4 ~0x1f:9 ~36dB, 1dB/step, 一般设置 0x13, 即 24dB
lineingain	LINEINL 增益 0:0dB, 1:6dB
gpio_spk	PA 使能引脚
pa_msleep_time	操作 PA 之后的延时时间 (用来避免 pop 音)
pa_level	PA 引脚使能方式。0: 低电平有效；1：高电平有效
adcdrc_cfg	是否使用 adcdrc. 0: 不使用；1：使用
adchpf_cfg	是否使用 adchpf. 0: 不使用；1：使用
dacdrc_cfg	是否使用 dacdrc. 0: 不使用；1：使用
dachpf_cfg	是否使用 dachpf. 0: 不使用；1：使用

4.2 i2s 配置

i2s 配置位于代码中，路径如下：

```
ekernel/drivers/hal/source/sound/sunxi-i2s/sun8iw19-daudio.c

在该结构中定义：

struct sunxi_daudio_param {
        uint8_t tdm_num;
        uint8_t clk_parent;
        uint8_t daudio_master;
        uint8_t audio_format;
        uint8_t signal_inversion;
        uint16_t pcm_lrck_period;
        uint8_t msb_lsb_first:1;
        uint8_t sign_extend:2;
        uint8_t tx_data_mode:2;
        uint8_t rx_data_mode:2;
        uint8_t slot_width_select;
        uint8_t frametype;
        uint8_t tdm_config;
        uint16_t mclk_div;
};
```

表 4-2: snddaudio 配置

snddaudio 配置	snddaudio 配置说明
tdm_num	表示第几组 i2s
clk_parent	1:use as clk parent, 0：not use
daudio_master	1: SND_SOC_DAIFMT_CBM_CFM(codec clk & FRM master), 即 daudio 接口作为 slave, codec 作为 master 2: SND_SOC_DAIFMT_CBS_CFM(codec clk slave & FRM master), 一般不用 3: SND_SOC_DAIFMT_CBM_CFS(codec clk master & frame slave), 一般不用 4: SND_SOC_DAIFMT_CBS_CFS(codec clk & FRM slave), 即 daudio 接口作为 master, codec 作为 slave
audio_format	1: SND_SOC_DAIFMT_I2S(standard i2s format) 2: SND_SOC_DAIFMT_RIGHT_J(right justfied format) 3: SND_SOC_DAIFMT_LEFT_J(left justfied format) 4: SND_SOC_DAIFMT_DSP_A(pcm. MSB is available on 2nd BCLK rising edge after LRC rising edge) 5: SND_SOC_DAIFMT_DSP_B(pcm. MSB is available on 1nd BCLK rising edge after LRC rising edge)

snddaudio 配置	snddaudio 配置说明
signal_inversion	1: SND_SOC_DAIFMT_NB_NF(normal bit clock + frame) 2: SND_SOC_DAIFMT_NB_IF(normal BCLK + inv FRM) 3: SND_SOC_DAIFMT_IB_NF(invert BCLK + nor FRM) 4: SND_SOC_DAIFMT_IB_IF(invert BCLK + FRM)
pcm_lrck_period	一般可配置 16/32/64/128/256 个 bclk
msb_lsb_first	0: msb first; 1: lsb first
tx_data_mode	0: 16bit linear PCM;1: reserved;2: 8bit u-law;3: 8bit a-law
rx_data_mode	0: 16bit linear PCM;1: reserved;2: 8bit u-law;3: 8bit a-law
slot_width_select	支持 8bit, 16bit, 32bit 宽度
frametype	0: short frame = 1 clock width; 1: long frame = 2 clock width
tdm_config	0: pcm mode; 1: i2s mode
mclk_div	0: not output(normal setting this); 1/2/4/6/8/12/16/24/32/48/64/96/128/176/192: 给外部 codec 提供时钟，频率是 pll_audio/mclk_div

4.3 声卡加载配置

声卡配置位于代码中, 路径如下:

```
ekernel/drivers/hal/source/sound/card_default.c

使用snd_card_register函数进行声卡的注册
int snd_card_register(const char *name, struct snd_codec *codec, int platform_type)

name:          表示声卡的名字，aw-alsa-lib中都需要通过该名字找到对应的声卡
codec:         codec结构体，根据实际使用的codec进行配置，如V833的audiocodec,使用sun8iw19_codec;
platform_type: 与linux中ASOC框架类似，也需要指定使用哪种类型的platform,如CPUDAI, DAUDIO等

举例:
注册V833 audiocodec声卡
snd_card_register("audiocodec", &sun8iw19_codec, SND_PLATFORM_TYPE_CPUDAI);

注册daudio0 声卡
snd_card_register("snddaudio0", &dummy_codec, SND_PLATFORM_TYPE_DAUDIO0);
```

4.4 aw-alsa-lib 插件配置

aw-alsa-lib 与 linux 上 alsa-lib 一样支持多种插件，插件的选择、使用，需要在代码中进行配置。

路径如下：

```
/ekernel/drivers/hal/source/sound/component/aw-alsa-lib/alsa_config.c
```

下面根据插件类型进行简单说明：

4.4.1 hw 插件

```
#define SND_PCM_HW_CONFIG(name, device) \
{ \
    .card_name  = #name,  \
    .device_num = device, \
}

#define DEFINE_SND_PCM_HW_CONFIG(name, device) \
    const snd_pcm_hw_config_t snd_##name##_hw_config = \
                SND_PCM_HW_CONFIG(name, device);

static DEFINE_SND_PCM_HW_CONFIG(audiocodec, 0);
static DEFINE_SND_PCM_HW_CONFIG(snddaudio0, 0);
static DEFINE_SND_PCM_HW_CONFIG(snddaudio1, 0);
static DEFINE_SND_PCM_HW_CONFIG(snddaudio2, 0);
static DEFINE_SND_PCM_HW_CONFIG(snddmic, 0);
static DEFINE_SND_PCM_HW_CONFIG(sndspdif, 0);

card_name: 声卡名称,需要与card_default.c中注册声卡时指定的名称一致
device_num: pcm设备序号，目前仅支持一个pcm设备，所以只能为0
```

4.4.2 softvol 插件

```
static const snd_pcm_softvol_config_t snd_pcm_softvol_config1 = {
    .type       = "softvol",
    .slave = {
        .pcm        = "PlaybackDmix",
    },
    .control = {
        .control_name   = "Soft Volume Control1",
        .card_name  = "audiocodec",
    },
```

```
    .min_dB         = -51.0,
    .max_dB         = 0.0,
    .resolution     = 256,
};

static const snd_pcm_softvol_config_t snd_pcm_softvol_config2 = {
    .type       = "softvol",
    .slave = {
        .pcm        = "PlaybackDmix",
    },
    .control = {
        .control_name   = "Soft Volume Control2",
        .card_name  = "audiocodec",
    },
    .min_dB         = -51.0,
    .max_dB         = 0.0,
    .resolution     = 100,
};


type:           插件类型softvol
slave:          pcm从设备的信息
  pcm:          pcm设备的名称,必须为hw类型
control:        控件信息
  control_name: 控件名称
  card_name:    声卡名称，指定控件位于哪个声卡中
min_dB:         最小衰减
max_dB:         最大衰减
resolution:     精度
```

注意，在第一次使用 softvol 插件进行播放时才会生成该控件。

4.4.3 dsnoop 插件

```
static const snd_pcm_dsnoop_config_t snd_pcm_dsnoop_ref_config = {
    .type       = "dsnoop",
    .ipc_key    = 1111,
    .slave = {
        .pcm        = "hw:audiocodec",
        .format     = SND_PCM_FORMAT_S16_LE,
        .rate       = 16000,
        .channels   = 1,
        .period_size    = 1024,
        .periods    = 8,
    },
};

type:           插件类型dsnoop
ipc_key:        需要进行同时录音的均要指定同一ipc_key
slave:          pcm从设备的信息
  pcm:          pcm设备的名称,必须为hw类型
  format:       采样格式
  rate:         采样率
  channels:     通道数
```

```
  period_size: period_size大小,决定中断触发频率
  periods:     periods大小,决定buffer size大小
```

4.4.4 dmix 插件

```
static const snd_pcm_dmix_config_t snd_pcm_dmix_config = {
    .type       = "dmix",
    .ipc_key    = 2222,
    .slave = {
        .pcm        = "hw:audiocodec",
        .format     = SND_PCM_FORMAT_S16_LE,
        .rate       = 16000,
        .channels   = 1,
        .period_size    = 1024,
        .periods    = 8,
    },
};

type:           插件类型dmix
ipc_key:        需要进行混音的均要指定同一ipc_key
slave:          pcm从设备的信息
  pcm:          pcm设备的名称,必须为hw类型
  format:       采样格式
  rate:         采样率
  channels:     通道数
  period_size:  period_size大小,决定中断触发频率
  periods:      periods大小,决定buffer size大小
```

4.4.5 asym 插件

```
static const snd_pcm_asym_config_t snd_pcm_asym_config = {
    .type       = "asym",
    .playback_pcm   = "PlaybackDmix",
    .capture_pcm    = "CaptureDsnoop",
};


type:           插件类型softvol
playback_pcm:   指定播放设备的名称
capture_pcm:    指定录音设备的名称
```

4.4.6 route 插件

```
static const snd_pcm_route_config_t snd_pcm_route_config = {
    .type = "route",
    .slave = {
        .pcm = "hw:audiocodec",
        .channels = 1,
    },
    .ttable = {
        {0, 0, 0.5},
        {1, 0, 0.5},
        TTABLE_CONFIG_END
    },
};

type:           插件类型route
slave:          pcm从设备的信息
  pcm:          pcm设备的名称
  channels:     通道数
ttable:         各通道配置,上面的配置表示将输入的两声道数据分别作0.5倍衰减，然后合成单声道数据
                第一个值表示输入的通道序号
                第二个值表示输出的通道序号
                第三个值表示衰减值
                ttable配置最后请务必添加TTABLE_CONFIG_END,表示配置结束
```

4.4.7 rate 插件

```
#define DEFINE_SND_PCM_RATE_CONFIG(name, formats, rates, converters) \
    const snd_pcm_rate_config_t snd_##name##_rate_config = { \
        .type       = "rate", \
        .slave = { \
            .pcm        = "hw:"#name, \
            .format     = formats, \
            .rate       = rates, \
        }, \
        .converter  = #converters, \
    }

static DEFINE_SND_PCM_RATE_CONFIG(audiocodec, SND_PCM_FORMAT_S16_LE, 48000, speexrate);

type:           插件类型rate
slave:          pcm从设备的信息
  pcm:          pcm设备的名称
  format        采样精度
  rate          采样率
converter       指定使用的重采样算法,建议使用speexrate
```

4.4.8 plug 插件

```
static const snd_pcm_plug_config_t snd_pcm_plug_config = {
    .type           = "plug",
    .slave          = {
        .pcm            = "PlaybackSona",
        .format         = SND_PCM_FORMAT_S16_LE,
        .channels       = 1,
        .rate           = 48000,
    },
    .rate_converter = "speexrate",
    .route_policy   = "default",
    .ttable         = {
        TTABLE_CONFIG_END
    },
};

type:               插件类型plug
slave:              pcm从设备的信息
  pcm:              pcm设备的名称
  format:           采样格式
  channels:         通道数
  rate:             采样率
rate_converter:     指定使用的重采样算法名称
route_policy:       使用route插件时的策略,可选average,copy,duplicate,default即为copy
```

5 常用工具及调试方法

5.1 aw-alsa-utils

aw-alsa-utils 是基于 aw-alsa-lib 实现的一些常用工具，例如 amixer,aplay,arecord。

5.1.1 amixer

amixer 可以操作控制音频驱动中创建的控件 (kcontrol), 例如设置 mixer,mux, 音量等。

使用方法：

```
Usage: amixer <options> [command]


Available options:
  -c,--card N           select the card

Available commands:
  amixer                show all controls for default card, default card 0
  amixer set numid val  set control contents for one control
  amixer get numid      get control contents for one control
```

举例：获取指定声卡的所有控件名

```
msh />amixer -c 0
Card Name:audiocodec.
0   codec hub mode
        Value=hub_disable       enum=hub_disable  hub_enable
1   Left LINEOUT Mux
        Value=DACL_SINGLE       enum=DACL_SINGLE  DACL_DIFFER
2   Left Input Mixer LINEINL Switch
        Value=Off       enum=Off  On
3   codec trigger substream mode
        Value=ADC_ASYNC         enum=ADC_ASYNC  ADC_I2S_SYNC
4   digital volume
        Value=0         min=0  max=63
5   LINEIN gain volume
        Value=0         min=0  max=1
6   MIC1 gain volume
        Value=31        min=0  max=31
7   LINEOUT volume
        Value=31        min=0  max=31
msh />
msh />amixer -c 1
Card Name:snddaudio0.
0   sunxi daudio audio hub mode
        Value=hub_disable       enum=hub_disable  hub_enable
1   sunxi daudio loopback debug
        Value=0         min=0  max=1
```

图 5-1: amixer_c

获取 audiocodec 声卡的所有控件名:

```
msh />amixer
0   codec hub mode
        Value=hub_disable       enum=hub_disable  hub_enable
1   Left LINEOUT Mux
        Value=DACL_SINGLE       enum=DACL_SINGLE  DACL_DIFFER
2   Left Input Mixer LINEINL Switch
        Value=Off       enum=Off  On
3   codec trigger substream mode
        Value=ADC_ASYNC         enum=ADC_ASYNC  ADC_I2S_SYNC
4   digital volume
        Value=0         min=0  max=63
5   LINEIN gain volume
        Value=1         min=0  max=1
6   MIC1 gain volume
        Value=31        min=0  max=31
7   LINEOUT volume
        Value=31        min=0  max=31
```

图 5-2: amixer

根据控件 numid, 可以进行获取、设置操作。lineout 音量控件的 numid=7。

获取当前硬件音量:

```
msh />amixer get 7
7   LINEOUT volume
        Value=31        min=0  max=31
```

图 5-3: amixer_get

设置当前硬件音量:

```
msh />amixer set 7 16
7   LINEOUT volume
        Value=16        min=0  max=31
msh />
```

图 5-4: amixer_set

5.1.2 aplay

aplay 是播歌测试工具, 支持播放 wav 音乐文件。

代码路径:

/ekernel/drivers/hal/source/sound/component/aw-alsa-utils/aplay.c

表 5-1: aplay 选项

选项	功能
-D	指定 pcm 设备名称
-p	指定 period size
-b	指定 buffer size
-l	循环播放测试
-s	停止循环播放测试
-v	播放时列出各个插件的详细信息
-h	列出使用说明

```
msh />aplay -h
Usage: aplay [option] wav_file
    -D,         pcm device name
    -H,         Hub pcm device name
    -p,         period size
    -b,         buffer size
    -l,         loop playback test
    -s,         stop loop playback test
    -v,         show pcm setup

test wav file:
16K_16bit_1ch
8K_16bit_2ch
```

通过 aplay 指定内置 wav 名字即可播放 (不指定的话默认为第一首), 如:

- aplay 16K_16bit_1ch
- aplay 8K_16bit_2ch

另外可以指定文件系统上的音频文件, 如:

- aplay /data/test.wav

5.1.3 arecord

arecord 是录音测试工具。

```
代码路径:
components/aw/sound/aw-alsa-utils/arecord.c
```

表 5-2: arecord 选项

选项	功能
-D	指定 pcm 设备名称
-r	指定采样率
-f	指定采样精度
-c	指定通道数
-p	指定 period size
-b	指定 buffer size
-t	指定录音时间
-h	列出使用说明

使用举例：

```
执行命令:arecord -r 16000 -t 10 -c 1 -D audiocodec /data/test.wav

会录制得到10s的16K,16bit, 1通道的音频数据,保存在/data/test.wav
```

5.2 dump 寄存器

5.2.1 通过 reg_read/reg_write 命令

p/m 命令可以读写 SoC 上的寄存器，可以通过查看 SoC 的 user manual, 得到具体模块的寄存器地址。

举例：

```
audiocodec模块寄存器基地址为0x05096000

将0x5096000到0x5096010的寄存器打印出来:
msh />p 0x05096000 20
start_addr=0x05096000 end_addr=0x05096014
[0x05096000]=0x00000000
[0x05096004]=0x00000000
[0x05096008]=0x00000000
[0x0509600c]=0x00000000
[0x05096010]=0x00004000

将0x5096010的值设置为0x00005000
msh />p 0x05096000 0x00005000

再用p命令查看修改是否生效:
```

```
msh />p 0x05096000 20
start_addr=0x05096000 end_addr=0x05096014
[0x05096000]=0x00000000
[0x05096004]=0x00000000
[0x05096008]=0x00000000
[0x0509600c]=0x00000000
[0x05096010]=0x00005000
```

著作权声明

商标声明

免责声明