RTOS Audio
开发指南

版本号: 1.0
发布日期: 2020.7.9

版本历史

版本号	日期	制/修订人	内容描述
1.0	2020.7.9	AWA1636	1. 初版

目　录

插　图

1 概述

1.1 文档简介

本文档是让开发者了解 Sunxi 平台音频系统框架，能够在 Sunxi 平台上开发新的音频方案。

1.2 目标读者

音频系统开发人员。

1.3 适用范围

表 1-1：适用产品列表

产品名称	内核版本	驱动文件
F133	Melis	hal/source/sound/*
V459	Melis	hal/source/sound/*

2 模块介绍

2.1 模块功能介绍

在 SUNXI 中，从软件上可以存在多个音频设备。分别为 audiocodec，daudio0，daudio1，daudio2, daudio3,dmic, spdif。

2.1.1 AudioCodec 模块功能

Audio Codec 驱动所具有的功能：

- 支持多种采样率格式 (8KHz，11.025 KHz, 12 KHz, 16 KHz, 22.05 KHz, 24 KHz, 32 KHz, 44.1 KHz , 48 KHz, 96KHz, 192KHz)，其中录音最大支持 48KHz；
- 支持同时 playback 和 record(全双工模式)；
- 支持 mixer 接口；
- 支持 dapm 接口；
- 支持 16bit/24bit 数据精度；
- 支持 DAC，采样率为 8kHz~192kHz，支持差分输出；
- 支持 ADC，采样率为 8kHz~48kHz；
- DAC 及 ADC 均支持 DRC 功能；
- DAC FIFO 长度 128*24bits, ADC FIFO 长度 128*24bits；

2.1.2 Daudio 模块功能

驱动所具有的功能：

- 支持多种采样率格式 (8kHz，11.025kHz，16kHz，22.05kHz，24kHz，32kHz，44.1kHz，48kHz，88.2kHz，96kHz，176.4kHz，192kHz);
- 支持主从模式
- 支持 Left-justified,Right-justified,Standar mode I2S,PCM mode
- 支持 1~8 通道；
- 支持同时 playback 和 record(全双工模式)；
- 支持 i2s、pcm 协议格式配置；
- 支持 16bit/24bit/32bit 数据精度；

2.1.3 DMIC 模块功能

驱动所具有的功能：

- 支持多种采样率格式 (8kGz，11.025kHz，16kHz，22.05kHz，24kHz，32kHz，44.1kHz，48kHz);
- 最多支持 8 通道;
- 只支持 record;
- 支持 64 OSR 以及 128 OSR;
- 支持 16bit/24bit 数据精度；

2.1.4 S/PDIF 模块功能

驱动所具有的功能：

- 支持多种采样率格式 (44.1kGz，48kHz，96kHz，192kHz);
- 支持单声道和立体声输出;
- 支持 16bit/20bit/24bit 数据精度；
- 支持同时 playback 和 record(全双工模式)；

2.2 相关术语

2.2.1 硬件术语

表 2-1: 硬件术语

相关术语	解释说明
audiocodec	芯片内置音频接口
DMIC	外置数字 MIC 接口
SPDIF	外置音响音频设备接口，一般使用同轴电缆或光纤接口
I2S	外置音频通道接口
Daudio	数字音频接口，可配置成 i2s/pcm 格式标准音频接口

2.2.2 软件术语

表 2-2: 软件术语

相关术语	解释说明
Sunxi	全志科技使用的 linux 开发平台
ASOC	ALSA System on Chip
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
样本长度 (sample)	样本是记录音频数据最基本的单位，常见的有 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
DRC	音频输出动态范围控制
HPF	高通滤波
XRUN	音频流异常状态，分为 underrun 和 overrun 两种状态
DAPM	动态音频电源管理
hp	headphone 缩写，耳机/耳麦
交错模式 (interleave)	是一种音频数据的记录模式，在交错模式下，数据以连续帧的形式存放，即首先记录完帧 1 的左声道样本和右声道样本（假设为立体声格式），再开始帧 2 的记录，而在非交错模式下，首先记录的是一个周期内所有帧的左声道样本，再记录右声道样本，数据是以连续通道的方式存储。多数情况下，只需要使用交错模式

2.3 模块配置介绍

2.3.1 menuconfig 配置

在命令行进入 source 目录，执行 make menuconfig 进入配置主界面，并按以下步骤操作：

- 1. 选择 Kernel Setup 选项进入下一级配置，如下图所示：

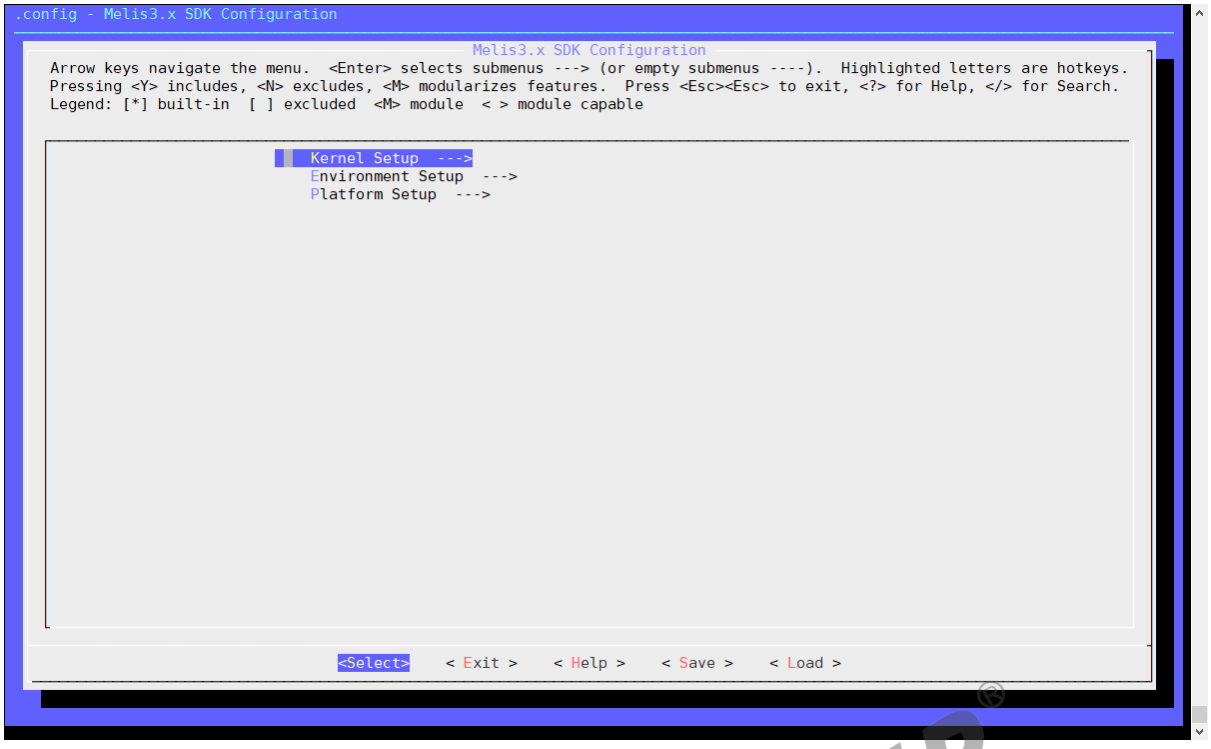

图 2-1: Kernel Setup

- 2. 选择 Drivers Setup 选项，进入下一级配置，如下图所示：

图 2-2: Drivers Setup

- 3. 选择 SoC HAL Drivers 选项，如下图所示：

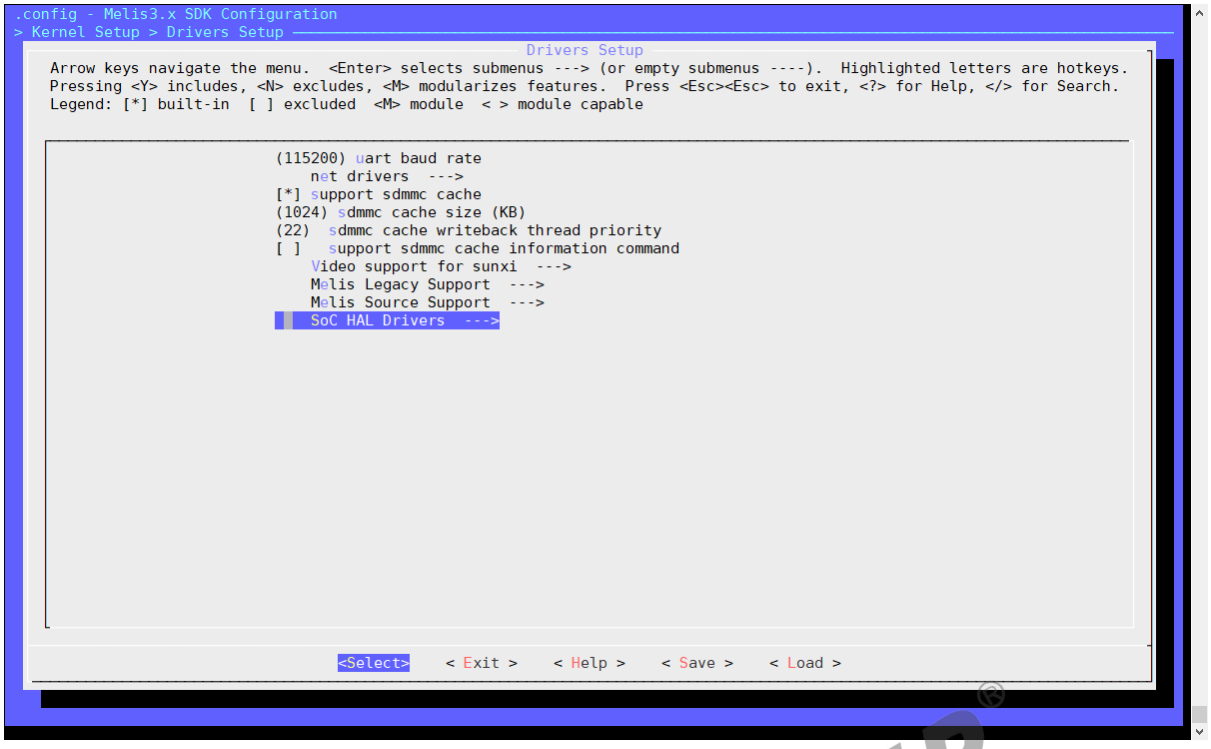

图 2-3: SoC HAL Drivers

- 4. 选择 Sound card support 选项，进入下一级配置，如下图所示：

图 2-4: Sound card support

- 5. 选择 AllWinner CODEC drivers 选项，如下图所示：

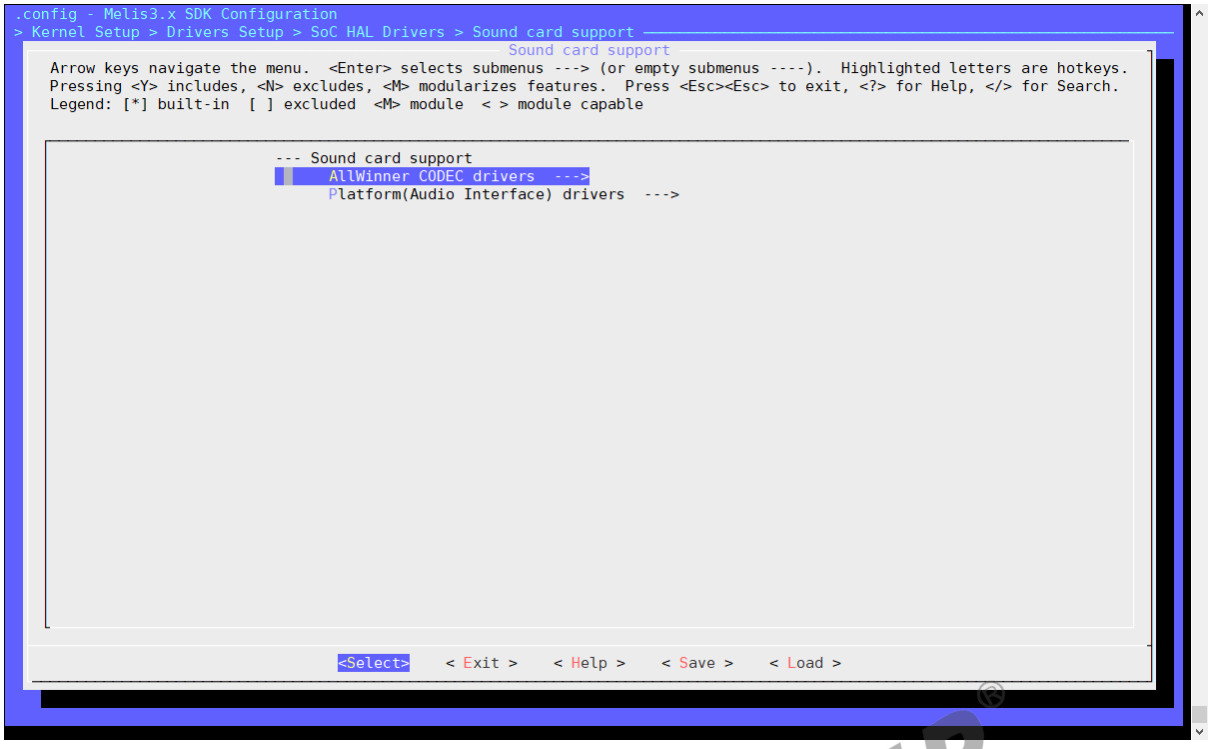

图 2-5: AllWinner CODEC drivers

- 6. 选择需要的模块，可选择直接编译进内核，如下图所示：

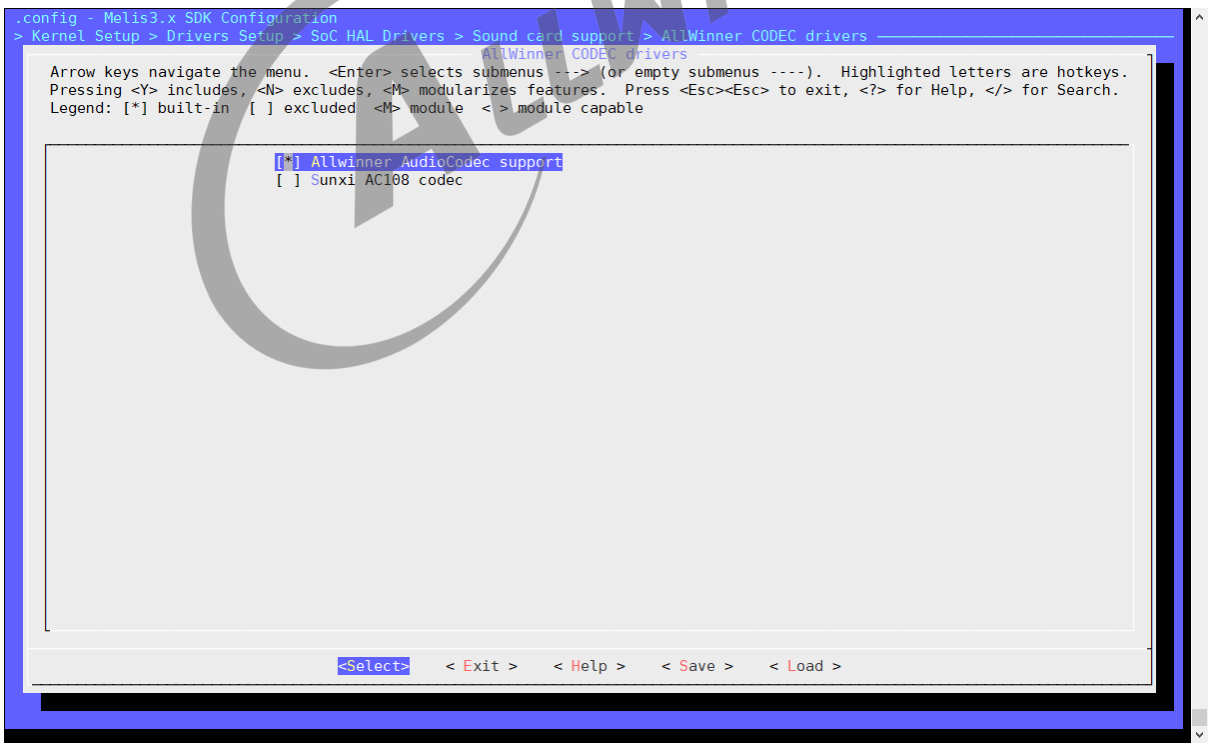

图 2-6: module

- 7. 返回上级目录，选择 Platform(Audio Interface) drivers 选项，配置对应的接口，如下图所示：

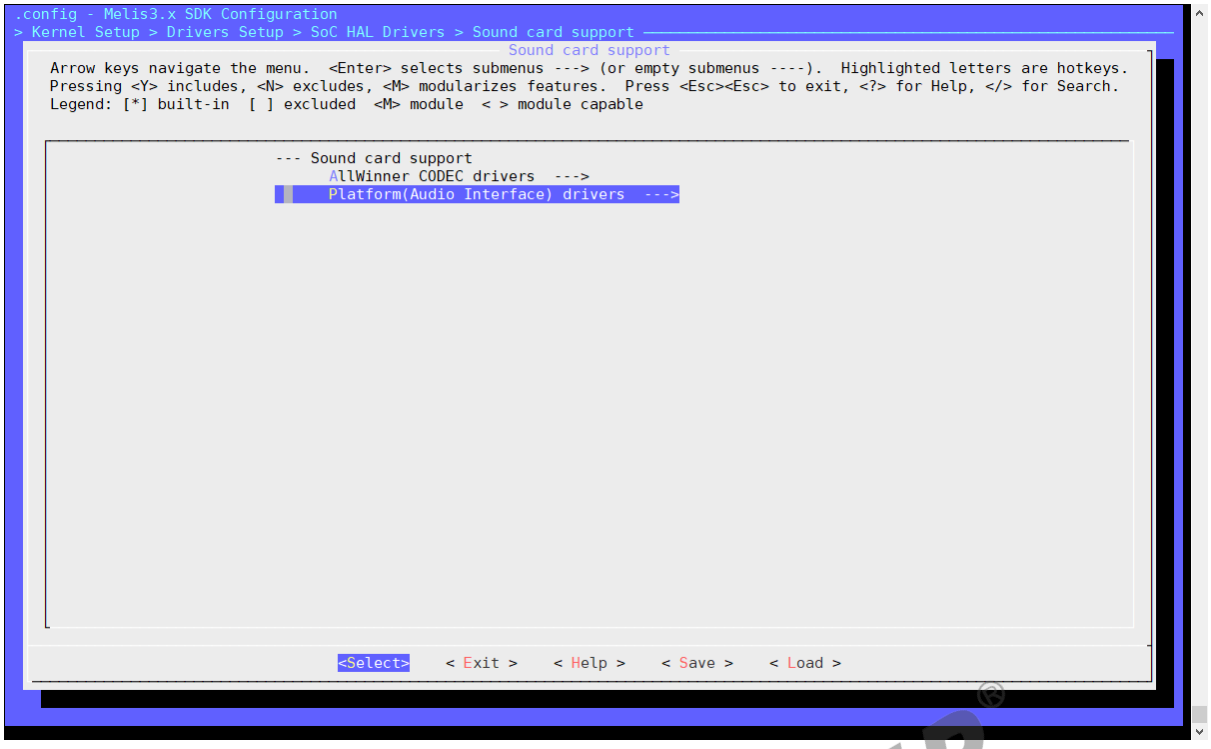

图 2-7: module

- 8. 选择需要的接口，可选择直接编译进内核，如下图所示：

图 2-8: module

2.4 源码模块结构

```
hal/source/sound/card_default.c    // 每个方案都要有该文件，用于配置加载哪些声卡

hal/source/sound/
├── core/                          // 音频驱动核心层代码
├── codecs/                        // codec驱动代码
│   ├── dummy_codec.c              // dummy codec,用于加载没有实际硬件codec的声卡
│   ├── sun8iw19-codec.c           // V459 audiocodec驱动
│   ├── sun8iw20-codec.c           // F133 audiocodec驱动
│   ├── ac107.c                    // ac107 codec
│   ├── ac108.c                    // ac108 codec
│   └── sunxi_rw_func.c          // 操作audiocodec寄存器的通用接口
└── platform/                      // platform驱动代码
    ├── sunxi-dummy-cpudai.c       // Allwinner cpu-dai驱动
    ├── sunxi-pcm.c                // Allwinner platform驱动的通用接口
    ├── sunxi-dmic.c               // Allwinner DMIC驱动
    └── sunxi-daudio.c             // Allwinner i2s驱动
```

2.5 驱动框架介绍

2.5.1 音频驱动硬件框架图

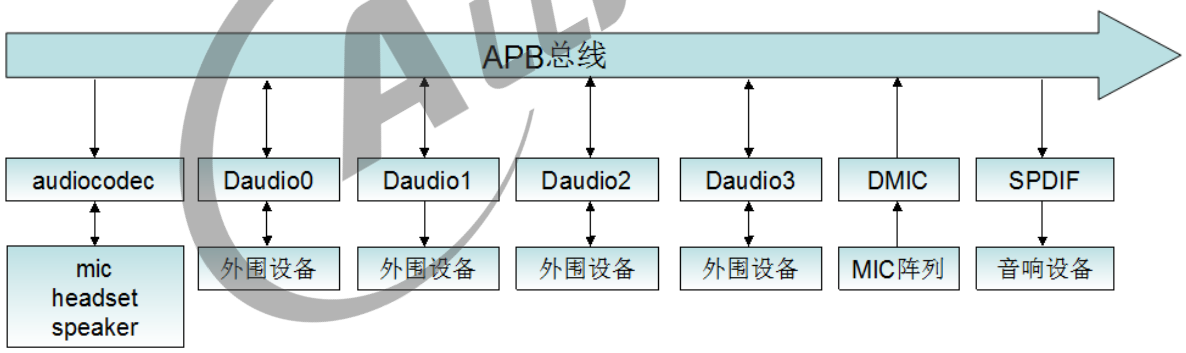

图 2-9: hardware

2.5.2 音频驱动软件框架图

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：

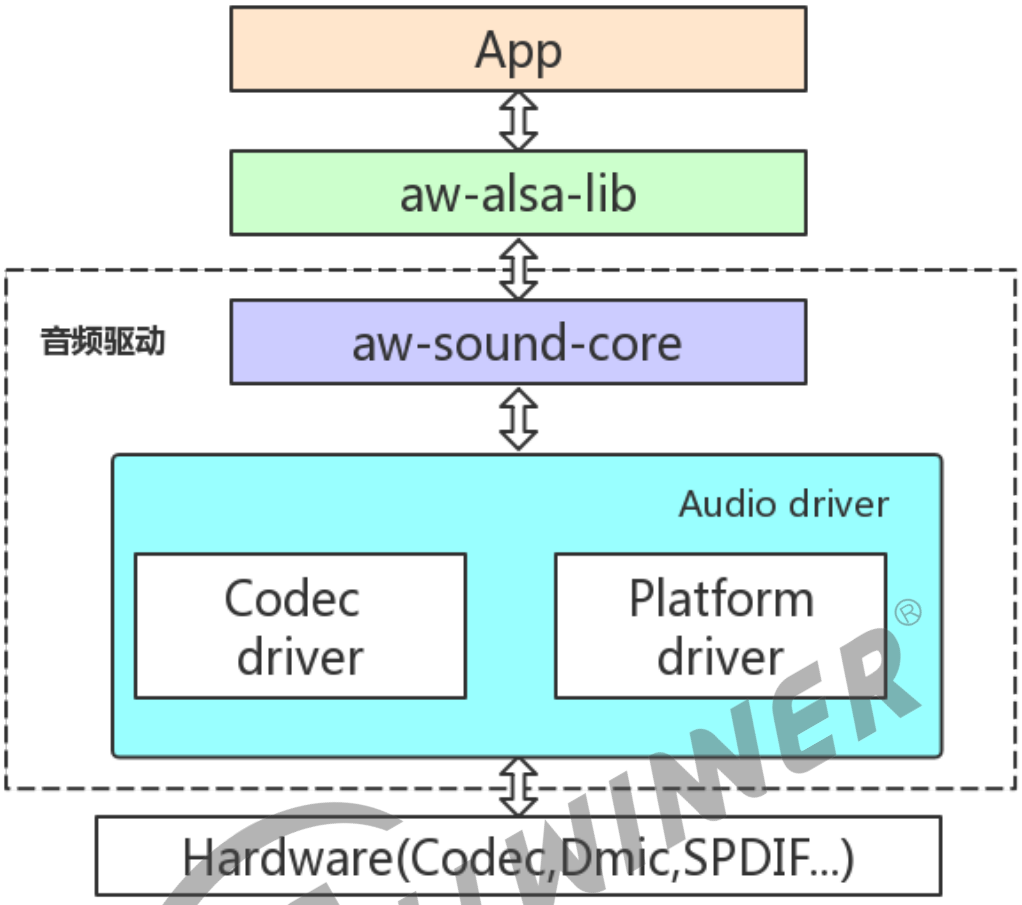

图 2-10: RTOS 音频驱动

3 模块接口说明

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 4.5 章节

接口头文件在 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

3.1.1 snd_pcm_open

- 作用：打开声卡得到 pcm 设备。
- 函数原型：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

3.1.2 snd_pcm_close

- 作用：关闭 pcm 设备
- 函数原型：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

3.2 hw_params 设置相关接口:

3.2.1 snd_pcm_hw_params_alloca

- 作用：创建 hw_params 的宏
- 函数原型：void snd_pcm_hw_params_alloca(void)
- 参数:
  - null
- 返回值:
  - null

3.2.2 snd_pcm_hw_params_any

- 作用：填充 hw_param 配置
- 函数原型：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:
  - 0：成功
  - <0：error code

3.2.3 snd_pcm_hw_params_set_format

- 作用：设置采样格式
- 函数原型：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:
  - 0：成功
  - <0：error code

3.2.4 snd_pcm_hw_params_set_channels

- 作用：设置通道数
- 函数原型：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:
  - 0：成功
  - <0：error code

3.2.5 snd_pcm_hw_params_set_rate

- 作用：设置采样率
- 函数原型：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:
  - 0：成功
  - <0：error code

3.2.6 snd_pcm_hw_params_set_period_size

- 作用：设置 period_size
- 函数原型：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:
  - 0：成功
  - <0：error code

3.2.7 snd_pcm_hw_params_set_buffer_size

- 作用：设置 buffer_size
- 函数原型：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:
  - 0：成功
  - <0：error code

3.2.8 snd_pcm_hw_params

- 作用：安装 hw_params 到声卡中
- 函数原型：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:
  - 0：成功
  - <0：error code

3.3 sw_params 设置相关接口:

3.3.1 snd_pcm_sw_params_alloca

- 作用：创建 sw_params 的宏

- 函数原型：snd_pcm_sw_params_alloca()
- 参数:
  - null
- 返回值:
  - null

3.3.2 snd_pcm_sw_params_current

- 作用：填充 sw_param 配置
- 函数原型：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:
  - 0：成功
  - <0：error code

3.3.3 snd_pcm_sw_params_set_start_threshold

- 作用：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)
- 函数原型：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:
  - 0：成功
  - <0：error code

3.3.4 snd_pcm_sw_params_set_stop_threshold

- 作用：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)
- 函数原型：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

3.3.5 snd_pcm_sw_params_set_avail_min

- 作用：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
- 函数原型：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:
  - 0：成功
  - <0：error code

3.3.6 snd_pcm_sw_params

- 作用：安装 sw_params 到声卡中
- 函数原型：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:
  - 0：成功
  - <0：error code

3.4 pcm 数据读写接口:

3.4.1 snd_pcm_writei

- 作用：写入 size 帧 pcm 数据
- 函数原型：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:
  - >0：已写入的帧数
  - -EBADFD：表示 pcm 设备目前的 state 不允许当前操作
  - -EPIPE：表示出现 underrun

3.4.2 snd_pcm_readi

- 作用：读出 size 帧 pcm 数据
- 函数原型：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:
  - >0：已读出来的帧数
  - -EBADFD 表示 pcm 设备目前的 state 不允许当前操作
  - -EPIPE 表示出现 overrun

3.5 pcm 设备控制接口:

3.5.1 snd_pcm_prepare

- 作用：将 pcm 设备设置为 prepare 状态
- 函数原型：int snd_pcm_prepare(snd_pcm_t *pcm)
- 参数:
  - pcm: pcm 操作句柄
- 返回值:
  - 0：成功
  - <0：error code

3.5.2 snd_pcm_drop

- 作用：停止 pcm 设备, 并丢掉 pending frames
- 函数原型：int snd_pcm_drop(snd_pcm_t *pcm)
- 参数:
  - pcm: pcm 操作句柄
- 返回值:
  - 0：成功
  - <0：error code

3.5.3 snd_pcm_drain

- 作用：停止 pcm 设备, 并会把 pending frames 耗尽
- 函数原型：int snd_pcm_drain(snd_pcm_t *pcm)
- 参数:
  - pcm: pcm 操作句柄
- 返回值:
  - 0：成功
  - <0：error code

3.5.4 snd_pcm_pause

- 作用：暂停/恢复 pcm 设备
- 函数原型：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:
  - 0：成功
  - <0：error code

3.5.5 snd_pcm_recover

- 作用：从异常状态 (xrun) 恢复
- 函数原型：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:
  - 0：成功
  - <0：error code

3.5.6 snd_pcm_state

- 作用：得到当前 pcm 状态
- 函数原型：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

3.5.7 snd_pcm_delay

- 作用：
  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
- 函数原型：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:
  - 0：成功
  - <0：error code

3.6 control 控件操作接口

3.6.1 snd_ctl_num

- 作用：获取指定声卡的控件个数
- 函数原型：int snd_ctl_num(const char *name)
- 参数:
  - name: 声卡名称
- 返回值:
  - 0：成功
  - <0：error code

3.6.2 snd_ctl_get

- 作用：获取指定声卡、指定控件名称的信息
- 函数原型：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:
  - 0：成功
  - <0：error code

3.6.3 snd_ctl_get_bynum

- 作用：获取指定声卡、指定控件 id 的信息
- 函数原型：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:
  - 0：成功
  - <0：error code

3.6.4 snd_ctl_set

- 作用：设定指定声卡、指定控件名称的值
- 函数原型：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:
  - 0：成功
  - <0：error code

3.6.5 snd_ctl_set_bynum

- 作用：设定指定声卡、指定控件 id 的值
- 函数原型：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:
  - 0：成功
  - <0：error code

4 软件配置说明

4.1 AudioCodec 配置

AudioCodec 配置位于代码中, 路径如下：

```
hal/source/sound/codecs/xxx-codec.c
在该结构中定义struct sunxi_codec_param default_param
```

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 0~0x7, 0:0dB, 1~7:15~33dB, 3dB/step, 一般设置 0x4, 即 24dB
mic2gain	mic2 增益，可设定范围 0~0x7, 0:0dB, 1~7:15~33dB, 3dB/step, 一般设置 0x4, 即 24dB
mic3gain	mic3 增益，可设定范围 0~0x7, 0:0dB, 1~7:15~33dB, 3dB/step, 一般设置 0x4, 即 24dB. 如果作为 aec 回路，则需要设置为 0dB
adcgain	adc 增益，可设定范围 0~0x7, 表示-4.5~6dB, 1.5dB/step, 一般设置 0x3, 即 0dB
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
pa_msleep_time	操作 PA 之后的延时时间 (用来避免 pop 音)
gpio_spk	PA 使能引脚
gpio_spk_power	功放芯片供电开关

4.2 i2s 配置

i2s 配置位于代码中, 路径如下：

```
hal/source/sound/platform/sunxi-daudio.c
在该结构中定义struct sunxi_daudio_param daudio_param[]
```

snddaudio 配置	snddaudio 配置说明
tdm_num	表示第几组 i2s
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)
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
slot_width_select	支持 8bit, 16bit, 32bit 宽度
msb_lsb_first	0：msb first; 1：lsb first
frametype	0：short frame = 1 clock width; 1：long frame = 2 clock width
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
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：

```
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R328的audiocodec,使用sun8iw18_codec;如
    ac108,使用ac108_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V459 audiocodec声卡
snd_card_register("audiocodec", &sun8iw19_codec, SND_PLATFORM_TYPE_CPUDAI);

注册AC108声卡
snd_card_register("ac108", &ac108_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：

```
在每个方案目录下面:
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

```
static const snd_pcm_hw_config_t snd_pcm_hw_config = {
    .card_name  = "audiocodec",
    .device_num = 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 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       = 48000,
        .channels   = 1,
```

```
        .period_size    = 1024,
        .periods    = 4,
    },
};

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
```

4.4.3 dsnoop 插件

```
static const snd_pcm_dsnoop_config_t snd_pcm_dsnoop_ref_config = {
    .type       = "dsnoop",
    .ipc_key    = 1114,
    .slave = {
        .pcm        = "hw:ac107",
        .format     = SND_PCM_FORMAT_S16_LE,
        .rate       = 16000,
        .channels   = 1,
        .period_size    = 1024,
        .periods    = 4,
    },
};

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 softvol 插件

```
static const snd_pcm_softvol_config_t snd_pcm_softvol_config = {
    .type       = "softvol",
    .slave = {
        .pcm        = "PlaybackPlug",
    },
    .control = {
        .control_name   = "Soft Volume Master",
        .card_name  = "audiocodec",
    },
    .min_dB         = -51.0,
```

```
    .max_dB         = 0.0,
    .resolution     = 256,
};

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
```

说明

在第一次使用 *softvol* 插件进行播放时才会生成该控件；如果想要在声卡驱动加载的时候就添加该控件，请修改对应的 *codec* 文件，例如 *sun8iw18-codec.c* 中添加了下面的代码：

SND_CTL_KCONTROL_USER(“Soft Volume Master”, 255, 0, 255)

4.4.5 asym 插件

```
static const snd_pcm_asym_config_t snd_pcm_asym_config = {
    .type       = "asym",
    .playback_pcm   = "PlaybackSoftVol",
    .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 = "PlaybackRate",
        .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

```
static const snd_pcm_rate_config_t snd_pcm_rate_config = {
    .type           = "rate",
    .slave = {
        .pcm        = "PlaybackDmix",
        .format     = SND_PCM_FORMAT_S16_LE,
        .rate       = 48000,
    },
    .converter  = "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 file 插件

```
static const snd_pcm_file_config_t snd_pcm_file_cap_config = {
    .type       = "file",
    .slave      = {
        .pcm        = "CaptureDsnoop",
    },
    .format     = "raw",
    .mode       = "adb",
    .port       = 20191,
};

type:           插件类型file
slave:          pcm从设备的信息
  pcm:          pcm设备的名称
format:         音频格式，目前仅支持raw
mode:           file插件保存数据的模式，目前支持adb,network
port:           端口号
server:         服务器端ip，当mode为network时才有效
```

4.4.9 multi 插件

```
static const snd_pcm_multi_config_t snd_pcm_3mic_1ref_config = {
    .type       = "multi",
    .slaves     = {
        { "a", "CaptureDsnoop3Mic", 3 },
        { "b", "CaptureDsnoopRef", 1 },
        { NULL, NULL, -1 },
    },
    .bindings   = {
        { 0, "a", 0 },
        { 1, "a", 1 },
        { 2, "a", 2 },
        { 3, "b", 0 },
        { -1, NULL, -1 },
    },
};

type:           插件类型multi
slaves:         pcm从设备的信息,上述配置表示同时对a,b两个pcm设备进行录音
                第一个参数表示pcm设备的别名,方便后续bingdings域指定不同的pcm设备
                第二个参数表示pcm设备的名称
                第三个参数表示通道总数
                注意请在配置最后添加{ NULL, NULL, -1 },表示结束
bindings:       指定多个声卡的通道数排列,上述配置表示a声卡的3通道分别作为通道0,1,2输出,b声卡的通道0作为通
    道3输出
                第一个参数表示录音输出的通道序号
                第二个参数表示指定声卡的通道序号
                注意请在配置最后添加{ -1, NULL, -1 },表示结束
```

4.4.10 plug 插件

```
static const snd_pcm_plug_config_t snd_pcm_plug_config = {
    .type           = "plug",
    .slave          = {
        .pcm            = "PlaybackDmix",
        .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：

命令	功能
-c	指定要操作的声卡，默认为声卡 0, 可以指定声卡名称, 如 audiocodec
controls	列出指定声卡的所有控件
cget	获取指定控件的信息
cset	设定指定控件的值

举例：

```
获取audiocodec声卡的所有控件名：
amixer -c audiocodec controls

根据控件numid,可以进行获取、设置操作
lineout音量控件的numid=5

获取当前硬件音量：
amixer -c audiocodec cget numid=5

设置当前硬件音量：
amixer -c audiocodec cset numid=5 25
```

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：
hal/source/sound/component/aw-alsa-utils/aplay.c
```

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

如果配置了 CONFIG_BUILTIN_WAV_FILE，默认会把两个 wav 文件编译到固件包中，通过 aplay -h 会打印出当前 builtin 的 wav 文件：

```
cpu1>aplay -h
Usage: aplay [option] wav_file
-D,          pcm device name
-p,          period size
-b,          buffer size
-l,          loop playback test
-s,          stop loop playback test
-v,          show pcm setup

builtin 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 (请参考文件系统相关使用说明文档，将 wav 文件放入文件系统中)，如:

- aplay /data/test.wav

如果定义了不同的插件，可通过-D 指定进行测试：

- aplay -DPlaybackDmix /data/test.wav

5.1.3 arecord

arecord 是录音测试工具。

```
代码路径:
hal/source/sound/component/aw-alsa-utils/arecord.c
```

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
-d	指定录音时间
-l	循环录音测试, 录音数据不会保存
-k	停止循环录音测试
-h	列出使用说明
-t	录音结束后紧跟着播放 (注意仅用于测试，录音太长会申请大内存)

使用举例：

```
执行命令:arecord -r 16000 -f 16 -c 3 -d 5 /data/test.wav
他会录制得到5s的16K,16bit,3通道的音频数据,保存在/data/test.wav
```

```
循环录制老化:
执行命令:arecord -l
它会一直执行录音操作，并且录音内容不会保存,再输入arecord -k可以停止录音
```

```
注意，因为当前文件系统写入速度较慢,如果边录边保存，会导致overrun，所以程序会在一开始申请足够大的内存，
并把录制数据均保存在内存中，等录制时间到了后，再一起写入文件系统中。
```

5.1.4 soundcard

soundcard 查询声卡信息的工具。

```
代码路径:
hal/source/sound/component/aw-alsa-utils/card.c
```

选项	功能
-c	指定声卡序号
-l	列出当前已经注册的声卡
-i	列出指定声卡的详细信息
-s	列出指定声卡的详细播放/录音信息, 0-playback; 1-capture

使用举例：

```
soundcard -l
列出当前已加载的声卡:
Sound Card list:
card_num           card_name
       0          audiocodec


soundcard -c 0 -i
列出声卡0的详细信息
Sound Card:               0  audiocodec
PCM device:               0
   Playback stream: PCM AC-codecdai Playback
   Capture stream:  PCM AC-codecdai Capture


soundcard -c 0 -s 0
列出声卡0的播放流信息
Sound Card:               0  audiocodec
PCM device:               0
Playback stream: Open
rate:            48000
channels:        1
period_size:     1024
periods:         4
buffer_size:     4096
hw_ptr:          0x96c0
appl_ptr:        0x0

soundcard -c 0 -s 1
列出声卡0的录音流信息
Sound Card:               0  audiocodec
PCM device:               0
Capture stream: Closed
```

5.2 dump 寄存器

5.2.1 通过 reg_read/reg_write 命令

reg_read/reg_write 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0x5096000
    将0x5096000开始后面4个寄存器打印出来:
    reg_read 0x5096000 0x10

    将0x5096010的值设置为0x60004000
    reg_write 0x5096010 0x80004000
```

5.2.2 通过 audiocodec 驱动接口

需要修改下面源文件，打开宏 SUNXI_AUDIOCODEC_REG_DEBUG

```
hal/source/sound/codecs/sun8iw19-codec.c
```

然后在想要打印 audiocodec 寄存器的地方，加入函数 sunxi_audiocodec_reg_dump

打印的结果如下:

```
SUNXI_DAC_DPC         [0x000]: 0x80000000
SUNXI_DAC_FIFO_CTL    [0x010]: 0x63004000
SUNXI_DAC_FIFO_STA    [0x014]: 0x3e04
SUNXI_DAC_CNT         [0x024]: 0x3e800
SUNXI_DAC_DG          [0x028]: 0x0
SUNXI_ADC_FIFO_CTL    [0x030]: 0xe000400
SUNXI_ADC_FIFO_STA    [0x038]: 0x0
SUNXI_ADC_CNT         [0x044]: 0x0
SUNXI_ADC_DG          [0x04c]: 0x0
SUNXI_DAC_DAP_CTL     [0x0f0]: 0x0
SUNXI_ADC_DAP_CTL     [0x0f8]: 0x0
SUNXI_HP_CTL          [0x300]: 0x0
SUNXI_MIX_DAC_CTL     [0x303]: 0x40
SUNXI_LINEOUT_CTL0    [0x305]: 0xd0
SUNXI_LINEOUT_CTL1    [0x306]: 0x13
SUNXI_MIC1_CTL        [0x307]: 0x34
SUNXI_MIC2_MIC3_CTL   [0x308]: 0x4
SUNXI_LADCMIX_SRC     [0x309]: 0x4
SUNXI_RADCMIX_SRC     [0x30a]: 0x8
SUNXI_XADCMIX_SRC     [0x30b]: 0x10
SUNXI_ADC_CTL         [0x30d]: 0x3
SUNXI_MBIAS_CTL       [0x30e]: 0x21
SUNXI_APT_REG         [0x30f]: 0xd6
SUNXI_OP_BIAS_CTL0    [0x310]: 0x55
SUNXI_OP_BIAS_CTL1    [0x311]: 0x55
SUNXI_ZC_VOL_CTL      [0x312]: 0x2
SUNXI_BIAS_CAL_CTRL   [0x315]: 0x0
```

图 5-1: R329 audiocodec 寄存器

著作权声明

商标声明

免责声明