全 志 科 技
Allwinner Technology

A20 音频模块开发说明

V1.1

2014-06-20

Revision History

Version	Date	Changes compared to previous issue
V1.0	2013-03-15	初建版本
V1.1	2014-06-20	更新 audio_para 配置，删除部分与盒子无关的内容

目录

1. 前言

1.1. 编写目的

本文档目的是为了让开发者了解 A20 音频系统框架，能够在 A20 平台上开发新的音频方案。

1.2. 适用范围

本模块说明适用于 A20 平台

1.3. 相关人员

音频系统开发人员。

2. 音频模块介绍

linux 中的 audio 子系统采用 alsa 架构实现。alsa 目前已经成为了 linux 的主流音频体系结构。在内核设备驱动层，ALSA 提供了 alsa-driver，同时在应用层，ALSA 为我们提供了 alsa-lib，应用程序只要调用 alsa-lib 提供的 API，即可以完成对底层音频硬件的控制。

2.1. audio 模块功能介绍

在 a20 中，存在 4 个音频设备。分别为 audiocodec，hdmiaudio，spdif，i2s。其中芯片内置的 audiocodec 采用 alsa-pci 架构实现，hdmiaudio，spdif，i2s 采用 alsa-asoc 架构实现。switch 主要实现耳机检测的功能。i2s 都可以配置成 pcm 和 i2s 两种模式。

2.1.1. audio codec 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);
- 支持 mono 和 stereo 模式;
- 支持同时 playback 和 record(全双工模式);
- 支持 start, stop, pause 和 resume;
- 支持 mixer 接口
- 支持 3g 通话功能

2.1.2. hdmiaudio 功能

hdmiaudio 驱动所具有的功能：

- 支持多种采样率格式(8khz，11.025khz，12khz，16khz，22.05, 24khz，32khz，44.1, 48khz，88.2khz，96khz，176.4khz，192khz);
- 支持 mono 和 stereo 模式;
- 只支持 playback 模式，不支持 record 模式。
- 支持 start, stop, pause 和 resume;

2.1.3. spdif 功能

spdif 驱动所具有的功能：

- 支持多种采样率格式(22.05khz，24khz，32khz，44.1khz，48khz，88.2khz，96khz，176.4khz，192khz);
- 支持 mono 和 stereo 模式;
- 只支持 playback 模式，不支持 record 模式。

- 支持 start, stop, pause 和 resume;

2.1.4. i2s 功能

i2s 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 mono 和 stereo 模式;
- 支持同时 playback 和 record(全双工模式);
- 支持 start, stop, pause 和 resume;

I2s 驱动可以配置成 i2s 模式，也可以配置成 pcm 模式，如果配置成 pcm 模式，那么只支持 8k 采样率。在 a20 中，有两套 pcm 驱动，为了区分，分别命名为 pcm 和 i2s。

2.1.5. switch 耳机检测功能

Switch 支持 3 段耳机，4 段耳机的插拔检测。

2.2. 源码结构介绍

a20 中的音频子系统存放于 soc 目录下，如图 1 音频系统源码所示。其中 Audiocodec，hdmiaudio，i2s，spdif 都是一个独立的音频驱动。耳机检测驱动源码如图 2 switch 源码所示。

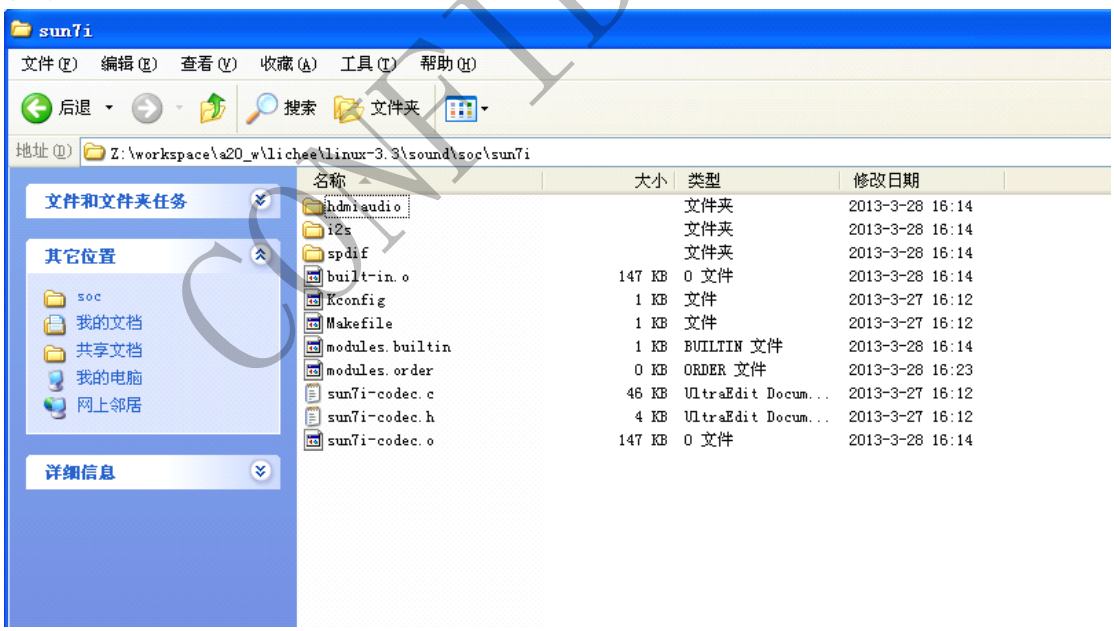

图 1 音频系统源码

\linux-3.3\drivers\staging\android\switch

名称	大小	类型	修改日期
.built-in.o.cmd	1 KB	Windows NT 命令...	2013-2-18 8:51
.switch_class.o.cmd	22 KB	Windows NT 命令...	2013-2-18 8:51
.switch_headset.o.cmd	26 KB	Windows NT 命令...	2013-2-18 8:51
built-in.o	123 KB	O 文件	2013-2-18 8:51
Kconfig	1 KB	文件	2013-2-18 8:43
Makefile	1 KB	文件	2013-2-18 8:43
modules.builtin	1 KB	BUILTIN 文件	2013-2-21 15:43
modules.order	0 KB	ORDER 文件	2013-2-22 17:43
switch.h	2 KB	H 文件	2013-2-18 8:43
switch_class.c	5 KB	C 文件	2013-2-18 8:43
switch_class.o	63 KB	O 文件	2013-2-18 8:51
switch_gpio.c	5 KB	C 文件	2013-2-18 8:43
switch_headset.c	11 KB	C 文件	2013-2-18 8:43
switch_headset.o	67 KB	O 文件	2013-2-18 8:51

图 2.Switch 源码

2.3. audio 相关术语介绍

Audio Driver: Acronyms	
Acronym	**Definition**
ALSA	Advanced Linux Sound Architecture
DMA	即直接内存存取，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.
OSS	Open Sound System
样本长度 sample	样本是记录音频数据最基本的单位，常见的有 8 位和 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。
交错模式 interleaved	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。不过多数情况下，我们只需要使用交错模式就可以了。
Audiocodec	芯片内置音频接口
Hdmiaudio	内置 hdmi 音频接口
Spdif	外置音响音频设备接口

2.4. audio 模块配置介绍

2.4.1. Menuconfig 配置

在编译服务器上，目录为\lichee\linux-3.3 上，输入命令：

```
make ARCH=arm menuconfig
```

如下所示：

```
/lichee/linux-3.3$ make ARCH=arm menuconfig
```

- 音频驱动配置

音频模块中包括 audiocodec，hdmiaudio，spdif，i2s 共 4 个音频驱动。他们的配置项如图 3 到图 7 所示。

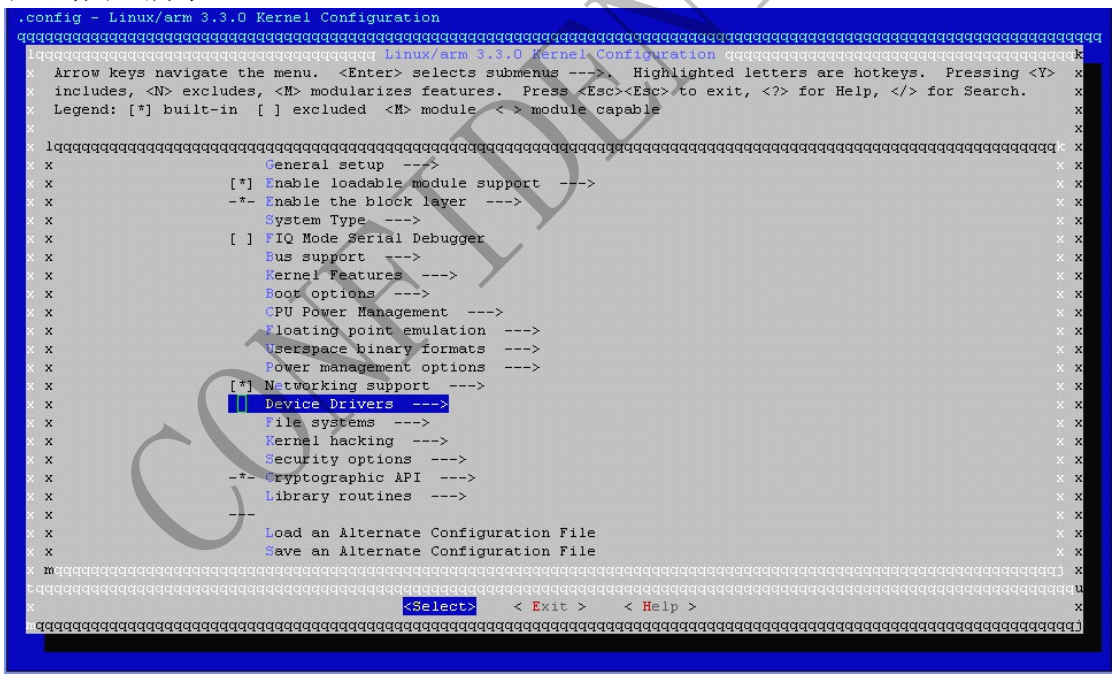

图 3.Device Drivers 选项配置

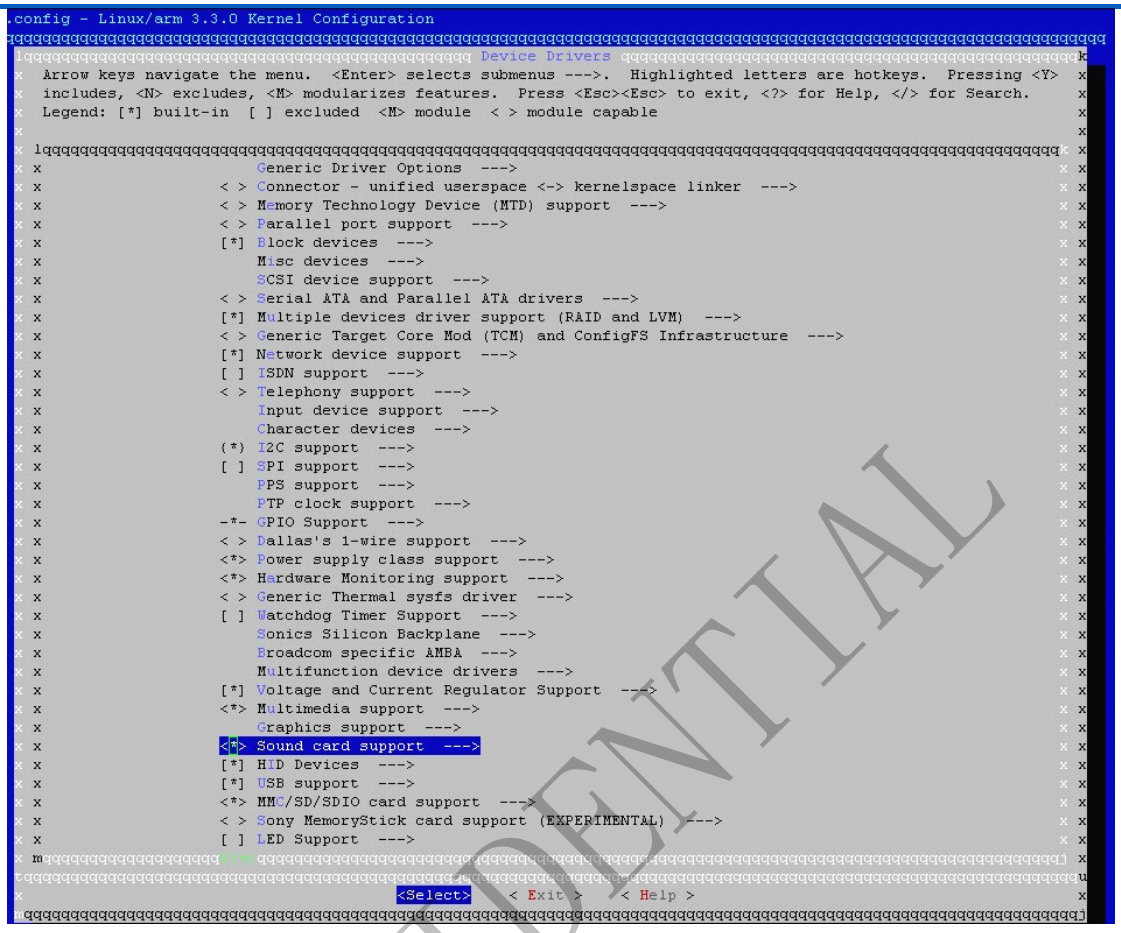

图 4.音频模块配置

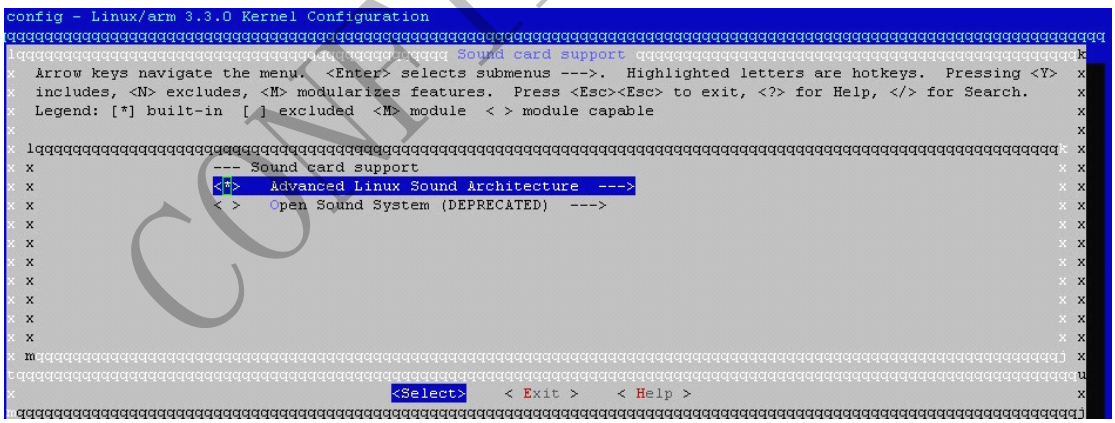

图 5.音频 ALSA 模块配置

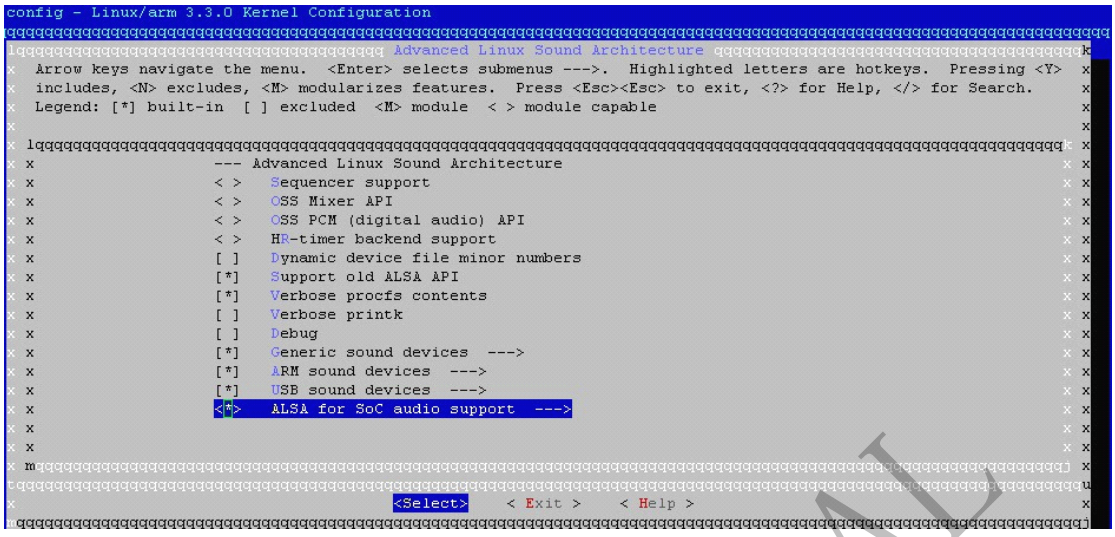

图 6.音频 soc 模块配置

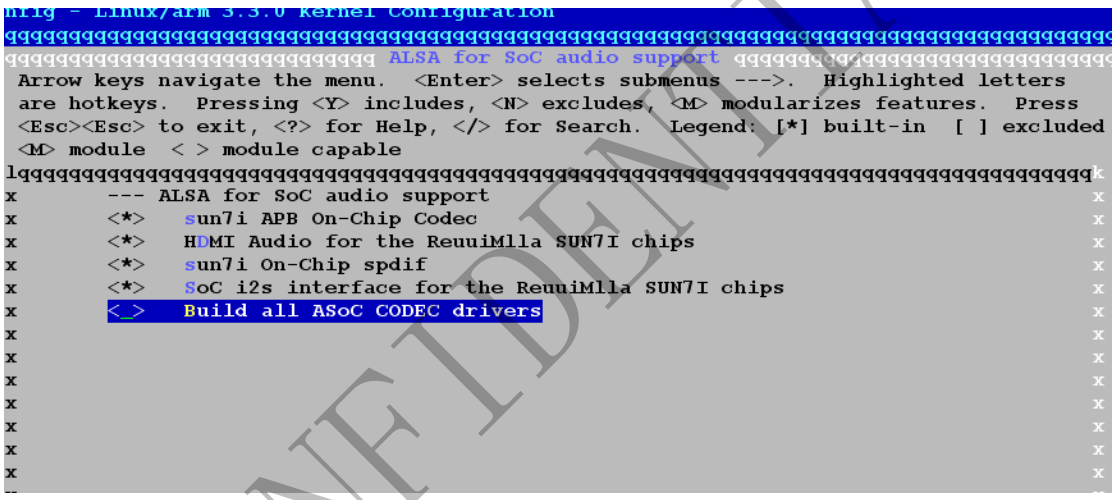

图 7.音频音频驱动配置

- Hdmi 配置

如果要编译 hdmiaudio，也需要将 hdmi video 编译到内核中。如图 8，图 9 所示。进入 device driver 后，选择 graphics support，然后选择 buildin HDMI driver support（sun7i）即可。

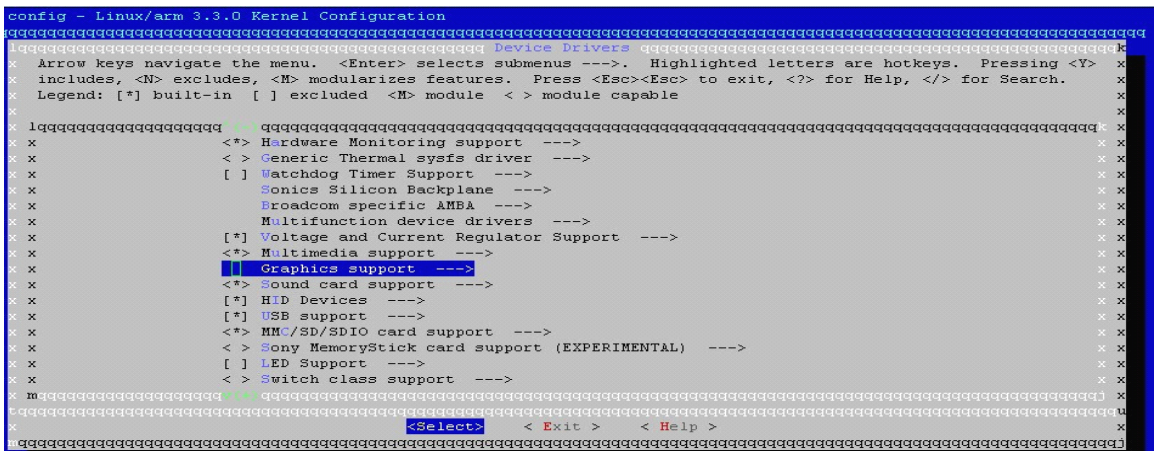

图 8　graphics support

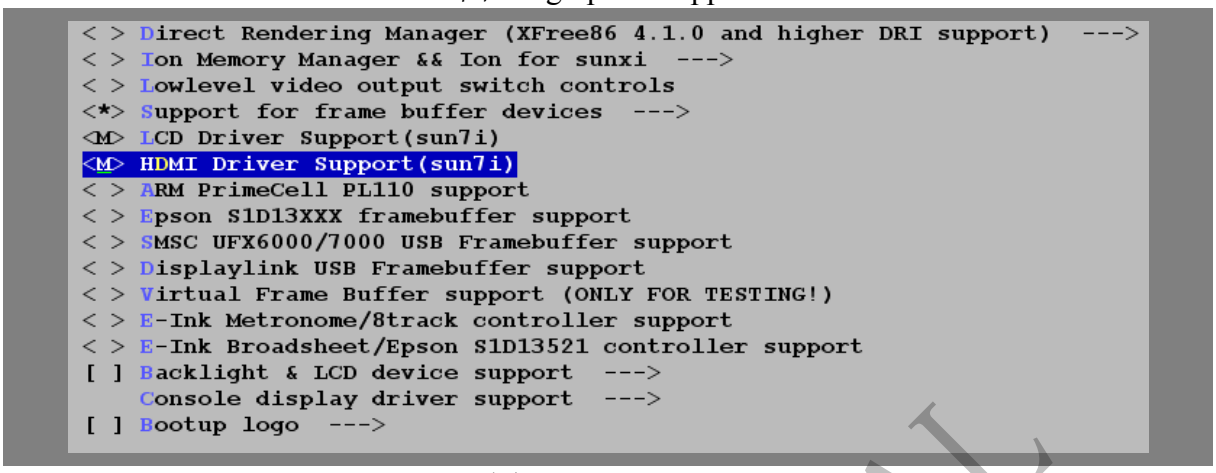

图 9　hdmi driver

- 耳机检测驱动配置

在 android 系统中，支持耳机动态检测。耳机检测驱动配置如图 10~图 12 所示。

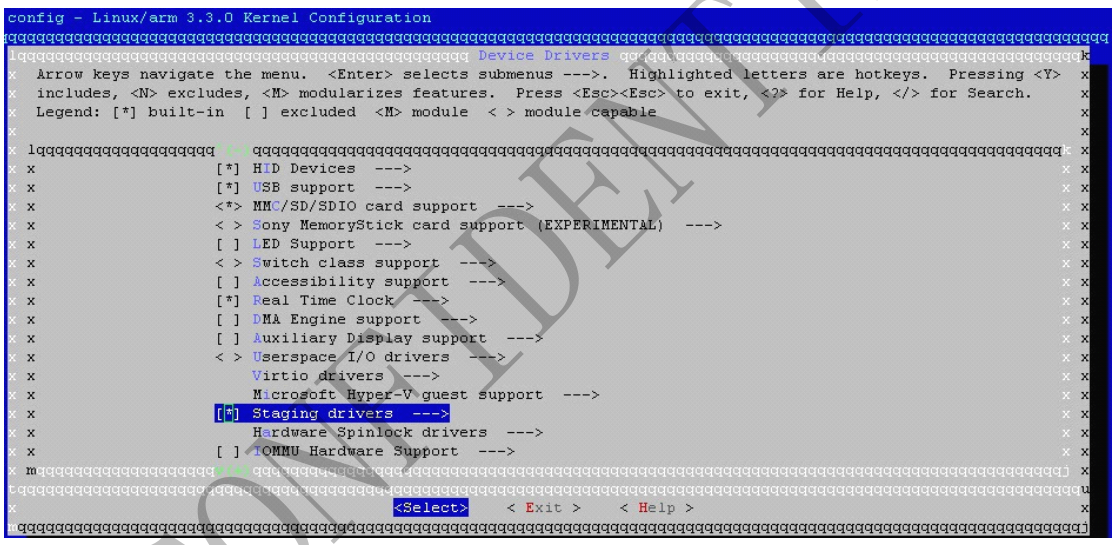

图 10 staging driver

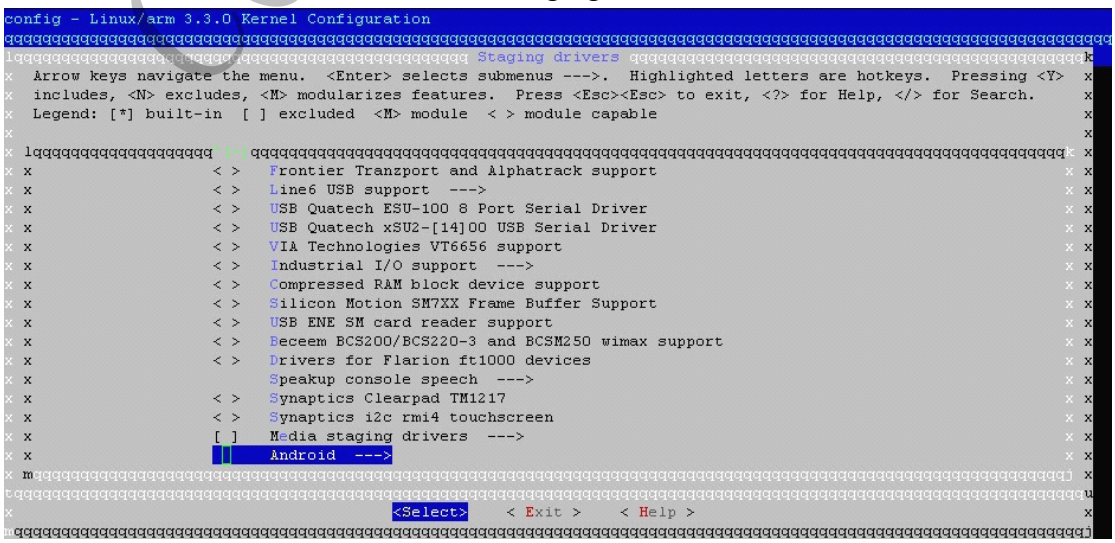

图 11.android 配置

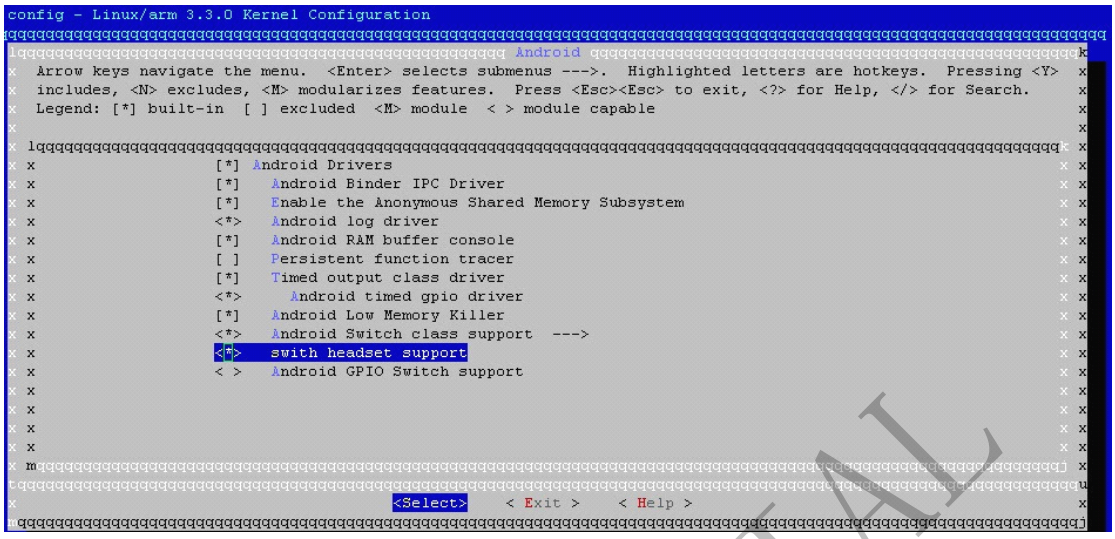

图 12 耳机检测驱动配置

2.4.2. Sysconfig.fex 配置

配置文件的位置：\lichee\tools\pack\chips\sun7i\configs\android\wing-XXX 目录下。音频中需要配置的有 audiocodec，i2s，pcm，spdif 四个音频驱动。

2.4.2.1. Audiocodec 配置

[audio_para]
audio_used = 1
audio_pa_ctrl_used = 0
audio_pa_ctrl = port:PB03<1><default><default><1>

配置项	配置项含义
audio_used =xx	Audiocodec 是否使用， 1：打开（默认）0：关闭
audio_pa_ctrl_used = xx	控制 audio_pa_ctrl 是否使用 1：打开 0：关闭（默认）
audio_pa_ctrl=xx	喇叭的 gpio 口控制。一般用于控制功放使能端。 管脚请根据实际情况分配。

2.4.2.2. spdif 配置

配置项	配置项含义
spdif_used=xx	xx 为 1 时加载该模块，为 0 时不加载[default value: 1]

spdif_dout	spdid out 的 GPIO 配置
spdif_din	spdid in 的 GPIO 配置
spdif_mclk	Spdif mclk 信号的 GPIO 配置

2.4.2.3. I2s 配置

I2s 可以配置成 pcm 模式，也可以配置成 i2s 模式。

配置项	配置项含义
i2s_used	xx 为1时加载该模块，为0时不加载[default value: 1]
i2s_channel	声道控制[default value: 2]
i2s_master	主从模式配置[1: SND_SOC_DAIFMT_CBM_CFM(codec clk & FRM master) SOC as slave & codec as master] [4: SND_SOC_DAIFMT_CBS_CFS(codec clk & FRM slave) SOC as master & codec as slave]
i2s_select	Pcm 和 i2s 模式选择[0：pcm 1：i2s]
audio_format	音频格式选择 [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	音频信号模式选择 [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)]
over_sample_rate	音频过采样率选择：support 128fs/192fs/256fs/384fs/512fs/768fs [default value: 256]
sample_resolution	采样精度选择：16bits/20bits/24bits[default value: 16]
word_select_size	位宽选择: 16bits/24bits/32bits[default value: 32]
pcm_sync_period	PCM 周期长度选择16/32/64/128/256 BCLKs [default value: 256]
msb_lsb_first	数据位模式选择 [0: msb first; 1: lsb first][default value: 0]
sign_extend	标志位扩展模式 [0: zero pending; 1: sign extend][default value: 0]
slot_index	时间片索引 [0: the 1st slot - 3: the 4th slot][default value: 0]
slot_width	时间片宽度 [8 bit width / 16 bit width][default value: 16]
frame_width	帧宽度 [0: long frame = 2 BCLK width; 1: short frame = 1 BCLK

	width][default value: 1]
tx_data_mode	Tx 数据传输模式[0: 16bit linear PCM; 1: 8bit linear PCM; 2: 8bit u-law; 3: 8bit a-law][default value: 0]
rx_data_mode	Rx 数据传输模式[0: 16bit linear PCM; 1: 8bit linear PCM; 2: 8bit u-law; 3: 8bit a-law][default value: 0]
i2s_mclk	I2sMCLK 信号的 GPIO 配置
i2s_bclk	I2sBCLK 信号的 GPIO 配置
i2s_lrclk	I2sLRCK 信号的 GPIO 配置
i2s_dout0	I2S out0的 GPIO 配置
i2s_dout1	暂不使用
i2s_dout2	暂不使用
i2s_dout3	暂不使用
i2s_din	I2sIN 信号的 GPIO 配置

2.4.2.4. pcm 配置

pcn 可以配置成 pcm 模式，也可以配置成 i2s 模式。

配置项	配置项含义
pcm_used	xx 为1时加载该模块，为0时不加载[default value: 1]
pcm_channel	声道控制[default value: 2]
Pcm_master	主从模式配置[1: SND_SOC_DAIFMT_CBM_CFM(codec clk & FRM master) SOC as slave & codec as master] [4: SND_SOC_DAIFMT_CBS_CFS(codec clk & FRM slave) SOC as master & codec as slave]
pcm_select	Pcm 和 i2s 模式选择[1：pcm 0：i2s]
audio_format	音频格式选择 [default value: 4] [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	音频信号模式选择 [default value: 3] [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)]
over_sample_rate	音频过采样率选择：support 128fs/192fs/256fs/384fs/512fs/768fs [default value: 512]

sample_resolution	采样精度选择：16bits/20bits/24bits[default value: 16]
word_select_size	位宽选择: 16bits/24bits/32bits[default value: 32]
pcm_sync_period	PCM 周期长度选择16/32/64/128/256 BCLKs [default value: 64]
msb_lsb_first	数据位模式选择 [0: msb first; 1: lsb first][default value: 0]
sign_extend	标志位扩展模式 [0: zero pending; 1: sign extend][default value: 0]
slot_index	时间片索引 [0: the 1st slot - 3: the 4th slot][default value: 0]
slot_width	时间片宽度 [8 bit width / 16 bit width][default value: 16]
frame_width	帧宽度 [0: long frame = 2 BCLK width; 1: short frame = 1 BCLK width][default value: 1]
tx_data_mode	Tx 数据传输模式[0: 16bit linear PCM; 1: 8bit linear PCM; 2: 8bit u-law; 3: 8bit a-law][default value: 0]
rx_data_mode	Rx 数据传输模式[0: 16bit linear PCM; 1: 8bit linear PCM; 2: 8bit u-law; 3: 8bit a-law][default value: 0]
pcm_bclk=xx	pcmBCLK 信号的 GPIO 配置
pcm_lrclk =xx	pcmLRCK 信号的 GPIO 配置
pcm_dout	pcm out 的 GPIO 配置
pcm_din	pcmIN 信号的 GPIO 配置
pcm_mclk =xx	暂不使用
pcm_bclk=xx	pcmBCLK 信号的 GPIO 配置
pcm_lrclk =xx	pcmLRCK 信号的 GPIO 配置

3. 音频模块体系结构描述

在 a20 中有 4 个音频设备驱动，分别为 audiocodec，hdmiaudio，spdif，pcm/i2s 以及一个耳机检测驱动 switch。在 a20 中，基本的音频框架如图 4.a20 音频框架所示。

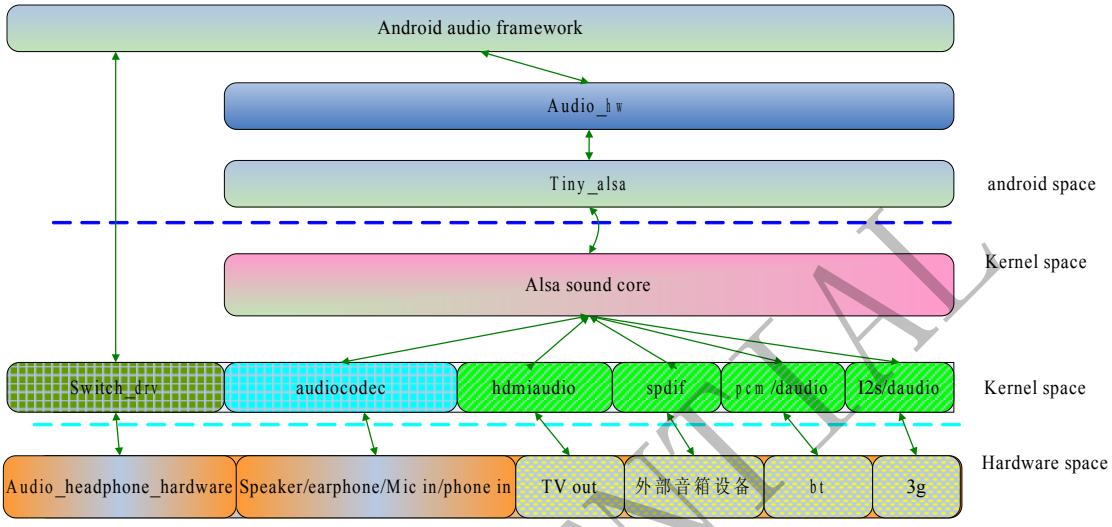

图 13 a20 音频框架

4. 接口描述

在音频框架中，audiocodec 属于模拟音频部分，hdmiaudio，spdif，i2s/pcm 属于数字音频。其中 audiocodec 在 3g 音频通话中支持模拟音频通路和通话录音功能接口。i2s 可以配置成 pcm 和 i2s 模式。Hdmiaudio，spdif 支持 raw data 模式。耳机检测驱动 switch 支持 android 标准的耳机检测接口。

4.1. Audiocodec 接口描述

在模拟音频驱动 audiocodec 中，支持 ADC 录音，DAC 播放，模拟音频通路。支持四路音频输入（mic1，mic2，line in， fm），两路音频输出（phone out， headphone）。如图 13.audiocodec 音频硬件通道所示。Audiocodec 中的音频通道接口也是根据图 13.audiocodec 音频硬件通道封装的。

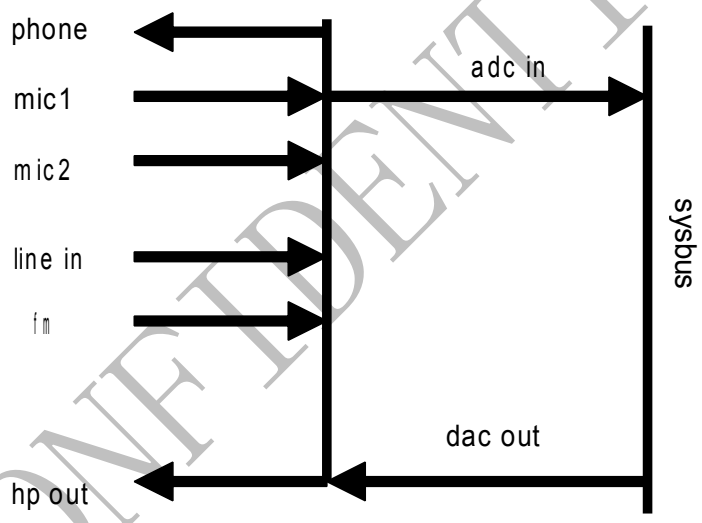

图 14.audiocodec 音频硬件通道

4.2. Hdmiaudio 接口描述

Hdmiaudio 中，支持 raw data 和 pcm data 模式。
当音频 channel 设置成 channels = 4 的时候，hdmiaudio 会相应设置成 raw data 模式。当音频 channel 设置成 channels = 2 或者 1 的时候，hdmiaudio 会设置成 pcm data 模式。
Hdmiaudio 是一个独立的音频驱动，接口支持 alsa lib 中的标准接口，不在这里一一列举。

4.3. spdif 接口描述

spdif 中，支持 raw data 和 pcm data 模式。
当音频 channel 设置成 channels = 4 的时候，spdif 会相应设置成 raw data 模式。当音频

channel 设置成 channels = 2 或者 1 的时候，spdif 会设置成 pcm data 模式。
spdif 是一个独立的音频驱动，接口支持 alsa lib 中的标准接口，不在这里一一列举。

4.4. I2s 接口描述

具体参考 2.4.2.2 中 i2s 的 sysconfig 配置。

4.5. switch 接口描述

Switch 接口支持 android 标准的耳机检测。支持 3 段耳机和四段耳机的插拔检测功能。不再描述。

5. 模块开发 demo

音频的外部接口跟普通的驱动不同。音频的 application 需要额外的 alsa-lib 进行外部接口的封装。在 android2.3.4 中，用的是 small alsa 应用库，而在 android4.0 以后，采用 tiny alsa 应用库进行外部接口的封装。所有的接口都是音频的标准接口。在这里不一一罗列。下面给出播放和录音的应用.

写一个音频应用程序涉及到以下几步

- opening the audio device
- set the parameters of the device
- receive audio data from the device or deliver audio data to the device
- close the device

在 a20 中，涉及 4 个音频驱动，alsa 的 lib 库支持任何一个音频驱动。请参考最小 playback 应用和录音应用以及 mixer 接口的使用方法。

demo 采用 tiny_alsa 库，可以从\android4.1\external\tinyalsa 中获得。

5.1. 最小的 playback 应用

```
/*
 * spdif test
 * play_high_rate-1633.c
 * (C) Copyright 2010-2016
 * reuuimllatech Technology Co., Ltd. <www.reuuimllatech.com>
 * huangxin <huangxin@reuuimllatech.com>
 *
 * some simple description for this code
 *
 * This program is free software; you can redistribute it and/or
 * modify it under the terms of the GNU General Public License as
 * published by the Free Software Foundation; either version 2 of
 * the License, or (at your option) any later version.
 *
 */

#include <include/tinyalsa/asoundlib.h>
#include <stdio.h>
#include <stdlib.h>
#include <stdint.h>
#include <string.h>
```

```
#define ID_RIFF 0x46464952
#define ID_WAVE 0x45564157
#define ID_FMT    0x20746d66
#define ID_DATA 0x61746164

#define FORMAT_PCM 1

struct wav_header {
        uint32_t riff_id;
        uint32_t riff_sz;
        uint32_t riff_fmt;
        uint32_t fmt_id;
        uint32_t fmt_sz;
        uint16_t audio_format;
        uint16_t num_channels;
        uint32_t sample_rate;
        uint32_t byte_rate;
        uint16_t block_align;
        uint16_t bits_per_sample;
        uint32_t data_id;
        uint32_t data_sz;
};

void play_sample(FILE *file, unsigned int device, unsigned int channels,
                                unsigned int rate, unsigned int bits);

int main(int argc, char **argv)
{
        FILE *file;
        struct wav_header header;
        unsigned int device = 0;

        if (argc < 2) {
                fprintf(stderr, "Usage: %s file.wav [-d device]\n", argv[0]);
                return 1;
        }

       file = fopen(argv[1], "rb");
        if (!file) {
                fprintf(stderr, "Unable to open file '%s'\n", argv[1]);
                return 1;
```

```
    }

    /* parse command line arguments */
    argv += 2;
    while (*argv) {
        if (strcmp(*argv, "-d") == 0) {
            argv++;
            device = atoi(*argv);
        }
        argv++;
    }

    fread(&header, sizeof(struct wav_header), 1, file);

    if ((header.riff_id != ID_RIFF) ||
        (header.riff_fmt != ID_WAVE) ||
        (header.fmt_id != ID_FMT) ||
        (header.audio_format != FORMAT_PCM) ||
        (header.fmt_sz != 16)) {
        fprintf(stderr, "Error: '%s' is not a PCM riff/wave file\n", argv[1]);
        fclose(file);
        return 1;
    }

    /* install signal handler and begin capturing */
    play_sample(file, device, header.num_channels, header.sample_rate,
                header.bits_per_sample);
    fclose(file);

    return 0;
}

void play_sample(FILE *file, unsigned int device, unsigned int channels,
                 unsigned int rate, unsigned int bits)
{
    struct pcm_config config;
    struct pcm *pcm0;
    char *buffer;
    int size;
    int num_read;
   int dtime = 15;
   int loop_time = 0;
```

```
    int i=0;

    config.channels = channels;
    config.rate = rate;
    config.period_size = 1024;
    config.period_count = 4;
    if (bits == 32)
        config.format = PCM_FORMAT_S32_LE;
    else if (bits == 16)
        config.format = PCM_FORMAT_S16_LE;
    config.start_threshold = 0;
    config.stop_threshold = 0;
    config.silence_threshold = 0;

   /*0 is audiocodec, 1 is hdmiaudio, 2 is spdif, 3 is i2s，4 is pcm*/
    pcm0 = pcm_open(3, device, PCM_OUT, &config);
    if (!pcm0 || !pcm_is_ready(pcm0)) {
        fprintf(stderr, "Unable to open PCM device %u (%s)\n", device,
pcm_get_error(pcm0));
        return;
    }

    size = pcm_get_buffer_size(pcm0);
    buffer = malloc(size);
    if (!buffer) {
        fprintf(stderr, "Unable to allocate %d bytes\n", size);
        free(buffer);
        pcm_close(pcm0);
        return;
    }

    printf("Playing sample: %u ch, %u hz, %u bit\n", channels, rate, bits);
    loop_time = dtime*rate/1024;
    printf("loop_time:%d\n", loop_time);
    do {
        num_read = fread(buffer, 1, size, file);
        if (num_read > 0) {
            if (pcm_write(pcm0, buffer, num_read)) {
                fprintf(stderr, "Error playing sample\n");
                  break;
            }
              if (feof(file)) {
```

```
                    fseek(file, 0L, SEEK_SET);
                    //break;
                }
            }
            i++;
            if(i > loop_time)
                break;
        } while ((num_read > 0));

        free(buffer);
        pcm_close(pcm0);
}
```

5.2. 最小的 capture 应用

```
/* tinycap.c
**
** Copyright 2011, The Android Open Source Project
**
** Redistribution and use in source and binary forms, with or without
** modification, are permitted provided that the following conditions are met:
**         * Redistributions of source code must retain the above copyright
**           notice, this list of conditions and the following disclaimer.
**         * Redistributions in binary form must reproduce the above copyright
**           notice, this list of conditions and the following disclaimer in the
**           documentation and/or other materials provided with the distribution.
**         * Neither the name of The Android Open Source Project nor the names of
**           its contributors may be used to endorse or promote products derived
**           from this software without specific prior written permission.
**
*/

#include <include/tinyalsa/asoundlib.h>
#include <stdio.h>
#include <stdlib.h>
#include <stdint.h>
#include <signal.h>

#define ID_RIFF 0x46464952
#define ID_WAVE 0x45564157
#define ID_FMT    0x20746d66
#define ID_DATA 0x61746164
```

```
#define FORMAT_PCM 1

struct wav_header {
	uint32_t riff_id;
	uint32_t riff_sz;
	uint32_t riff_fmt;
	uint32_t fmt_id;
	uint32_t fmt_sz;
	uint16_t audio_format;
	uint16_t num_channels;
	uint32_t sample_rate;
	uint32_t byte_rate;
	uint16_t block_align;
	uint16_t bits_per_sample;
	uint32_t data_id;
	uint32_t data_sz;
};

int capturing = 1;
static char filename[64] = "record1.wav";

unsigned int capture_sample(FILE *file, unsigned int device,
                            unsigned int channels, unsigned int rate,
                            unsigned int bits);


void sigint_handler(int sig)
{
	capturing = 0;
}

int main(int argc, char **argv)
{
	FILE *file;
		struct mixer *mixer;
	struct wav_header header;
	unsigned int device = 0;
	unsigned int channels = 2;
	unsigned int rate = 44100;
	unsigned int bits = 16;
	unsigned int frames;
```

```
rate = atoi(argv[1]);
channels = atoi(argv[2]);

if (argc < 2) {
        fprintf(stderr, "Usage: %s file.wav [-d device] [-c channels] "
                        "[-r rate] [-b bits]\n", argv[0]);
        return 1;
}

file = fopen(filename, "wb");
if (!file) {
        fprintf(stderr, "Unable to create file '%s'\n", argv[1]);
        return 1;
}

/* parse command line arguments */
argv += 2;
while (*argv) {
        if (strcmp(*argv, "-d") == 0) {
                argv++;
                device = atoi(*argv);
        } else if (strcmp(*argv, "-c") == 0) {
                argv++;
                channels = atoi(*argv);
        } else if (strcmp(*argv, "-r") == 0) {
                argv++;
                rate = atoi(*argv);
        } else if (strcmp(*argv, "-b") == 0) {
                argv++;
                bits = atoi(*argv);
        }
        argv++;
}

header.riff_id = ID_RIFF;
header.riff_sz = 0;
header.riff_fmt = ID_WAVE;
header.fmt_id = ID_FMT;
header.fmt_sz = 16;
header.audio_format = FORMAT_PCM;
header.num_channels = channels;
header.sample_rate = rate;
```

```
    header.byte_rate = (header.bits_per_sample / 8) * channels * rate;
    header.block_align = channels * (header.bits_per_sample / 8);
    header.bits_per_sample = bits;
    header.data_id = ID_DATA;

    /* leave enough room for header */
    fseek(file, sizeof(struct wav_header), SEEK_SET);

   mixer = mixer_open(0);
    if (!mixer) {
        fprintf(stderr, "Failed to open mixer\n");
        return EXIT_FAILURE;
    }
    /* install signal handler and begin capturing */
    signal(SIGINT, sigint_handler);
    frames = capture_sample(file, device, header.num_channels,
                                    header.sample_rate, header.bits_per_sample);
    printf("Captured %d frames\n", frames);

    /* write header now all information is known */
    header.data_sz = frames * header.block_align;
    fseek(file, 0, SEEK_SET);
    fwrite(&header, sizeof(struct wav_header), 1, file);
   mixer_close(mixer);
    fclose(file);

    return 0;
}

unsigned int capture_sample(FILE *file, unsigned int device,
                                    unsigned int channels, unsigned int rate,
                                    unsigned int bits)
{
    struct pcm_config config;
    struct pcm *pcm;
    char *buffer;
    unsigned int size;
    unsigned int bytes_read = 0;
   int dtime = 1;
   int loop = 0;

    config.channels = channels;
```

```
config.rate = rate;
config.period_size = 1024;
config.period_count = 4;
if (bits == 32)
        config.format = PCM_FORMAT_S32_LE;
else if (bits == 16)
        config.format = PCM_FORMAT_S16_LE;
config.start_threshold = 0;
config.stop_threshold = 0;
config.silence_threshold = 0;

pcm = pcm_open(0, device, PCM_IN, &config);
if (!pcm || !pcm_is_ready(pcm)) {
        fprintf(stderr, "Unable to open PCM device (%s)\n",
                        pcm_get_error(pcm));
        return 0;
}

size = pcm_get_buffer_size(pcm);
buffer = malloc(size);
if (!buffer) {
        fprintf(stderr, "Unable to allocate %d bytes\n", size);
        free(buffer);
        pcm_close(pcm);
        return 0;
}

printf("Capturing sample: %u ch, %u hz, %u bit\n", channels, rate, bits);


dtime = 12*60*60;
loop = dtime*rate/1024;
printf("loop:%d\n", loop);
for(loop;loop > 0;loop--)
{
      int ret = 0;
      //printf("hello,size:%d\n",size);
      ret = pcm_read(pcm, buffer, size);
      if (fwrite(buffer, 1, size, file) != size) {
              fprintf(stderr,"Error capturing sample\n");
              break;
       }
```

```
            bytes_read += size;
        }

        free(buffer);
        pcm_close(pcm);
        return bytes_read / ((bits / 8) * channels);
}
```

6. Declaration

This(A20 音频模块开发说明) is the original work and copyrighted property of Allwinner Technology ("Allwinner"). Reproduction in whole or in part must obtain the written approval of Allwinner and give clear acknowledgement to the copyright owner.

The information furnished by Allwinner is believed to be accurate and reliable. Allwinner reserves the right to make changes in circuit design and/or specifications at any time without notice. Allwinner does not assume any responsibility and liability for its use. Nor for any infringements of patents or other rights of the third parties which may result from its use. No license is granted by implication or otherwise under any patent or patent rights of Allwinner. This datasheet neither states nor implies warranty of any kind, including fitness for any particular application.