Audio Processing in Embedded Real-Time Linux Systems
Sa’id Muhammad SA’ID1, Mustafa GÜNDÜZALP2
1Nigerian College of Aviation Technology, Zaria, Nigeria
2Department of Electrical and Electronic Engineering, Yasar University, Izmir, Turkey

Abstract-- Embedded systems are fundamental in the development of ever more capable electronic and computing systems in existence today commonly in household electronics, mobile phones, cars, aeroplanes etc. Importance of audio cannot be over emphasized with regards to embedded real-time systems. A lot of researches have been done in this regard, however documentation remains scanty. This paper therefore describes audio capture and playback in view to better understand audio processing in Linux and specifically ARM powered embedded systems.

Keywords-- Embedded system, ALSA, GTK, ARM Processor, real-time, audio capture and playback

I. INTRODUCTION

Sound process in computers involve an analog audio signal, converted to digital, stored or processed by a DSP unit or a general-purpose CPU, and finally outputted as analog after reverse conversion. This simple process gets complex with variations of hardware.

These include the previously dominant Open Sound System (OSS) which allows applications like media players or web browsers to access the audio driver directly, and later the Advanced Linux Sound Architecture (ALSA) that has replaced it. [4]

ALSA have better hardware support. It allows improved user control and better support for multiple audio devices.

II. ALSA

The Advanced Linux Sound Architecture (ALSA) is the audio framework used in newer current Linux kernels for audio support. ALSA consists of a set of kernel drivers, an application programming interface (API) library and utility programs for supporting sound in Linux. It is backwards compatible with the older OSS.

III. TOOLS INCLUDED IN ALSA (ALSA-UTILS)

- alsacnf - the ALSA driver configurator script
- alsactl - an utility for soundcard settings management
- aplay/arecord - an utility for the playback / capture of .wav,.voc,.au files
IV. TYPICAL AUDIO APPLICATION

A typical audio application involves opening the audio device, setting parameters by which the program will run and running the actual program which involves receiving from or sending audio data to a device. [10]

```c
open_the_device();
set_the_parameters_of_the_device();
while(!done) {
    /* one or both of these */
    receive_audio_data_from_the_device();
    deliver_audio_data_to_the_device();
}
close_the_device
```

V. LISTING ALSA SOUND CARDS

The first thing to do in developing an ALSA application is to list all the available sound cards/devices on the machine. ALSA has some functions to list all available sound cards/devices. One of such functions is `snd_card_next()`.

Therefore, a loop is created with the pointer at an integer value of -1. The function will then change the value of `int` to the number of the first card/device in the system. This continues to increment until there is no more sound cards to show, then ALSA sets `int` back to -1.

```c
#include <stdio.h>
#include <string.h>
#include <alsa/asoundlib.h>
int main (int argc, char **argv)
{
    register int err;
    int cardNum, totalCards;
    totalCards = 0;
    cardNum = -1;
    for (;;)
    {
        if ((err = snd_card_next(&cardNum)) < 0) {
            printf("Next card not available: %s\n", snd_strerror(err));
            break;
        }
        printf("ALSA found %i cards\n", totalcards);
        snd_config_update_free_global();
    } [4]
```

After including all necessary libraries, that’s the `stdio`, `string` and `asoundlib` headers, variables were then defined. `cardNum` is the card number at the instance initially set at -1 and `totalCards` is the total number of cards found set to 0.

An “if” block is used for the `snd_card_next` function. When card number is -1, ALSA will fetch the first card. This continues until there are no more cards, then it sums up all cards found and print out `totalCards` found.

ALSA allocates memory space to load its configuration file when a handle is called. To free the memory and unload the information, the function `snd_config_update_free_global` is called with null value.

VI. SETTING HARDWARE PARAMETERS

The following parameters can be set for audio application based on card type or audio chip capabilities.

- Sample rate (8Khz, 22Khz or 44.1Khz)
- Playback bit resolution (8 bit, 16 bit, 32 bit)
- Channels (mono, stereo or multi-channel)

ALSA has a function that enables the direct settings of hardware parameters, that is the `snd_pcm_set_params()`.

With the device parameters set, a Linux audio driver can be developed using ALSA for various applications which will be detailed in the next chapter. A comprehensive GUI designed with GTK will also accompany the driver for easy access.

VII. OPENING DEVICE AND SETTING PARAMETERS

The following opens the default PCM device for playback and set some parameters using newest ALSA API

```c
#define ALSA_PCM_NEW_HW_PARAMS_API
#include <alsa/asoundlib.h>
int main() {
    int rc;
    snd_pcm_t *handle;
    snd_pcm_hw_params_t *params;
    unsigned int val;
    int dir;
    snd_pcm_uframes_t frames;
    // to open PCM device for playback
    rc = snd_pcm_open(&handle,"default",SND_PCM_STREAM_PLAYBACK, 0);
```
if (rc < 0) {
    fprintf(stderr,"unable to open pcm device:%s\n",
            snd_strerror(rc));
    exit(1);
} 
// Allocating default hardware parameters object
snd_pcm_hw_params_alloca(&params);
//changing to desired hardware parameters
snd_pcm_hw_params_set_access(handle, params,
SND_PCM_ACCESS_RW_INTERLEAVED);
snd_pcm_hw_params_set_format(handle, params,
SND_PCM_FORMAT_S16_LE);
snd_pcm_hw_params_set_channels(handle,
params, 2);
val = 44100;
snd_pcm_hw_params_set_rate_near(handle,para
ms, &val, &dir);
    frames = 32;
snd_pcm_hw_params_set_period_size_near(hand
le, params, &frames, &dir);
// Writing the parameters to the driver
rc = snd_pcm_hw_params(handle, params);
if (rc < 0) {
    fprintf(stderr,"unable to set hw
parameters: %s",snd_strerror(rc));
    exit(1);
} 
return 0;
"[4]

This will open a default playback device, initially with
default hardware parameters, the change them to
[interleaved – stereo (2-channel) – signed 16 bit little
dian – 44100 bit rate – 32 frame size]. This completes
the device initialization. [16]

VIII. AUDIO PLAYBACK

Audio playback is achieved by reading stream audio data
from memory and writing it to the PCM device. For this
section, a 5sec data is read from input and written to the
default PCM device. This follows after opening the device
and setting its parameters as shown above.

"#define ALSA_PCM_NEW_HW_PARAMS_API
#include <alsa/asoundlib.h>
int main() {
    long loops;
    int rc;
    int size;
    snd_pcm_t *handle;
    snd_pcm_hw_params_t *params;
    unsigned int val;
    int dir;
    snd_pcm_uframes_t frames;
    char *buffer;

    *****************************
    snd_pcm_hw_params_get_period_size(params,
    &frames, &dir);
    size = frames * 4; /* 2 bytes/sample, 2 channels */
    buffer = (char *) malloc(size);
    snd_pcm_hw_params_get_period_time(params,&
    val, &dir);

    // 5 seconds in microseconds/period time
    loops = 5000000 / val;
    while (loops > 0) {
        loops--;
        rc = read(0, buffer, size);
        if (rc == 0) {
            fprintf(stderr, "end of file on input
\n");break;
        } else if (rc != size) {
            fprintf(stderr, "short read: read %d bytes\n",
            rc);
            rc = snd_pcm_writei(handle, buffer, frames);
            if (rc == -EPIPE) {
                fprintf(stderr, "underrun occurred
\n";
                snd_pcm_prepare(handle);
            } else if (rc < 0) {
                fprintf(stderr,"error
\n",snd_strerror(rc));
                from
writei:%s
\n",snd_strerror(rc));
            } else if (rc != (int)frames) {
                fprintf(stderr,"short write, write %d rames\n",
                rc);
            }
        }
    }
}

The error code EPIPE means xrun (underrun for
playback or overrun for capture). Underrun happens
when an application does not feed new samples in time to
alsa-lib while overrun happens when an application does
not take new captured samples in time from alsalib. [14]

After initialization, sound samples are written to the
sound card to produce playback. Bytes are read from
standard input for one period, then written to sound card
continuously for 5sec.
IX. AUDIO CAPTURE

The same initialization procedure applies for audio capture or sound recording, but instead of opening the device for playback, it is opened for capture with the syntax

```c
rc = snd_pcm_open(&handle,"default", SND_PCM_STREAM_CAPTURE, 0)
```

The audio capture program is also similar to the playback, the program is as below

```c
#define ALSA_PCM_NEW_HW_PARAMS_API
#include <alsa/asoundlib.h>
int main() {
    long loops;
    int rc;
    int size;
    snd_pcm_t *handle;
    snd_pcm_hw_params_t *params;
    unsigned int val;
    int dir;
    snd_pcm_uframes_t frames;
    char *buffer;
    // ***************
    snd_pcm_hw_params_get_period_size(params, &frames, &dir);
    size = frames * 4; /* 2 bytes/sample, 2 channels */
    buffer = (char *) malloc(size);
    snd_pcm_hw_params_get_period_time(params, &val, &dir);
    loops = 5000000 / val;
    while (loops > 0) {
        loops--;
        rc = snd_pcm_readi(handle, buffer, frames);
        if (rc == -EPIPE) {
            fprintf(stderr, "overrun occurred\n");
            snd_pcm_prepare(handle);
        } else if (rc < 0) {
            fprintf(stderr,"error from read: \n", snd_strerror(rc));
        } else if (rc != (int)frames) {
            fprintf(stderr, "short read, read %d frames\n", rc);
        } rc = write(1, buffer, size);
        if (rc != size) {
            fprintf(stderr,"short write: wrote %d bytes\n", rc);
        }
        rc = snd_pcm_drain(handle);
        snd_pcm_close(handle);
        free(buffer);
        return 0;
    } /* 4
```

When the PCM stream is opened, the capture mode is specified as SND_PCM_STREAM_CAPTURE. Within the main processing loop, samples are read from the sound hardware using snd_pcm_readi and written to standard output using write. Errors and overrun are checked like in the play program.

For both audio capture and playback, the data stream has to be directed to a file. The mixer tool is used to set the level of the recording source. For ALSA, there is an in package ALSAmixer, a textmode based mixer program for ALSA soundcard drivers.

X. SUMMARY

The paper will serve as a well documented guide for a beginner in ALSA programming. It describes the procedure of installing an ALSA driver on a Linux system, basic configuration and coding an ALSA progr to run on an embedded Linux system.

The Advanced Linux Sound Architecture (ALSA) had replaced the Open Sound System (OSS) as an effective and easier sound system tool. The ease with which it is employed in various systems with different hardware composition shows its flexibility. ALSA can emulate the previous OSS architectures drivers and may also be configured as a dummy driver on systems with no sound card installed.
Acknowledgment

This work was partly developed from the MSc. Thesis of the primary author under the supervision of the co-author at Yaşar University, Izmir, Turkey. We would also like to thank KENTKART Company, Izmir, Turkey for their technical support in this work.

REFERENCES