Implementation of Various Sampling Rates through Linux Audio Codec Driver

Abstract: An audio codec, being a product originally conceived in industry, can be said to have had a development path, where user demands interacted with industry competition, in order to produce the next generation of soundcard devices. As such, the soundcard has evolved to a product, that most of today’s consumer PC users have very specific demands from they expect to control the soundcard using their favorite ‘media player’ or ‘recorder’ audio software from the PC; while the soundcard interfaces with audio equipment like speakers or amplifiers. For professional users, the character of audio software’ and audio equipment’ may encompass far more specialized and complex systems however, the expectations of the users in respect to basic interaction with this part of the system is still the same high-level, PC software control of the audio reproduced or captured on the hardware. This project outlines how an Integrated Power Management/Audio Codec chip can be demonstrated to behave as a full duplex, mono, 8-bit 44.1kHz soundcard, through an implementation of: a PC audio driver for ALSA (Advanced Linux Sound Architecture).

The main contribution of this paper is to bring a holistic aspect to the discussion on the topic of implementation of soundcards also by referring to open-source driver, processor code and test methods; and outline a complete implementation of an open yet functional soundcard system.

Keywords: Audio codec, ALSA, Linux, Driver, sound card, Audio.
1. Introduction:

A contemporary PC user, typically expects a sound card to be a piece of hardware, that can be manipulated by audio software (most typically exemplified by media players) and allows interfacing of the PC to audio reproduction and/or recording equipment. As such, a sound card can be considered to be a system, that encompasses design decisions on both hardware and software levels that also demand a certain understanding of the architecture of the target PC operating system. The main contribution of this paper is to bring a holistic aspect to the discussion on the topic of implementation of soundcards also by referring to open-source driver, processor code and test methods; and outline a complete implementation of an open yet functional soundcard system. A development of a soundcard thus requires, to some extent, an interdisciplinary approach requiring knowledge of both electronics and software engineering, along with operating system architecture. An open soundcard may bring actual benefits to electronic instrument designers, beyond opportunity for technical study: where it is used to modulate a digital audio signal in real time. Usual approach would be to read the data as a serial port at this limits the analog bandwidth (~ 5 kHz) and it forces the user to code a conversion data could be received directly as a 44.1 kHz audio signal in full audio analog bandwidth.

2. Previous Work:

An open soundcard implementations couldn’t provide a basis for the development here: the Linux kernel contains many open soundcard drivers, but written for commercial hardware. Thus, this project’s basis is mostly in own previous work demonstrates legacy hardware controlled by PC software; Modern operating systems address this issue by providing a driver architecture; where, in programming a driver, the programmer gains a more fine-grained temporal control. In the context of the open GNU/Linux operating system, acquaintance with its current low-level audio library ALSA is thus necessary for implementation of soundcard drivers. In context of software that the term frees or open will be applied in this project. To begin with, the driver is developed on Ubuntu GNU/Linux operating system; with the main corresponding tool for development, gcc being likewise open. The audio framework for Linux, ALSA, follows the same license and the main high-level, user audio
programs used, Audacity and arecord, are likewise open. The processor used in the platform is typically of the OMAP™ 3 architecture applications Processor.

3. Basic Structure of ALSA:
Advanced Linux Sound Architecture (known by the acronym ALSA) is a free and open source software framework providing an API for device drivers for sound cards. As such, it is a Linux kernel component. From the 2.6.x kernels onward ALSA has been the default Linux sound architecture, replacing its predecessor OSS. The Advanced Linux Sound Architecture (ALSA) provides audio and MIDI functionality to the Linux operating system. ALSA has the following significant features:

- Efficient support for all types of audio interfaces, from consumer soundcards to professional multichannel audio interfaces.
- Fully modularized sound drivers.
- User space library (alsa-lib) to simplify application programming and provide higher level functionality.

3.1. Benefits of Alsa for Audio Codec:
From the 2.6.x kernels onward ALSA has been the default Linux sound architecture, replacing its predecessor OSS. Android uses 2.6 and higher Linux kernel. So, if you want your codec to work with any of the Android systems, it has to be ALSA compliant. The following components are supported by ALSA:

- PCM interface: Managing digital audio capture and playback.
- Controls interface: General purpose facility for managing registers of sound cards and querying available devices.
- Raw MIDI interface: Access to a MIDI bus on a sound card. Works directly with the MIDI events. Protocol and timing management up to the programmer.
- Mixer interface: Controls the devices on sound cards that route signals and control volume levels. Built on top of the control interface
- Timer interface: Access to timing hardware on sound cards, used for synchronizing sound events.
- Sequencer interface: A higher level interface for MIDI programming and sound synthesis than the raw MIDI interface. Handles much of the MIDI protocol and timing.
- Hardware dependent Device
4. Architecture Design:
The audio sub chip in the audio codec has two interfaces: PCM for voice signaling and I2S for audio signaling. The Bluetooth interface is also available. This uses the same clock and sync signals as for the PCM interface. Both ports can function as master or slave. Sampling modes available are Voice channel with 8-kHz or 16-kHz sampling modes Audio channel with 8, 11.025, 12, 16, 22.05, 24, 32, 44.1, or 48-kHz sampling modes. 96 kHz is supported on the RX path voice codec with MCLK = 26MHz. The audio phase-locked loop (PLL) supports 26-MHz, 19.2-MHz, or 38.4-MHz ability to work with two ports with the clock frequencies 19.2 MHz or 26 MHz, or with a single TDM port with the frequencies 19.2 MHz, 26 MHz, or 38.4 MHz. The voice PCM interface is available when the system clock is 26MHz. The audio supports common features like pop noise reduction, side tone functionality, bass boost function, uplink and downlink programmable gain amplifiers, and the DTMF tone generator. The external vibrator control is provided in this module and can be controlled through an audio signal or direct I2C writes to the registers.

The figure shows simplified block diagram connected to PC here the driver makes RS232 device appear as a serial port in the PC OS, that the user can write arbitrary data to. The driver will format this data as necessary for USB transport, and send it on wire; the RS232 will then accept this data and convert it to TTL-level (0-5V) RS-232 signal (and the same happens for the reverse direction, when reading). Given that RS-232 is conceptually much easier to understand.

In order to specify what sampling rates, in terms of digital audio, would this hardware support - the most important factor to consider is the data transfer rate, that can be achieved between the OMAP™ 3 and the RS232 over the serial link.

![Figure 1: Simplified Block Diagram connected to PC](image-url)

In order to specify what sampling rates, in terms of digital audio, would this
hardware support the most important factor to consider is the data transfer rate, that can be achieved between the OMAP™ 3 and the RS232 over the serial link. As far as this serial link goes, the OMAP™ 3 states maximum rate of 2.5 Mbps while the RS232 states up to 3 MBaud. As the tw14000 driver supports 2 Mbps by default. A speed of 2Mbaud translates to 200000 Bps, which would be enough to carry 200000/44100=4.5 mono/8-bit/44.1kHz channels or two mono/16-bit/44.1 kHz channels or one CD quality stereo/16-bit/44.1 kHz channel. However, one still needs to determine what actual data transfer rates can be achieved, and under which conditions.

4.1. Building and Running:
Both the source code, and instructions for building and running, can be found. The source code consists of a modified version of .c; the ALSA-specific part in .h; associated headers With this in place, high-level audio software will be able to address the playback and capture audio data through it. The input can have their signal captured at 44.1 kHz in audio software. The McBSP on the OMAP3530 modules provides full duplex, direct serial interface between codes inside the TPS65950. It support I2S format to the TPS65950. At this soundcard output; on which, when audio software plays back audio data, (analog) PWM output is generated (audible). In addition to processing I/O requests, drivers also handle configuration requests. Applications may configure the device, for example, by setting the bandwidth of a network card or the volume for a sound card. Configuration requests may change both driver and device behavior for future I/O requests. Some interfaces are used to configure settings for the audio interface and to extract information.

` snd_soc_devic *socdev – Gives information on the codec device. ` 
` snd_soc_new_pcms - Interface for the platform pcm device ` 

5. Driver Architecture:
The audio codec driver additionally exposes CD quality, stereo/16-bit/44.1kHz capability to allow for direct playback interface with Audacity (and most media player software). Audacity is a free digital audio editor and recording application, available for Windows, Mac OS X, Linux and other operating systems; An Audacity is a software
program for editing, mixing, and applying effects to digital audio recordings according to various sampling rates. By declaring the driver capable of 16-bit stereo, we have not changed the number of sub streams. The driver will automatically converts the 16 bit stereo stream from audacity to an 8 bit by preserving the 44100 Bps rate before it sends it to USB and thus an audio digital loopback can be demonstrated on this driver directly from audacity. while the transfer of incoming USB data to PC memory as well as the transfer of data from dma_area to user memory of high-level audio software as part of the ALSA middle layer likely involves DMA the transfer of memory that is performed as part of audio codec timer_function.

5.1. Flowchart:

6. Result:
The audio test is divided into two tests, one for audio in and one for audio out. Audio is recorded into the audio in port and then played out the audio out port.
Audio In:-Make Sure your player is running and Audio Line in is connected to board.
Audio Out:-The recorded audio should be heard on the Speakers.
The recorded wave files at various sampling rates from 8 kHz to 44.1 kHz
1) Recorded wave File At 8000 HZ Sample Rate
2) Recorded wave File At 11025 HZ Sample Rate

3) Recorded wave File At 16000 HZ Sample Rate

4) Recorded wave File At 22050 HZ Sample Rate
5) Recorded wave File At 32000 HZ Sample Rate

6) Recorded wave File At 41000 HZ Sample Rate
References:

1. “Sound Systems on Linux: From the Past to the Future” Takashi Iwai <tiwai@suse.de> SuSE Linux AG, Nuremberg, Germany Linux 2003 Conference, Edinburgh, Scotland


5. “Recovering Device Drivers” Michael M. Swift, Muthukaruppan Annamalai, Brian N. Bershad, and Henry M. Levy Department of Computer Science and Engineering University of Washington Seattle, WA 98195 USA