Linux Sound Architecture

Caius Cioran
University of Paderborn
caius@mail.uni-paderborn.de

December, 23 2013

Abstract

Sound output is one of the major tasks of the Soundgates application. But the sound architecture in Linux can be built upon different systems in different layers, dependent on the system's requirements. This is why this paper gives an overview about drivers, different audio technologies and provides a suitable architecture for the Soundgates system.

1 Introduction

The Zedboard offers an ADAU1761 Audio Codec for digitial audio processing. But instead of wiring our Hardware Sound Components directly via the i2s interface to the chip, we want to benefit from a software abstraction through an existing and already evaluated Linux driver. However, the Linux sound architecture is flexible and simultaneously complex, since there is always more than one application for the same purpose. Additionally, the raw sound is passed from the application through different layers until it reaches the audio device. The goal is to find a suitable sound architecture for the soundgates program. Therefore, the second Section deals with an introduction to the Linux Operating System, its system concept and hardware drivers. Section 3 gives an overview about the available sound layers together with some of their most famous implementations, before Section 4 explains a suitable example architecture for the Soundgates system. The last Section concludes the content and gives an overview about possible extensions.

2 Linux

Linux is an open source Operating System which was initially released in 1991 by Linus Torvalds. Due to its open minded license and availability for a wide range of architectures it became very popular in many environments like embedded systems, notebooks, servers or even High End Power Computers [8]. Unlike in proprietary Operating Systems it is

possible to modify each single aspect to satisfy any given requirement. Even hardware devices without official Linux support by the vendors usually get an open source driver by the large community. However, at least the top manufacturers understand the significance of Linux and therefore support their devices with first party software [5]. The Linux kernel was declared as stable with version 1.0 in 1994. But development of further improvements and features is an ongoing process [6]. The kernel itself is used by different groups or companies along with other free software to create Linux distributions. Thus, it is easy to use Linux as an Operating System without any specific knowledge. Distributions usually differ in the pre-installed applications along with some other modifications i.e. at the kernel. Additionally, a packet management tool is usually included to reduce the complexity of installing other programs. Debian, Ubuntu and Fedora are some of the most popular distributions today [7].

2.1 Driver

In order to make communication between Hardware devices and applications possible it is mandatory to install the corresponding driver. They provide a specific interface to programmers to abstract underlying complexity. Therefore different functionalities can be accessed by simple method calls. Linux partitions the system's virtual memory into kernel and user space. An overview about this is given in Figure 1. On the one hand, this procedure improves the security to prevent malicious behavior from one application towards another by separating their memory address range accesses. On the other hand, it grants access permissions and synchronization between multiple programs [8]. In order to switch into kernel space an application has to make a system call. They are executed by writing the system call's id and the necessary parameters into the target architecture's specific registers. To abstract this architecture dependent code and to achieve portability between multiple systems Linux provides an Application Programming Interface (API). It provides multiple interfaces to programs and uses the implementation of the target architecture. An access to the printf() method from inside of an application redirects the call to the stdio library, which executes the specific system-calls (Figure 2). Again, the target driver is abstracted by a device node which is basically a file inside of the file system. Values can be sent to the physical device by writing and received by reading from a target device. The correct driver is bound through an device id number to the corresponding device node [4].

The Linux Kernel provides two different inclusion mechanisms for drivers. On the one hand they can be statically compiled into the Kernel to make them available during booting phase. This is necessary for the network driver if for example the Kernel has to load its file system from a remote server. On the other hand they can be compiled as modules which can be loaded and unloaded at runtime [5]. Thus, it is not necessary to reboot the system after installing a new driver, which comes in handy for developers and users.

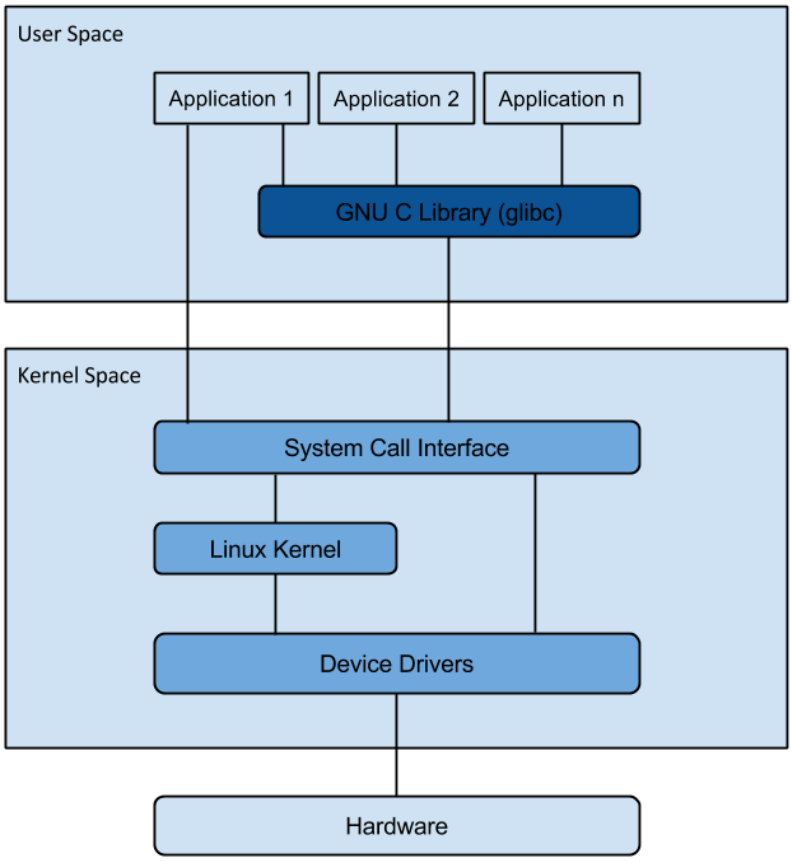

Figure 1: The Linux system architecture is divided into two layers. Every application is started inside of the user space and in order to get access to system components it has to make a system call to switch into kernel space. The figure is based on [8].

3 Sound Architectures

A computer stores audio files in form of bits, just like any other data. This digital data has to be transformed into analog signals to produce sound. Correspondingly, to record any noise from the outside world, it also has to be transformed into digital data. That process is realized through an audio-interface and controlled by a device driver [2]. However, Linux supports audio output via different APIs and drivers, with different capabilities. Since there is no definite standard it is up to the user whether to use the default architecture of the installed distribution or to configure it to the own needs. Some possible configurations are depicted in Figure 3. The following chapter introduces some popular Linux Sound APIs.

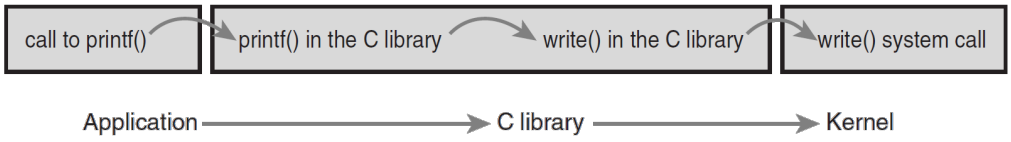

Figure 2: The glibc library offers different interfaces to abstract system calls. This makes an application portable to other architectures and simplifies programming. Taken from [8].

3.1 Open Sound System (OSS)

Originally, the Linux kernel included the OSS, which supported on its release one single sound card. But soon the driver evolved and began supporting additional audio-interfaces. The basic functionality is very similar to the device node procedure mentioned in chapter 2.1. Basically, OSS creates the '/dev/dsp' device, which can be read from to capture sound or written to, for audio output (depicted in Figure 3 - A). This makes the usage simple but reduces the functionality, since Full duplex, the procedure of recording and playing sound at the same time, is not possible [3]. Additionally, it does not support software mixing. Thus, just one single application at a time is able to communicate with the device if the soundcard does not support hardware mixing. Therefore, it is neither possible to hear the sound of a game nor recording audio while listening to music [4]. Another disadvantage is the lack of a user space library, which could provide functions like resampling of data. Consequently, the developer has to take care of using the supported sound format of the used audio device [9]. When it became public that OSS goes proprietary and therefore closed source, it has been decided to remove it from Linux and replace it by another Open Source software with a focus on more functions and flexibility like the support for surround sound and software mixing [9]. In 2007 OSS went as 'OSS v4' Open Source again, but it did not regain the popularity of its old days [3].

3.2 Advanced Linux Sound Architecture (ALSA)

Since OSS did not include software mixing and went proprietary, the Linux community developed the ALSA system, which became the standard audio system since Linux 2.6 [3]. The ALSA software includes the ALSA Kernel driver and the user-space library alsa-lib. Again, the kernel driver makes the communication between the system and audio-interface possible by creating multiple device nodes. Nevertheless, the ALSA kernel supports software mixing of multiple sound channels in kernel space and allows thereby full-duplex operations even without the optional library (Figure 3 - A). But the hardware layer can be abstracted with the alsa-lib API to make the development process for music applications more comfortable. So the process of reading and writing on devices is replaced by executing well defined interfaces (Figure 3 - B). In order to produce sound, the audio-interface has to be configured like for example setting the desired sample rate. Afterwards the program has to write data into a sound buffer, which will be trans-

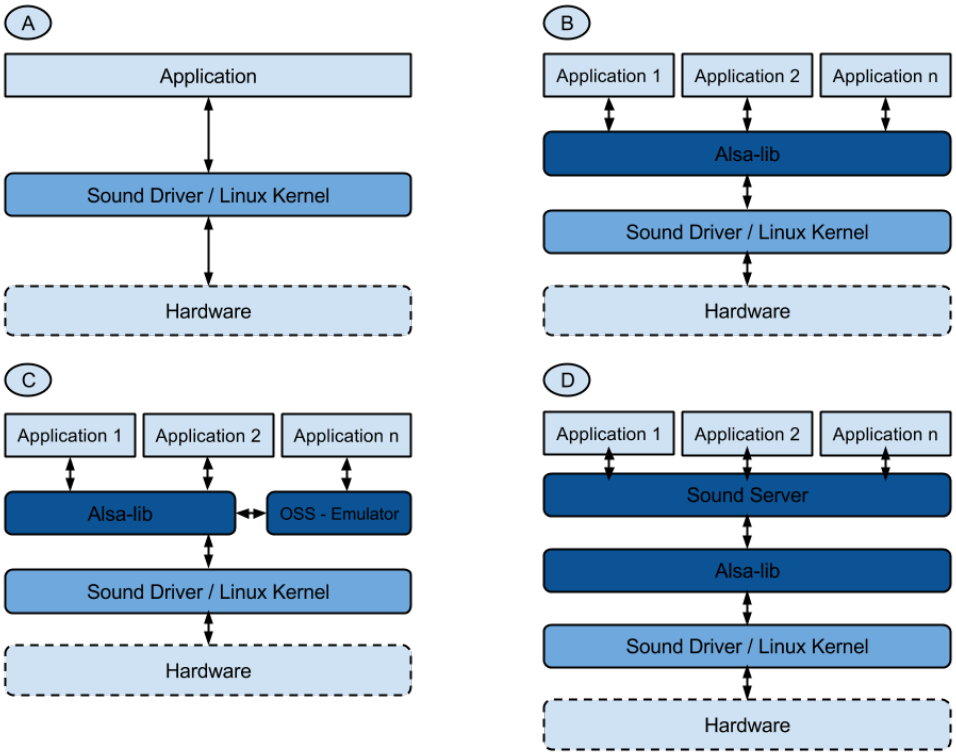

Figure 3: Linux sound architecture consists of different layers. The sound driver is the only indispensable part, since it makes communication between software and hardware possible. An additional library is suitable to abstract the communication with the driver (B) and a sound server makes software mixing of application output possible (D) [3].

ferred to the audio-interface as soon as it has collected the right amount of data. The same procedure holds vice versa for capturing sound. The audio device captures analog signals and transforms them into digital data. As soon as the capture buffer is filled it generates an interrupt to notify the processor. Additionally, ALSA includes a compatibility mode to old programs by emulating the OSS interface via emulated devices (depicted in Figure 3 - C).

3.3 PulseAudio

Instead of providing direct access to the connected audio-interface, PulseAudio works as a layer between applications and the driver (depicted in Figure 3 - D). It is in charge of handling sound flow between running applications, thus one application can directly use the sound output of another program as its input, even over a network interface. Additionally, it mixes the sound output of different applications into one signal to give parallel access to the soundcard. Additional to Linux PulseAudio supports Win32 systems [3]. However, due to some timing constraints OSS can not be used with any audio server [9].

3.4 Jack

JACK Audio Connection Kit (JACK) is another audio server with a focus on interconnecting professional audio applications and hardware output. Through its low latency it is possible to capture sound, add a filter and finally play it through the speakers with a minimal delay [10].

4 Soundgates

The Soundgates synthesizer plays a patch of sound producing components, which can be calculated in software and in hardware. A connection between two nodes represents that the second node uses the output of the first as its own input. A patch is created inside of the Soundgates editor and different values can be modified at runtime by external components like mobile phones. The application can be compiled as a linux executable. Therefore, it is necessary to determine the best suitable sound architecture, which depends on the trade-off between latency and functions. Of course, latency is directly correlated to the used layers, but every single layer adds more functions and flexibility. To get an overview about possible constellations it is necessary to specify the target hardware.

4.1 Hardware environment

In order to combine flexible hardware components to produce sound, along with the power of an all purpose Central Processing Unit (CPU) to execute Linux, we decided to use the Zedboard for the Soundgates system. It offers a dual Corex-A9 processor together with Programmable Logic cells and an ADAU1761 audio-interface [1]. Additionally, the ZedBoard is supported by ReconOS[1], which provides posix like hardware threads and therefore makes communication to the Soundgates system possible.

4.2 Sound Architecture Analysis

The first step to make sound output possible is to select a suitable driver. Despite the fact that OSS is practically deprecated, it additionally does not contain any features beyond playing and capturing sound. The only noticeable advantage is fast communication with the hardware. ALSA on the other side offers a huge range of device drivers including the ADAU1761[2] and a user space library to control the chip. Moreover, the future could bring new use-cases to the soundgates system like using a microphone as an input for a sound component. This requires full duplex, which is not supported by OSS (see Section 3). This makes ALSA the primary choice for the bottom layer. The next layer is alsa-lib. It abstracts the sound driver, supports portability of code and makes programming easier. Furthermore, the usage of this library should not result in noticeable latency penalties,

[1] https://github.com/EPiCS/reconos

[2] http://wiki.analog.com/resources/tools-software/linux-drivers/sound/adau1361

since it does not include any heavy operations on data. This is why it should be included into the sound architecture. Nevertheless, a sound server is not needed since we do not need to mix sounds of different applications, because Soundgates Synthesizer will be the only running process with contact to the audio-interface. But if this behavior will be needed in a future version it is pretty easy to include a sound server afterwards. The target system is depicted in Figure 4.

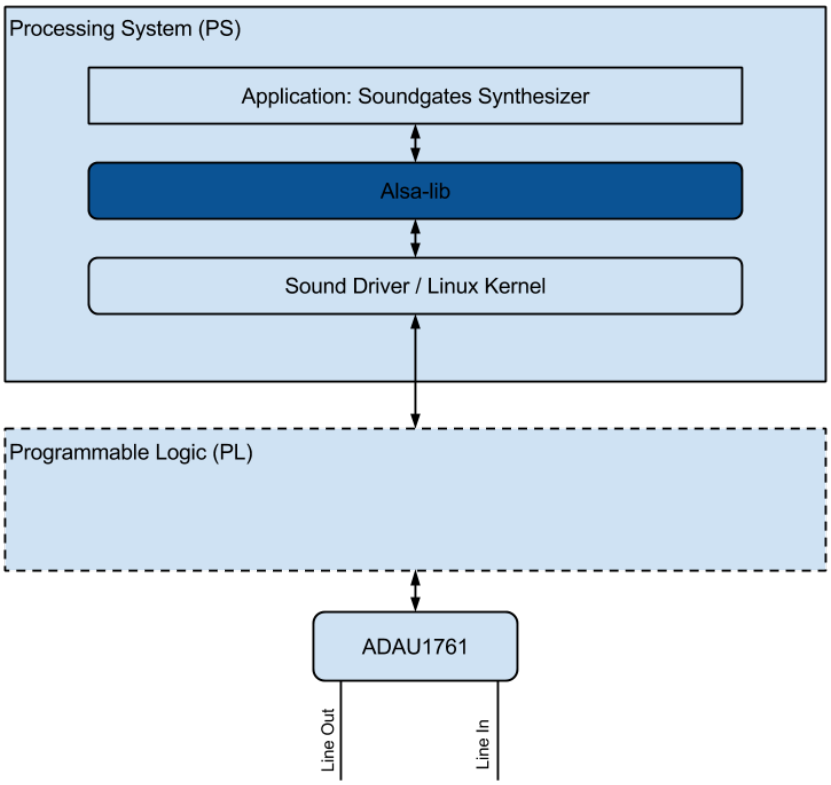

Figure 4: A suitable sound architecture for the Soundgates system. The application uses the alsa-lib to communicate with the sound driver. However, the driver consists of an implementation of i2s and i2c, which is necessary to control the audio-interface, since it is not directly connected to ZedBoard [1]. A reference design is given by the vendor[4].

5 Conclusion

The audio support for the Open Source Operating System Linux evolved from one single audio-interface to a high end sound producing system. It started with OSS, which neither supports software mixing nor full duplex audio. So it was replaced by ALSA to satisfy the growing needs of musicians and developers. Additionally, the new standard supports

[2]http://wiki.analog.com/resources/fpga/xilinx/kc705/adv7511

a huge amount of soundcards out of the box. However, the Linux sound architecture has some additional layers. A sound server like PulseAudio can be used as a software mixer or even to stream audio between applications. Professional musicians are free to interchange PulseAudio with Jack to satisfy their requirements. The goal was to construct a valuable sound architecture for the Soundgates system. It contains the ALSA kernel driver, along with the alsa-lib library to abstract its usage. A sound server is not necessary because the Soundgates synthesizer is the only application with contact to the audio device. However, this system is extendable if any requirements would change during the development process. If we would like to implement sound components on mobile devices we could add a sound server layer which is capable of routing the data.

6 Acronyms

API Application Programming Interface
ALSA Advanced Linux Sound Architecture
CPU Central Processing Unit
JACK JACK Audio Connection Kit
OSS Open Sound System

References

[1] Avnet. Zedboard - hardware user's guide. `http://www.zedboard.org/sites/default/files/documentations/ZedBoard_HW_UG_v2_0.pdf`, October 2013.

[2] Paul Davis. A tutorial on using the alsa audio api. `http://equalarea.com/paul/alsa-audio.html`, 2002.

[3] Free Electrons. Audio in embedded linux systems. `http://free-electrons.com/doc/embedded_linux_audio.pdf`, September 2009.

[4] Martin Gräfe. C und linux. Carl Hanser Verlag, April 2010.

[5] Christopher Hallinan. Embedded linux primer: A practical, real-world approach. Prentice Hall, September 200.

[6] Tim Jones. Anatomy of the linux kernel: History and architectural decomposition. `http://www.ibm.com/developerworks/linux/library/l-linux-kernel/`, June 2007.

[7] Unsigned Integer Limited. Distrowatch. `http://distrowatch.com/`, December 2013.

[8] Robert Love. Linux kernel development. Developer's Library, June 2010.

[9] Lennart Poettering. A guide through the linux sound api jungle. `http://0pointer.de/blog/projects/guide-to-sound-apis`, September 2008.

[10] TuxRadar. How it works: Linux audio explained. `http://tuxradar.com/content/how-it-works-linux-audio-explained`, April 2010.