The JamBerry - A Stand-Alone Device for Networked Music Performance Based on the Raspberry Pi

Florian Meier
Hamburg University of Technology
Schwarzenbergstraße 95
21073 Hamburg, Germany,
florian.meier@tuhh.de

Marco Fink and **Udo Zölzer**
Dept. of Signal Processing
and Communications
Helmut-Schmidt-University
Holstenhofweg 85,
22043 Hamburg, Germany,
marco.fink@hsu-hh.de

Abstract

Today's public Internet availability and capabilities allow manifold applications in the field of multimedia that were not possible a few years ago. One emerging application is the so-called Networked Music Performance, standing for the online, low-latency interaction of musicians. This work proposes a stand-alone device for that specific purpose and is based on a Raspberry Pi running a Linux-based operating system.

Keywords

Networked Music Performance, Audio over IP, ALSA, Raspberry Pi

1 Introduction

The ways of today's online communication are versatile and rapidly evolving. The trend went from text-based communication, over audio-based communication, and finally constituted in multimedia-based communication. One arising branch of online communication is the so-called Networked Music Performance (NMP), a special application of Audio over IP (AoIP). It allows musicians to interact with each other in a virtual acoustic space by connecting their instruments to their computers and a software-based link-up. This procedure allows artistic collaborations over long distances without the need of traveling and hence, can enrich the life of artists. Instead of increasing the content dimensionality and therefore the data rate, the challenge in AoIP is to fulfill a certain delay threshold that still allows musical interaction.

For the purpose of providing an easy-to-use system realization, an all-in-one device, entitled the *JamBerry*, is presented in this work. The proposed system, as shown in Fig. 1, consists of the well-known Raspberry Pi [1] and several custom hardware extensions. These are necessary since the Raspberry Pi does not provide high-quality audio output and no audio input at all. Furthermore, the proposed device includes several hardware components allowing a quick and simple connection of typical audio hardware and instruments. The Raspberry Pi itself can be described as chip-card-sized single-board computer. It was initiated for educational purposes and is now widely-used, especially in the hardware hobbyist community since it provides various interfaces for all sorts of extensions.

Figure 1: The JamBerry Device

The paper is structured as following. An introduction into the topic of Audio over IP is given in Section 2, including the requirements and major challenges when building such a system. Section 3 gives a detailed view on the actual AoIP software running on the JamBerry. The necessary extensions of the Linux audio drivers and the integration in the ALSA framework is depicted in Section 4. The custom hardware extensions to the Raspberry Pi are explained in Section 5. Section 6 highlights the capabilities of the JamBerry in the contexts of audio and network parameters, whereas concluding thoughts can be found in Section 7.

2 Audio over IP

Transmission of Audio over IP-based networks is nowadays a wide-spread technology with two main applications: Broadcasting audio streams and telephony applications. While the first one provides no return channel, the second one allows for direct interaction over large distances. Although, the requirements in terms of audio

quality and latency for playing live music together are not fulfilled by current telephony systems.

The massive spreading of broad-band Internet connections and increase in network reliability allows the realization of AoIP systems now. Therefore, this topic gained much research attention in the last years. A good introduction into the topic of Networked Music Performances and the associated problems can be found in [2], while [3] gives an extensive overview of existing systems.

An early approach was SoundWIRE [4] by the Center for Computer Research in Music and Acoustics (CCRMA), where later JackTrip [5] was developed. JackTrip includes several methods for counteracting packet loss such as overlapping of packets and looping of data in case of a lost packet. It is based on the JACK sound system, just like NetJack [6] that is now part of JACK itself. To avoid the restriction to JACK, Soundjack [7] is based on a generic audio core and hence, allows cross-platform musical online interaction.

The Distributed Musical Rehersal Environment [8] focuses on preparing groups of musicians for a final performance without the need to be at the same place. Remote rehersal is also one of the applications of the DIAMOUSES framework [9] that has a very versatile platform including a portal for arranging jam sessions, MIDI support and DVB support for audience involvement.

2.1 Requirements

The goal of this project was to build a complete distributed music performance system to show the current state of research and establish a platform for further research. The system is supposed to be usable in realistic environments such as rehearsal rooms. Therefore, it should be a compact system that integrates all important features for easy to setup jamming sessions. This includes two input channels with various input capabilities to support high-amplitude sound sources such as keyboards or preamplified instruments, as well as low-amplitude sound sources like microphones and passive guitar pickups. Furthermore, it should drive headphones and provide line-level output signals.

The system should support sampling rates of 48 kHz with a bit depth of 16 bit. Higher values do not provide much benefit in quality. Furthermore, no further signal processing steps, depending on highly-detailed signaled representations, are involved. To allow the interaction with several musicians but still stick to the computational constraints of the Raspberry Pi, the system shall support up to four interconnected JamBerries.

2.2 Challenges

Transmission of audio via the Internet is considerably different from point-to-point digital audio transmission techniques such as AES/EBU [10] and even Audio over Ethernet (AoE) techniques like Dante or EtherSound [11]. The transmission of data packets via the Internet is neither reliable nor properly predictable. This leads to audio data being considerably delayed or even vanished in the network.

This is commonly counteracted by using large data buffers where the packets arriving in irregular time intervals are additionally delayed so that late packets can catch up. Unfortunately, large buffers are contradictory to the requirements of distributed music performances since a minimum latency is essential. Interaction of several musicians is solely achievable when the round trip delay does not exceed a certain threshold [12; 13]. Secondly, even large buffers do not prevent dropouts resulting from lost packets. Therefore, this project takes two completive approaches:

- Audio data packets that do not arrive in time are substituted by a technique called error concealment. Instead of playing back silence, audio is calculated from preceding data.
- The data buffer length is dynamically adjusted to the network conditions. This enables minimum latency while still providing good audio quality.

3 Software System

The AoIP software itself is a multi-threaded C++11 application running in user space. It accesses the audio hardware via the well-known ALSA [14] library. The user interaction takes place via a WebSocket interface that enables the use of a JavaScript/HTML GUI that can be accessed via the integrated touchscreen as well as from a remote PC or tablet. The WebSocket interface is provided by a library [15] written during this project running the WAMP [16] protocol.

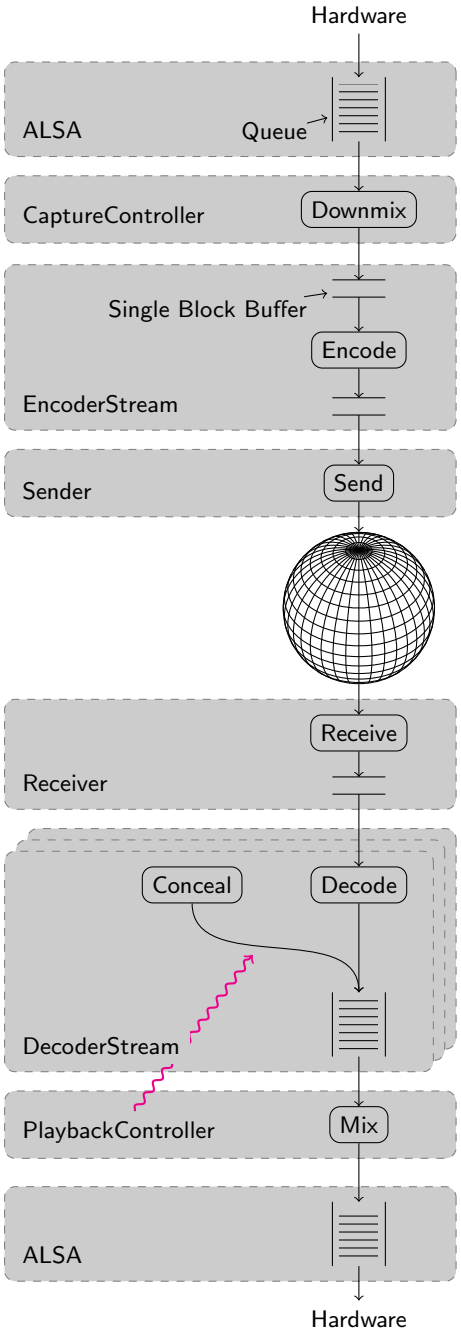

Figure 2: Data flow of the JamBerry software

The data flow of the audio through the software is depicted in Fig. 2. Audio is captured from the hardware via the ALSA library. As soon as a block (120 or 240 samples) of new data is available, it is taken by the *CaptureController* that mixes the signal down to a single channel. Transmitting multiple streams is possible, too, but provides a negligible benefit in this scenario. The data can be transmitted as raw data. Alternatively, the required data rate can be reduced by utilization of the Opus [17; 18] low-latency coding procedure. The encoding is done by the *EncoderStream* that passes the data to the sender for sending it to all connected peers via unicast UDP. Currently, there is no discovery protocol implemented, so the peers have to be entered manually. As soon as the data is received at the receiver, it is decoded and pushed into the receiver buffer queue. The *PlaybackController* mixes the data from various sources and enables ALSA to access the result. Thus, a continuous reading of data is realized. In the case of missing data an error concealment procedure is triggered to replace the missing data and avoid gaps in the playback. The current implementation utilizes the concealment procedure from the Opus codec, since its complexity is low in contrast to other known concealment strategies [19; 20; 21]. Alternatively, the last block can be repeated until newer data arrives (so-called "wavetable mode" as in [5]). The queuing process at the receiver is explained in more detail in the following.

3.1 Adaptive Queuing

In order to achieve minimum latency while maintaining good audio quality, the length of the audio buffer is adjusted to the network conditions within the playback thread. The corresponding control routine is depicted in Fig. 3.

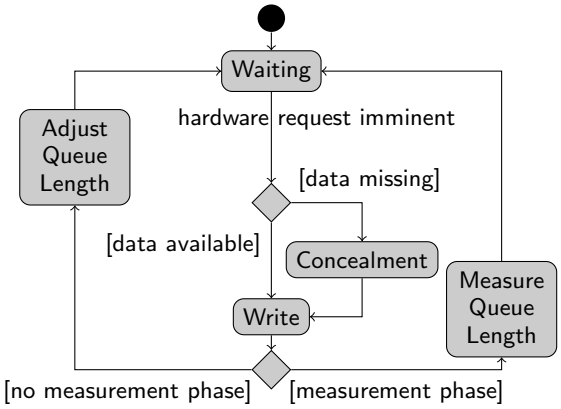

Figure 3: Process of Playback Thread

The ALSA data queue is kept very short to avoid unnecessary delays that would increase the overall latency. The *PlaybackController* monitors the state of ALSA and just before the hardware will request new data, it is written to the ALSA buffer. Whenever current audio data exists in the moment of the hardware request, this data is utilized. In the case of missing data, the error concealment routine is triggered to produce the corresponding data. The computa-

tion of concealment data takes some time. This period of time is taken into account to provide the data just at the right point in time.

In order to maintain a reasonable buffer size, a simple open-loop control was implemented. A buffer size that is unreasonably large would result in useless delay. When the buffer is too small, a major part of the audio packets arrives too late. Although a certain amount of packets can be concealed, the audio quality decreases with a rising amount of lost packets.

Right after a new connection was established, the buffer size is set to a very high value. In the following few seconds, the length of the queue in samples Q is measured and the standard deviation σ_Q is calculated. After the measurement phase is over, the optimal queue length is calculated as

$$Q_{opt} = \beta \cdot \sigma_Q, \tag{1}$$

where the constant $\beta \geq 1$ accounts for packets outside the range of the standard deviation. When the current queue length is outside the interval

$$[Q_{opt} - Q_{tol}, Q_{opt} + Q_{tol}], \tag{2}$$

the corresponding number of samples is dropped or generated. Once the queue is adjusted to the current network characteristic, this occurs very infrequently so the audible effect is insignificant. The parameters β and Q_{tol} are used to trade-off the amount of lost packets, and therefore the audio quality, against the latency.

4 Linux Kernel Driver

The Raspberry Pi has neither audio input nor proper audio output. The existing audio output

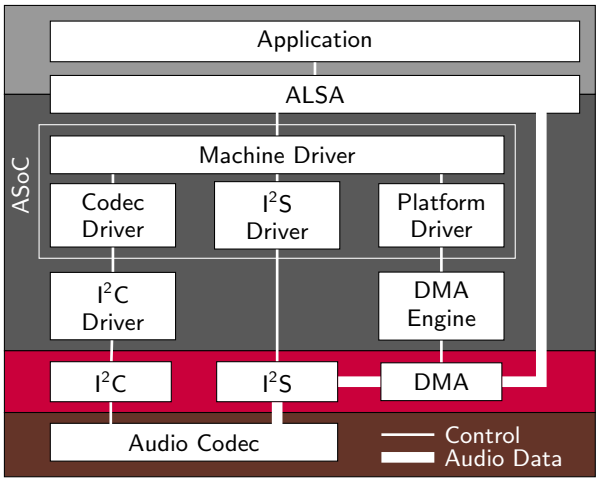

Figure 4: Structure of ASoC and the embedment into the Linux audio framework

is driven by a pulse-width modulation (PWM) interface providing medium quality audio. Fortunately, there is another possibility for audio transmission: The Broadcom SoC on the Raspberry Pi provides a digital I²S interface [22] that can be accessed by pin headers. Together with an external audio codec as explained in the next section, this enables high quality audio input and output. However, the Linux kernel lacked support for the I²S peripheral of the Raspberry Pi. An integral part of this project was therefore to write an appropriate kernel driver.

Since this driver should be as generic as possible, it is implemented as a part of the ALSA System on Chip (ASoC) framework. It is a subsystem of ALSA tailored to the needs of embedded systems that provides some helpful abstractions that makes it easy to adapt the driver for use with other external hardware. Actually, today there is quite a large number of both open and commercial hardware that uses the driver developed during this project.

Fig. 4 depicts the general structure of ASoC as used for this project. When an application starts the playback of audio, it calls the corresponding function of the ALSA library. This again calls the appropriate initializers for the involved peripheral drivers that are listed in the machine driver. In particular this is the codec driver that is responsible for control commands via I²C, the I²S driver for controlling the digital audio interface, and the platform driver for commanding the DMA engine driver. DMA (Direct Memory Access) is responsible for transmitting audio data from the main memory to the I²S peripheral and back. The I²S peripheral forwards this data via the I²S interface to the audio codec. For starting the playback of the codec, the codec driver will send an appropriate command by using the I²C subsystem. The codec driver is used for transmitting other codec settings such as volume, too.

These encapsulations and generic interfaces are the reason for the software structure's flexibility and reusability. For using a new audio codec with the Raspberry Pi, only the codec driver and the slim machine driver have to be replaced. In many cases only the wiring by the machine driver has to be adapted since there are already many codec drivers available. The spreading of these drivers is based on the frequent usage of ASoC on different platforms.

5 Hardware

Since the Raspberry Pi does not provide proper analog audio interfaces, major effort was spent designing audio hardware, matching the NMP requirements. Furthermore, a touchscreen for user-friendly interaction was connected that requires interfacing hardware. Due to these extensions, the JamBerry can be used as a standalone device without the need of external peripherals such as a monitor.

An overview of the external hardware is depicted in Fig. 5. The extension's functionality is distributed module-wise over three printed circuit boards: A codec board that contains the audio codec for conversion between analog and digital domain. It is stack mounted on the Raspberry Pi. This board is connected to the amplifier board that contains several amplifiers and connectors. The third board controls the touchscreen and is connected to the Raspberry Pi via HDMI. In the following, the individual boards are explained in more detail.

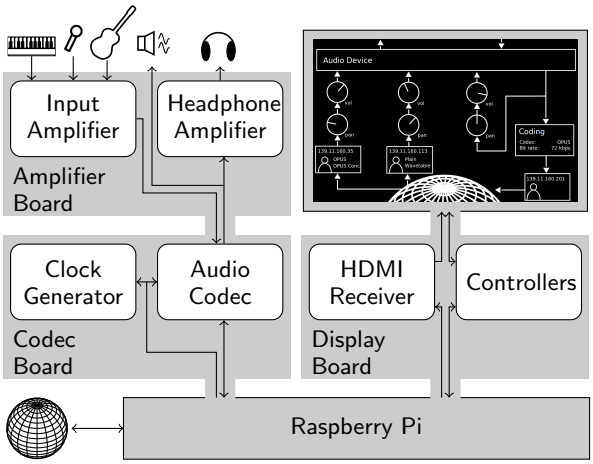

Figure 5: Hardware Overview

5.1 Codec Board

The main component on the digital audio board is a CS4270 audio codec by Cirrus Logic. It has integrated AD and DA converters that provide sample rates of up to 192 kHz and a maximum of 24 bits per sample. It is connected to the I^2S interface of the Raspberry Pi for transmission of digital audio and to the I^2C interface for control. A linear voltage regulator provides power for the analog part of the audio codec, while the digital part is directly driven by the voltage line of the Raspberry Pi. The audio codec is controlled by an external master clock generator. This enables fine-grained synchronization of the sampling frequency on different devices and prevents clock drifts as shown in [23]. The MAX9485 clock generator provides this possibility by a voltage controlled oscillator that can be tuned by an external DAC.

5.2 Amplifier Board

The analog audio board is designed to provide the most established connection possibilities. On the input side two combined XLR/TRS connectors allow the connection of various sources such as microphones, guitars or keyboards. Since these sources provide different output levels that have to be amplified to line-level for feeding it into the audio codec, a two-stage noninverting operational amplifier circuit is placed channel-wise in front of the AD conversion unit. It is based on OPA2134 amplifiers by Texas Instruments that have proven their usability in previous guitar amplifier projects. The circuit allows an amplification of up to 68 dB.

On the output side a direct line-level output is provided as well as a MAX13331 headphone amplifier. It can deliver up to 135 mW into 32 Ω headphones. Furthermore, the analog audio board contains the main power supply for the JamBerry.

5.3 Touchscreen Driving Board

In order to provide enough display space for a pleasant usage experience, but still maintain a compact system size, a 7" screen size is used. A frequently used, thus reliable, and affordable resistive touchscreen of that size is the AT070TN92. For using it together with the Raspberry Pi, a video signal converter is needed to translate from HDMI to the 24 bit parallel interface of the TFT screen. This is provided by a TFP401A by Texas Instruments. The touch position on the screen can be determined by measuring the resistance over the touch panel. This measurement is subject to noise that induces jittering and results in imprecise mouse pointers. The AD7879-1W touch controller is used to conduct this measurement since it provides integrated mean and median filters that reduce the jitter and is controlled via I^2C. The same interface is provided by a DAC for controlling the backlight of the TFT. An additional cable connection for the I^2C connection was avoided by reusing the DDC interface inside the HDMI cable as carrier for the touch and brightness information.

6 Evaluation

The system was evaluated in terms of overall latency introduced by the network as well as audio quality.

6.1 Network

In order to evaluate the behavior of the system under various and reproducible network conditions, a network simulator was implemented. Fig. 6 shows the use of a single JamBerry device connected to the simulator that bounces the received data back to the sender.

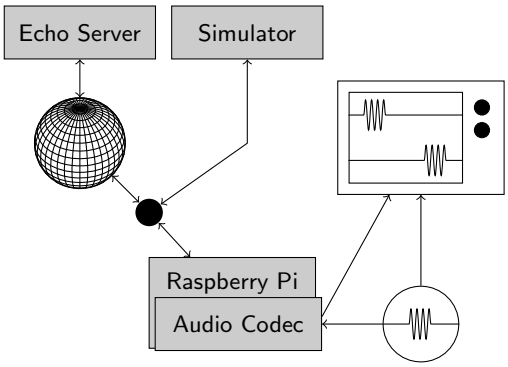

Figure 6: Software Evaluation System

For calibrating the simulator to real conditions a network connection of 13 hops to a server, located in a distance of 450 km, is used. Fig. 7 shows the distribution of the packet delay. The average delay is about 18 ms with a standard deviation of 4 ms.

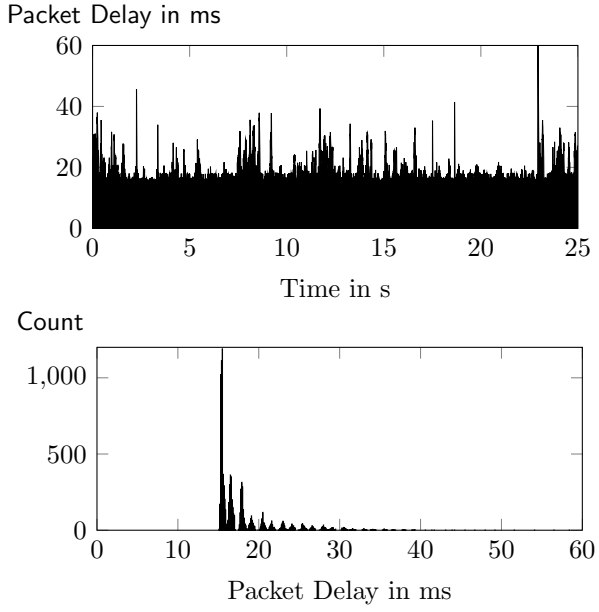

Figure 7: Time series and histogram of the packet delay over the test route

The overall latency is measured by generating short sine bursts and feeding them into the JamBerry. This signal is compared to the resulting output signal by means of an oscilloscope. In addition, GPIO pins of the Raspberry Pi are toggled when the sine burst is processed in different software modules as presented in Sect. 3.

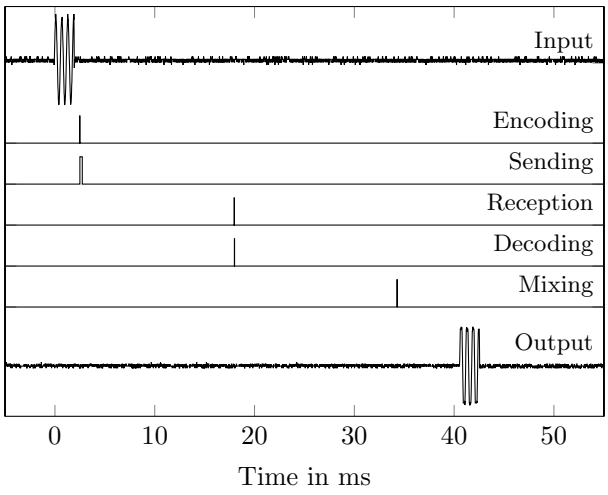

Figure 8: Journey of a Sine Burst

A resulting oscillogram can be seen in Fig. 8. The overall latency is about 40 ms. The time between sending and reception is 15 ms. This matches the time for the actual transmission. Between decoding and mixing, the packet is delayed in the buffer queue for about 16 ms. This buffering is needed to compensate the high jitter of the connection.

For the following evaluation, the overall latency is measured by using the above method while recording the amount of lost packets. Fig. 9 demonstrates the influence of factor β in Eq. (1) while having a constant jitter variance of 9.5 ms^2. With low β, the optimal queue length is short, so the overall latency is short, too. Although, since there is less time for late packets to catch up, the amount of packet loss is very high. With increasing β, the amount of lost packets decreases, but the latency increases. Since sophisticated error concealment algorithms can compensate up to 2% packet loss [19], a constant $\beta = 3$ is chosen for the next evaluation, which is illustrated in Fig. 10. It demonstrates how the control algorithm handles various network conditions. With increasing network jitter variance, the system is able to adapt itself by using a longer queue length. This increases the overall latency, but not the packet loss so the audio quality stays constant.

6.2 Audio

The evaluation of the JamBerry's audio quality was performed module-wise. Therefore, the audio output, audio input, the headphone amplifier and pre-amplifiers were independently inspected. First of all, the superiority of the proposed audio output in contrast to the original PWM output shall be demonstrated.

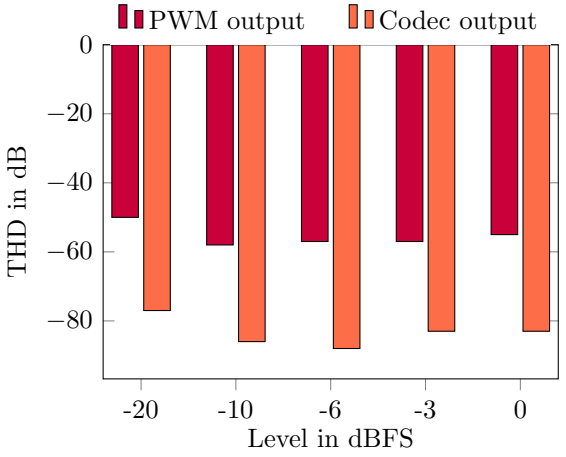

Figure 11: THD of the Raspberry Pi PWM and the codec board output for a 1 kHz sine using different signal levels

If a 1 kHz sine tone is replayed using both outputs accordingly and inspect the corresponding output spectra, as done in Fig. 12, it becomes apparent that the quality is increased significantly using the new codec board. The PWM output introduces more distortion, visible in Fig. 12 in form of larger harmonics at multiples of the fundamental frequency. For example, the amplitude of the first harmonic differs in about 40 dB. Also the noise floor at higher

	Level in dBFS	THD in dB	SNR in dB
Outputs			
PWM output	0	-57	55
Codec output	0	-81	80
Input			
Codec input	0	-91	71

Table 1: Digital audio hardware characteristics

	Gain in dB	THD in dB	SNR in dB
Amplifiers			
Headphone	16	-85	79
Input	17	-81	66
Input	34	-74	48

Table 2: Analog audio hardware characteristics

frequencies is significantly lower. A difference of up to 10 dB can be recognized in Fig. 12. At 50 Hz ripple voltage from the power supply can be seen. Using a power supply of higher quality can reduce this disturbance.

The distortion and noise, audio hardware introduces to audio signals signal is typically expressed in total harmonic distortion (THD) and Signal-Noise-Ratio (SNR), respectively. THD describes, in most conventions, the ratio of the energy of harmonics, produced by distortion, and the energy of the actual signal. In contrast, SNR represents the ratio between the original signal energy and the added noise.

The THD's of the two outputs are illustrated for several signal levels in Fig. 11. Apparently,

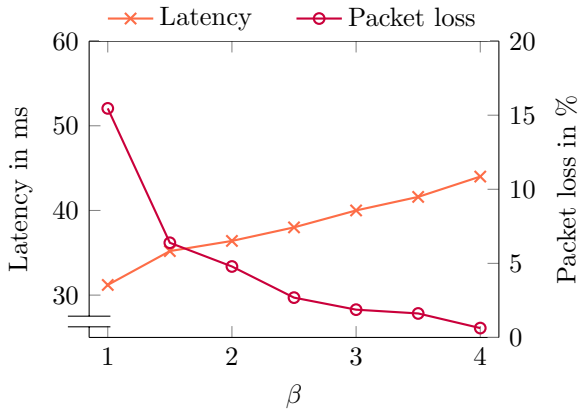

Figure 9: Latency and packet loss against packet loss tuning factor β

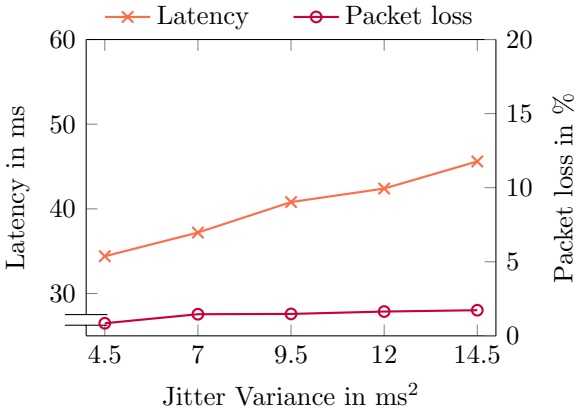

Figure 10: Adaption of latency and packet loss to various jitter conditions

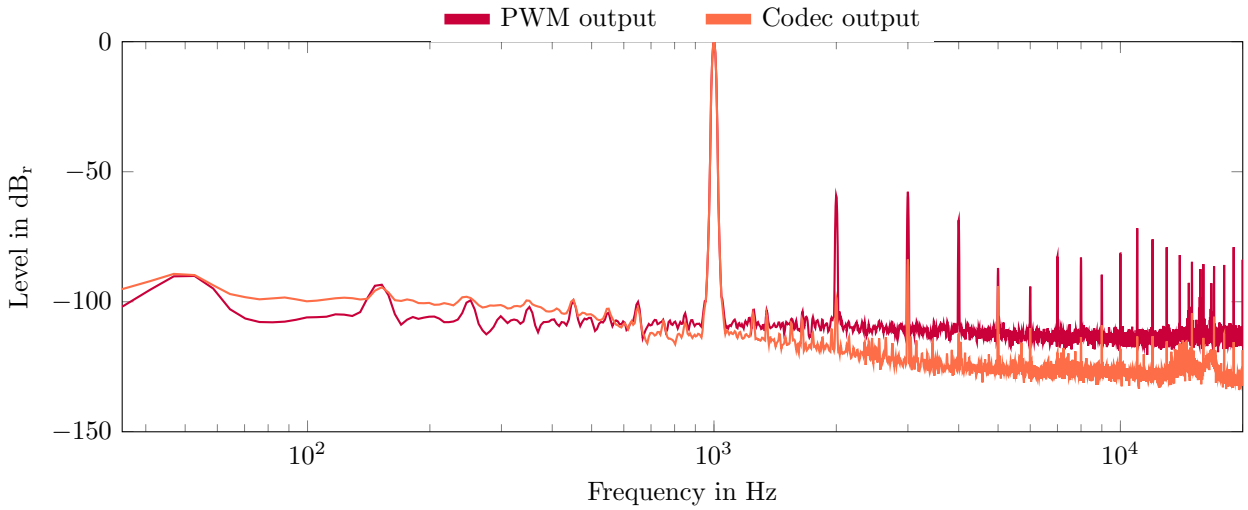

Figure 12: Spectra of the Raspberry Pi PWM and the codec board output for an 1 kHz sine

the THD of the new codec board is at least −20 dB lower than the original output for all analyzed signal levels.

These outcomes and the corresponding measurement results of the other audio hardware modules are listed in Tab. 1 and 2. The identified values should allow a high-quality capturing and playback of instruments. Analog amplification is always connected with the addition of noise. Therefore, the values of the input amplifier decrease with an increase of gain. For achieving even better quality, the flexibility of the device allows for connection of almost any kind of music equipment, like favored guitar amplifiers or vintage synthesizers.

7 Conclusions

The goal of this project was to create a standalone device, called the JamBerry, capable of delivering the well-known experience of a distributed network performance in a user-friendly way. The device is based on the famous Raspberry Pi and is enhanced by several custom hardware extensions: a digital and an analog extension board, providing high-quality audio interfaces to the Raspberry Pi, and a touchscreen to allow standalone operation of the device.

The performance was evaluated under lab conditions and the authors assume that the system and especially the audio quality shall satisfy the need of most musicians. Besides the described device design proposal, the main author shares the ALSA kernel driver that is included in the Linux mainline kernel since version 3.14 allowing the versatile connection of the Raspberry Pi with external audio hardware.

References

[1] The Raspberry Pi Foundation. Raspberry Pi Homepage. `www.raspberrypi.org`.

[2] Alexander Carôt and Christian Werner. Network Music Performance-Problems, Approaches and Perspectives. In *Proceedings of the "Music in the Global Village"-Conference*, Budapest, Hungary, September 2007.

[3] Alain Renaud, Alexander Carôt, and Pedro Rebelo. Networked Music Performance: State of the Art. In *Proceedings of the AES 30th International Conference*, March 2007.

[4] Chris Chafe, Scott Wilson, Al Leistikow, Dave Chisholm, and Gary Scavone. A Simplified Approach to High Quality Music and Sound over IP. In *Proceedings of the COST-G6 Conference on Digital Audio Effects (DAFx-00)*, Verona, Italy, December 2000.

[5] Juan-Pablo Cáceres and Chris Chafe. JackTrip: Under the hood of an engine for network audio. *Journal of New Music Research*, 39(3), 2010.

[6] Alexander Carôt, Torben Hohn, and Christian Werner. Netjack - Remote music collaboration with electronic sequencers on the Internet. In *Proceedings of the Linux Audio Conference (LAC 2009)*, Parma, Italy, April 2009.

[7] Alexander Cart and Christian Werner. Distributed Network Music Workshop with

Soundjack. In *Proceedings of the 25th Tonmeistertagung*, Leipzig, Germany, November 2008.

[8] Dimitri Konstantas, Yann Orlarey, Olivier Carbonel, and Simon Gibbs. The Distributed Musical Rehearsal Environment. *IEEE MultiMedia*, 6(3), 1999.

[9] Chrisoula Alexandraki and Demosthenes Akoumianakis. Exploring New Perspectives in Network Music Performance: The DIAMOUSES Framework. *Computer Music Journal*, 34(2), June 2010.

[10] European Broadcasting Union. Specification of the Digital Audio Interface (The AES/EBU interface), 2004.

[11] Stefan Schmitt and Jochen Cronemeyer. Audio over Ethernet: There are many solutions but which one is best for you? In *Embedded World*, Nürnberg, Germany, March 2011.

[12] Chris Chafe, Juan-Pablo Caceres, and Michael Gurevich. Effect of temporal separation on synchronization in rhythmic performance. *Perception*, 39(7), 2010.

[13] Alexander Carôt, Christian Werner, and Timo Fischinger. Towards a comprehensive cognitive analysis of delay influenced rhythmical interaction. In *Proceedings of the International Computer Music Conference (ICMC'09)*, Montreal, Quebec, Canada, August 2009.

[14] Advanced Linux Sound Architecture (ALSA) Project Homepage. `www.alsa-project.org`.

[15] Florian Meier. wamp_cpp - WAMP Server Implementation for C++. `github.com/koalo/wamp_cpp`.

[16] WAMP - the WebSocket application messaging protocol. `wamp.ws/spec`.

[17] Jean-Marc Valin, Koen Vos, and Timothy B. Terriberry. Definition of the Opus Audio Codec. Internet Engineering Task Force, RFC 6716, September 2012.

[18] Jean-Marc Valin, Timothy B. Terriberry, Christopher Montgomery, and Gregory Maxwell. A High-Quality Speech and Audio Codec With Less Than 10-ms Delay. *IEEE Transactions on Audio, Speech, and Language Processing*, 18(1), January 2010.

[19] Marco Fink, Martin Holters, and Udo Zölzer. Comparison of Various Predictors for Audio Extrapolation. In *Proceedings of the International Conference on Digital Audio Effects (DAFx'13)*, Maynooth, Ireland, September 2013.

[20] Colin Perkins, Orion Hodson, and Vicky Hardman. A Survey of Packet Loss Recovery Techniques for Streaming Audio. *IEEE Network*, 12(5), 1998.

[21] Benjamin W. Wah, Xiao Su, and Dong Lin. A Survey of Error-Concealment Schemes for Real-Time Audio and Video Transmissions over the Internet. In *Proceedings of the International Symposium on Multimedia Software Engineering*, Taipei, Taiwan, December 2000.

[22] I^2S Bus. Specification, Philips Semiconductors, June 1996.

[23] Alexander Carôt and Christian Werner. External Latency-Optimized Soundcard Synchronization for Applications in Wide-Area Networks. In *Proceedings of the AES Regional Convention*, Tokio, Japan, July 2009.