Case Study:
Building an Out Of The Box Raspberry Pi Modular Synthesizer

Jürgen Reuter

Karlsruhe,
Germany,
reuter_j@web.de

Abstract

The idea is simple and obvious: Take some Raspberry Pi computing units, each as a reusable synthesizer module. Connect them via a network. Connect a notebook or PC to control and monitor them. Start playing on your virtual analog modular synthesizer. However, is existing Linux audio software sufficiently mature to implement this vision out of the box? We investigate how far we get in building such a synthesizer, what existing software to choose with focus on networking, analyse what limits we hit and what features still need to be implemented to make our vision become reality.

Keywords

Raspberry Pi, Virtual Anolog Modular Synthesizer, Distributed Networked Audio

1 The Vision

The popular *Raspberry Pi* (or, shortly, *RPi*)[Raspberry Pi Foundation, 2014b] is a small, cheap, yet powerful, computing unit with many I/O jacks with Linux/ARMv6 available as operating system (*OS*). It is predestinated for building networks of collaborative modules, with each RPi taking over the role of a synthesizer module with dedicated function as e.g. oscillator, filter or modulator. Using the RPi's *General Purpose I/O* (*GPIO*) pins or SPI interface, only minimal circuitry is required to equip the RPi with knobs (e.g. potentiometers or rotary encoders) or sliders, preferably on a separate, tiny board, also called a *shield*. This way, you get a distributed user interface, with knobs and sliders located directly on the module that it controls. Modules can be added to or removed from the network in a hot-plugging manner. If, for a different setup, you need, say, more oscillators and less modulators, you may change the role of a module simply by changing the software that it runs.

Compared with a virtual analog modular synthesizer running on a notebook or PC, the approach of a network of RPis reveals several advantages:

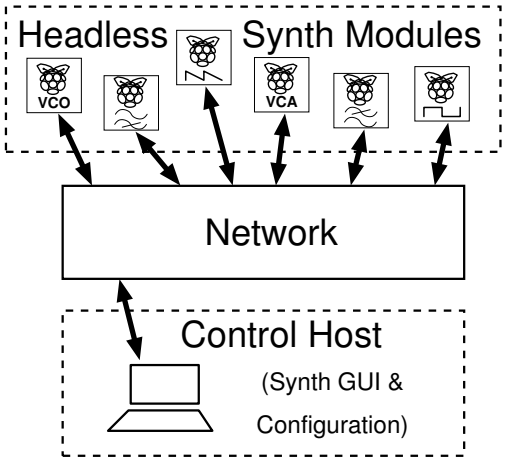

Figure 1: The Vision

- **Dedicated System.** The RPis are solely used for synthesis. The OS, residing on an SD card, can be tailored to this purpose. Many services are irrelevant for headless mode or use in a synthesizer and thus need not be installed, thus saving space and CPU time. The Linux audio RPi page[Linuxaudio.org, 2014] lists many tips for tuning overall latency. Once running, infrequent software updates should suffice, such that the chance to break the installed software e.g. with incompatible libraries can be reduced.
- **Distributed computing.** While the performance of the notebook or PC can easily become a bottleneck, in the distributed network audio computing performance scales with the number of modules.
- **Distributed interface.** With a single mouse and keyboard, you can control only a single input (such as a slider or knob) at one time. Optionally, the RPis can be

equipped with their own sliders and knobs, enabling to control them in parallel. Also, you may place the modules on, say, a large table, while on the virtual desktop of your notebook or PC, you are limited to size of your screen display.

- **Authenticity.** In a live performance, it is more comprehensible for the audience to see a musician put hands on physically existent modules and hear the resulting change of sound, rather than watching a PC user clicking and typing on his computer.

Still, a notebook or PC maybe useful as host for controlling network connections between the modules.

2 The Hardware

Our vision just integrates existing software and hardware, one may think. In fact, we have to carefully choose software that smoothely integrates with our RPis. We look at the RPi's hardware to better understand software requirements.

2.1 Audio Connections

For building an audio network, we have to decide what interconnects to use for audio transmission. Essential criteria are:

- **Duplex operation.** Synthesizer modules typically have both, input and output. We do not want to add hardware to gain full duplex operation.
- **Bandwidth.** Bandwidth must be sufficiently high for carrying multiple channels.
- **Audio and control data.** For low bandwidth data such as envelopes or frequency control, low bandwidth connections (e.g. MIDI) should be supported to save overall bandwidth.
- **Hardware protocol support.** To save computing resources, low-level issues (e.g. parity check bits or serializing / deserializing) should be implemented in hardware.

The RPi's hardware connectors capable of transmitting audio include the analog 3.5" audio output jack, USB, HDMI, GPIO / I2S, and Ethernet (Fig. 2).

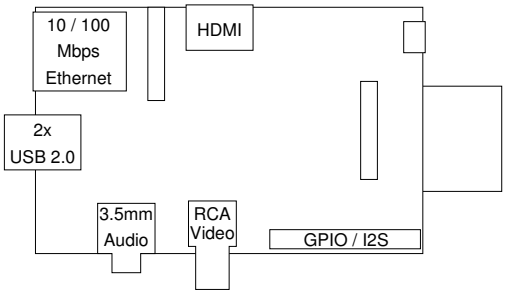

Figure 2: RPi Connectors Relevant for Audio

2.1.1 Analog 3.5" Audio Output Jack

Users report glitches, crackles and pops when using the 3.5" jack, at least in the early days of the RPi. It appearently supports only 11 bits of resolution [Linuxaudio.org, 2014]. Most important, the RPi has no analog audio input. Analog audio can not feed back into the RPi without additional hardware; hence we do not persue this jack.

2.1.2 USB

Linux supports audio over USB. The RPi model B's built-in USB 2.0 connectors only support connecting a USB device, but not another USB host[Raspberry Pi Foundation, 2012d]. Similar to the analog audio jack, symmetrical host-to-host transmission is impossible for RPi model B. In theory, RPi model A's single USB port can make the RPi act as device, via a host-to-host USB cable, but there does not seem anyone on the Web having confirmed that the USB drivers support this mode.

2.1.3 Audio via HDMI

While audio can be transmitted via HDMI, the RPi's built-in hardware does not support HDMI input; hence we do not persue this option.

2.1.4 GPIO pins

The RPi's GPIO pins are ideal for low-level input and output of binary data. For streaming audio data over GPIO, we would have to implement a full protocol stack in software, thus consuming much computing resources and limiting bandwidth. Specific GPIO pins implement I2S with hardware support[Raspberry Pi Foundation, 2012c]. While this interface may provide sufficient performance (users report varying experiences on this issue[stackexchange.com, 2013]), we would need special hardware (e.g. an I2S router). Some users claim that the kernel needs to be patched with an I2S kernel module to achieve high performance for audio data over I2S[GmbH, 2013]. At least, special I2S audio

drivers would have to be implemented to make audio applications aware of I2S.

2.1.5 TCP/IP or UDP over Ethernet

The RPi's Ethernet plug can be used for transmitting audio data. Neither TCP/IP nor UDP have been designed for realtime applications, but existing software for audio and video streaming over the internet shows that, with some effort, streaming is feasible as long as network bandwith is sufficiently high. We persue the approach of streaming audio over ethernet.

3 The Software

Preferring Ethernet for audio transmission, we next look into the software to choose and how to set it up. We prefer an out of the box solution of open source software.

3.1 Choosing Proper Audio Streaming Software

On Linux, there are competiting sound servers for streaming audio data over a network. In our short survey the following criteria are essential:

- **Availability.** The software must be available for ARMv6. Software, that is not open source or not under active development, is typically not available for this architecture.
- **Latency.** In modular synthesis audio and control data typically follow a path running through many modules. Therefore, the sound server should care for low latency.
- **Headless use.** The RPis will be typically controlled remotely and therefore run headlessly, that is without a display connected to them. No graphical environment such as X11 or a window manager should be required to run.

Audio streaming software like the *Enlightened Sound Daemon (ESD)* or *Phonon* are bound to a window manager. Therefore, ESD and Phonon are no valid candidates for our purpose. *sndio* is an audio server for OpenBSD, however, we are looking into a solution for Linux/ARM. The *aRts* sound server is out of development since 2006 and therefore not a viable choice. A quick internet search yields the following candidates:

- Network Audio System (NAS)
- PulseAudio
- JACK with netJACK

3.1.1 Network Audio System (NAS)

The Network Audio System (NAS)[radscan.com, 1996 2013] does not (yet) list any ARM architecture as supported platform. The man page of NAS states that the server automatically converts all data to the designed format or rate, that is, resampling may slow down overall performance.

3.1.2 PulseAudio

PulseAudio supports streaming over networks[freedesktop.org, 2013; archlinux.org, 2012 2014]. However, latency seems to be a major problem; only recently, major improvements have been announced[Lindner, 2013]. Also, PulseAudio resamples all data into some internal format and again resamples it for delivery. The RPi's limited computing performance should be saved for the actual audio processing of the synthesizer module's function.

3.1.3 JACK with netJACK

In contrast to PulseAudio and NAS[jack devel@jackaudio.org, 2012], JACK[jackaudio.org, 2006 2014] synchronizes all clients to one sound sink. JACK has been designed from the beginning with low latency in mind. Over the last few months, some remaining bugs that specifically appeared on the RPi/ARMv6 platform, have been fixed. We decide to pursue JACK, using (jackdmp) version 1.9.9. On our control host notebook, there was a pre-installed JACK (jackdmp) version 1.9.8 that we continue using. Any newer version should also work.

3.1.4 netJACK1 vs. netJACK2 vs. JackTrip

netJACK1 is available for both JACK1 and JACK2. In JACK1, netJACK1 is loaded with the command jackd -R -d net, while netJACK2 is not available. In JACK2, netJACK1 is loaded with jackd -R -d netone, while netJACK2 is loaded with jackd -R -d net.

The graphical application qjackctl can be used to load and configure netJACK1 or netJACK2 as backend. However, as of qjackctl version 0.3.9 bundled with current NOOBS, several bugs render qjackctl effectively useless when used with the netJACK2 backend on the RPi. Particularly, it uses netJACK1 options such as -o4 instead of -P 4 for setting up 4 output channels, and exhibited problems to detect an already running JACK instance. We prefer to use the RPis in headless mode anyway, i.e. without

using qjackctl. Running qjackctl on the control host seems to be fine for our purposes.

The documentation of JackTrip[Caceres and Chafe, 2010] explains how to setup a single JackTrip server with a single JackTrip client. The JackTrip server is a stand-alone application that, when started, appears as regular JACK client. When trying to start another JackTrip server instance, it complains that the associated UDP socket is already in use. The single JackTrip server instance spawns only a single readable client in qjackctl. The JackTrip documentation does not show any appearent way to connect multiple JackTrip clients. The latest ChangeLog entry of JackTrip dates from November 2010. As we need to connect multiple clients, we do not further persue JackTrip.

3.2 Putting it All Together

Next, we give a step by step instructions for installing and configuring all software for an RPi based modular synthesizer with JACK and netJACK.

3.2.1 Setting up NOOBS on the RPi

By now, there is no distribution tailored for audio applications on the RPi. Instead, we use the New Out Of Box Software (NOOBS) version 1.3.2 (Debian 3.10.24+ #614 PREEMPT armv6l kernel) based on the Wheezy Raspian distribution. We follow the instructions on the download webpage[Raspberry Pi Foundation, 2014a] and on the screen display, that we needed to attach to the RPi solely for the first installation, as well as a keyboard. Once one system is running, no more display or keyboard is needed. The SD card's contents can be copied to create another OS instance for another RPi.

When asked by NOOBS to select a distribution, we choose Raspian (i.e. Debian wheezy). With 16GB class 4 SD card and 10 MBit/s internet connection, the following installation roughly takes 45 minutes, including several automatic reboots. Finally, the raspberry configuration tool raspi-config is executed. The preset defaults should be fine.

JACK including netJACK should be already installed. For developing and compiling JACK clients, you need to install C header files for JACK with the command sudo apt-get install libjack-jackd2-dev, that installs the packages libdbus-1-dev and libjack-jackd2-dev. If there is no DHCP server on your network, do not forget to statically assign a unique IP address to each RPi and to set up a proper route to your network.

Now we have basic NOOBS installed on the RPi. Name for login on the RPi is pi, password is raspberry.

3.2.2 Setting Up JACK on the RPi

To automatically start a JACK slave instance on each RPi upon boot, put the following line into the /etc/rc.local script:

```
sudo -u pi /usr/bin/jackd -R -d net -n
module-name >/dev/null 2>&1 &
```

The instance will then appear to the master JACK instance on the control host as a remote slave instance called module-name. Alternatively, JACK can be started implicitly by the client application. Say, you have a low-pass filter implementation that connects to JACK with

```
jack_options_t options = JackNullOption;
client = jack_client_open (client_name,
         options, &status, server_name);
```

and your .jackdrc configuration file in your home directory containing the following line:

```
/usr/bin/jackd -R -d net -n low-pass \
               -C 3 -P 4
```

Then starting your client application will also start JACK. That is, in your /etc/rc.local script, you can also directly launch your client application.

3.2.3 Setting Up JACK on the Control Host

On our notebook we use JACK 1.9.8. The JACK master instance is started with the command jackd -R -d alsa or with whatever backend else you prefer over ALSA. After that, we load netJACK with the command jack_load netmanager, such that all JACK slaves on the RPis may connect to the JACK master.

Note that the JACK master on the control host must be started first. After that, boot the RPis. If JACK is automatically started on the RPi, then it will become visible to the control host. Use qjackctl on the control host to connect all RPis' inputs and outputs. Start playing your distributed RPi based modular synthesizer.

3.3 Synthesizer Software

Throughout this work, the author used very simple self-written JACK clients based on the simple_client.c example of the JACK distribution. In our out of the box spirit, we want to apply those existing LV^2 plugins[LV2, 2014]

for actual synthesis that do not require a GUI. Therefore we need a simple JACK client serving as host for LV^2 plugins that examines a given LV^2 plugin's I/O lines and exports them as JACK channels. The author does not know of an existing software doing this job, but the effort should not be too large. This work still has to be done.

4 Advanced Module Identity and Identification

For simplicity and ease of use, each RPi should represent exactly one synthesizer module. Then each RPi has a dedicated, clearly distinct function, helping to keep clear oversight over the whole system. This approach looks like waste of computing resources, if, for example, one RPi is dedicated as an oscillator. However, even tasks looking as simple as an oscillator may evolve into high complexity when adding sophisticated input controls, for example for morphing between sounds. We identify three options of identification: remote setup, SD card based identity, shield based identity.

4.1 Remote Setup

The software on the RPi's SD card contains all software for all supported module types, e.g. oscillator, low-pass filter or reverb effect. The function of a specific RPi is determined by remotely configuring it on the control host.

4.2 SD Card Based Identity

As the RPi uses its SD card as resident memory with complete OS and application software on it, the RPi gets a complete new identity by simply replacing the SD card. That is, we may create an oscillator SD card, a low-pass filter SD card, a reverb effect SD card, etc. The RPi represents a particular type of synthesizer module just by inserting the appropriate SD card. The RPi announces itself e.g. as oscillator or low-pass filter or reverb effect. The control host will collect all announcements and present all available modules to the user for wiring.

4.3 Shield Based Identity

For most modules, it is useful to provide hardware knobs or sliders directly attached to the modules, using a shield mounted directly on the RPi. While this approach requires (little) extra hardware and thus is not a pure out of the box solution, it has substantial advantages:

- **Visual module identification.** The extra hardware gives the RPi a visual identity and emphasizes that RPi's dedicated function. Each shield can be individually labelled (e.g. "master reverb effect") and typically has a set of knobs or sliders that also may help identify its function.
- **Parallel control of hardware knobs and sliders.** When controlling a virtual knob or slider on a screen, you need to place the mouse pointer on it. That is, only one input control can be used at a time. Switching between knobs or sliders takes time for relocating the mouse pointer. Hardware knobs and sliders enable parallel use and fast switching.
- **Module identity change by shield replacement.** If the shield provides an identifier for the RPi that represents the module's intended function (e.g. an LADSPA plugin ID), the RPi may automatically start any associated software or plug-in that implements the module's function indicated by the shield.

This way configuring an RPi as a dedicated module boils down to connecting a specific shield with knobs and sliders to it. The RPi's SD card holds the software for any supported module, and when plugging in a shield, the RPi can determine which module to represent.

4.4 Shield Design

While the author has not (yet) developed a shield, the design idea is simple and straightforward. We need circuitry connected to the RPi's GPIO pins that converts input from analog controls like knobs or sliders into digital signals. There are shields available with exactly this feature[abelectronics.co.uk, 2013; Raspberry Pi Foundation, 2012a; Modern Device, 2014]. Even cheap A/D converters are sufficient for low frequency signals such as movements of knobs and sliders[Sklar, 2012] and accessible via SPI[Brownell and others, 2013; Gzamboni, 2013]. Rotary encoders can be directly connected to the GPIO pins, as they simply consist of mechanical switches. The shield should contain a small serial EEPROM for uniquely identifying or describing the module's function and maybe storing a user-defined module label. The label must be stored on the shield, not on the RPi's SD card, as is names the module's function as stamped by the shield, regardless of the particular underlying RPi. Maybe designing and producing proper

shields for our synth modules can emerge as candidate for a tiny crowdfunding project.

5 Evaluation

Our attempt to set up a modular synthesizer using out of the box software shows remarkable limitations that should be considered as feature requests for the software that we discuss.

5.1 Audio Data Routing

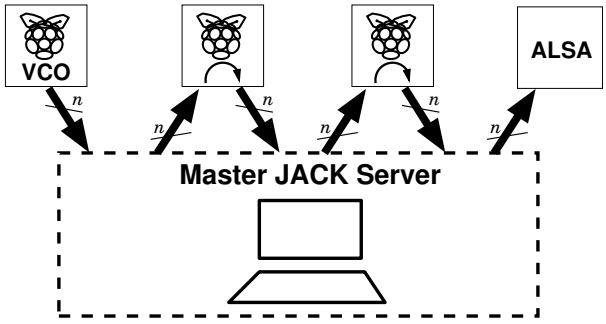

Figure 3: Routing via Master JACK Server

The probably most obstructive issue is the central routing of all audio data via the master JACK instance. The netJACK2 approach assumes a single master instance[Stéphane Letz et al., 2009]. Similarly, in netJACK1 a slave can only be connected to one master at a time. The master's backend (e.g. an ALSA sound device) determines the sample rate and format for all commnunication with all participating JACK slave instances. To prevent the master becoming a bottleneck, we would prefer RPi hosting a slave and a master JACK instance.

The qjackctl application follows the JACK and netJACK design and provides a GUI for configuring routing between the single master and multiple clients / slaves. If in a future version of JACK / netJACK multiple masters were supported, we would like to have an extended version of qjackctl capable of managing connections between two remote master instances.

For evaluating the impact of central routing, we measured the master JACK CPU load as a function of the number of audio channel connections. We connected three RPis to our notebook (Quad Core i5-2430M @ 2.4 GHz), used as control host for running the master JACK server. One of the RPis served as array of n oscillator outputs; the other two modules just looped through data from their n input channels to their n output channels. That is, for each channel audio data flows from the oscillator to the notebook, then to the first loop-through module and back to the notebook, then to the second loop-through module and back to the notebook and finally to the ALSA backend device (Fig. 3). That is, for n channels, there are $6n$ connections configured in qjackctl.

We measured the CPU load reported by jack_cpu_load() and varied the number of channels per module. Each box plot shows the range of CPU load of 480 samples (1sample/sec × 8min). For $n > 18$ (i.e. > 108 connections), severe problems like xrun errors and JACK crashes arised. Below this threshold, the system behaved smoothly with moderate load (Fig. 4). For comparison, the overall CPU load shown by xosview kept below 0.7.

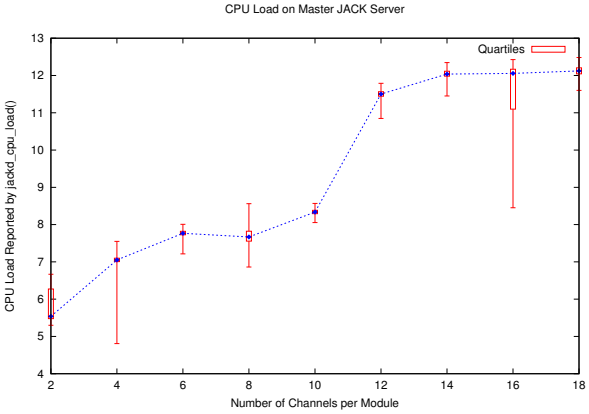

Figure 4: CPU Load on Master JACK Server

qjackctl often unexpectedly exited (with exit code 0), when a remote client reappeared after reboot. Sometimes it did not recognize when the connection to a remote client broke down due to client shutdown (maybe due to some problem of netJACK2). The master JACK server crashed when trying to connect more than 21 channels to the ALSA backend.

5.2 Labelling of Modules

On our control host notebook running qjackctl, by default all JACK slaves appear as clients with the name of the host they are running on. While this is reasonable default behaviour, we have a network of RPis with basically identical software setup. Therefore all RPis will appear as JACK clients labelled "raspberry" – the default host name for the Raspian Linux distribution. We could configure individual hostnames for each RPi, but it is the type of synthesizer module that should be displayed rather than a host identifier. Also, we do not want to change the host name each time the RPi changes the type of module. Luckily,

netJACK2 provides command line option -n to explicitly set the client name. For example, jackd -R -d net -n low-pass will result in a client called low-pass (Fig. 5). In netJACK1, there is no comparable option.

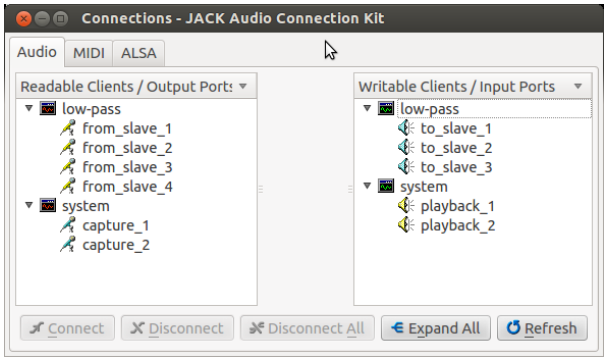

Figure 5: Slave started with jackd -R -d net -n low-pass -C 3 -P 4

5.3 Number of Channels

By default, a netJACK client is started with a number of audio input and output channels equal to that of the soundcard, i.e. mostly 2 for stereo sound. In contrast, a synthesizer module may have an arbitrary number of inputs and ouputs. Luckily again, the netJACK2 backend provides the command line options -C, -P, -i, and -o to specify the number of audio input, audio output, MIDI input, and MIDI output channels, respectively (Fig. 5). In netJACK1, the corresponding options are named -i, -o, -I, -O, respectively.

5.4 Labelling of Channels

The module software should be able to configure the names of a JACK slave's audio and MIDI input and output channels individually. For example, an oscillator may have a pitch control input, a noise content control input, a left channel audio output and a right channel audio output.

In JACK, port names can be set with the function jack_port_set_name(jack_port_t *port, const char *port_name). They are initially set to capture_n or playback_n for inputs or outputs, respectively. When the function is executed on the RPi, it refers to the port name locally shown on the RPi. Unfortunately, netJACK does not report slave port names to the master. Instead, on the master JACK instance, remote client channels always appear as from_slave_n, to_slave_n, midi_from_slave_n, and midi_to_slave_n (Fig. 5).

Of course, if software running on the control host has knowledge about all synthesizer module types that may appear in the network, it may derive labels channels labels from the slave's name. This workaround however does not qualify as out of the box solution.

5.5 Boot Time

Our NOOBS based setup takes almost one minute of time for booting an RPi. For an embedded system that you want to immediately start working with, this time is far too long. There exist tailored kernels and software configurations for faster booting, reduced to a minimum of what is required for the dedicated purpose. Choosing a fast SD card may speed up booting as well as using the kernel's fastboot option. RPi users report tips und tricks to reduce its boot time to as low as less than 20 seconds[Raspberry Pi Foundation, 2012b].

6 Conclusions

Linux on Raspberry Pi is *almost* mature for implementing our vision of a modular synthesizer based on a network of RPis connected to a notebook or PC as control host. The most outstanding problem in our setup is that *all* audio and MIDI data is routed through the master JACK instance running on the control host. Instead, the RPis should be able to communicate directly with each other. This approach however would require multiple JACK master instances to be part of the communication network, while the netJACK architecture currently assumes a single master instance.

As the RPis are run in headless mode, on the control host we would like to run an application capable of configuring the complete system. In particular, setting up direct connections between two RPis (once it will be supported) reaches beyond the scope of qjackctl. An extended version of qjackctl could turn out essential for our vision. The overall stability of qjackctl should be improved.

For using existing LV^2 plugins, there is missing some JACK client serving as LV^2 plugin host.

The author plans to get in contact with the authors of the depicted software to solve remaining issues. The example & testing code of this study is available at http://www.soundpaint.org/rpi-modular-synth/. The author wants to thank the anonymous reviewer for pointing out some missing point for true out of the box spirit and a further reference.

References

abelectronics.co.uk. 2013. ADC Pi - 8 Channel Analogue to Digital converter for the Raspberry Pi computer boards ADC PIV2. http://www.abelectronics.co.uk/products/3/Raspberry-Pi/17/ADC-Pi-V2---Raspberry-Pi-Analogue-to-\Digital-converter.

archlinux.org. 2012–2014. Pulseaudio/examples - archwiki. https://wiki.archlinux.org/index.php/PulseAudio/Examples#PulseAudio_over_network.

David Brownell et al. 2013. spidev. https://www.kernel.org/doc/Documentation/spi/spidev.

Juan-Pablo Caceres and Chris Chafe. 2010. JackTrip: JackTrip Documentation. https://ccrma.stanford.edu/groups/soundwire/software/jacktrip/.

freedesktop.org. 2013. Network. http://www.freedesktop.org/wiki/Software/PulseAudio/Documentation/User/Network/.

Modul 9 GmbH. 2013. Hifiberry dac – linux configuration — crazy audio. http://www.crazy-audio.com/projects/hifiberry-mini/hifiberry-mini-linux-configuration/.

Gzamboni. 2013. SPIdev. http://linux-sunxi.org/SPIdev.

jack devel@jackaudio.org. 2012. Discussion of the jack audio server and jack applications: Netjack for Thinclients Instead of Pulseaudio. http://comments.gmane.org/gmane.comp.audio.jackit/25406.

jackaudio.org. 2006-2014. JACK — connecting a world of audio. http://jackaudio.org/.

Mirko Lindner. 2013. PulseAudio 4.0 verringert Latenz und steigert Geschwindigkeit - Pro-Linux. http://www.pro-linux.de/news/1/19858/pulseaudio-40-verringert-latenz-und-\steigert-geschwindigkeit.html.

Linuxaudio.org. 2014. Raspberry Pi and realtime, low-latency audio [Linux-Sound]. http://wiki.linuxaudio.org/wiki/raspberrypi#on-board_audio.

LV2. 2014. LV2 Trac. http://lv2plug.in.

Modern Device. 2014. Lots of Pots Board for Raspberry Pi. http://moderndevice.com/product/lots-of-pots-lop-board-for-raspberry-pi.

radscan.com. 1996–2013. The Network Audio System (NAS). http://www.radscan.com/nas.html.

Raspberry Pi Foundation. 2012a. Gertboard — Raspberry Pi. http://www.raspberrypi.org/archives/tag/gertboard.

Raspberry Pi Foundation. 2012b. How to speed up boot time if run headless? http://www.raspberrypi.org/phpBB3/viewtopic.php?f=29&t=25777.

Raspberry Pi Foundation. 2012c. Raspberry Pi – I2S: Anyone got it running? (answer is yes!). http://www.raspberrypi.org/phpBB3/viewtopic.php?t=8496.

Raspberry Pi Foundation. 2012d. Raspberry Pi - Model A Q: Can it be USB client instead of USB host? http://www.raspberrypi.org/phpBB3/viewtopic.php?f=63&t=15696.

Raspberry Pi Foundation. 2014a. Downloads — Raspberry Pi. http://www.raspberrypi.org/downloads.

Raspberry Pi Foundation. 2014b. Raspberry Pi — An ARM GNU/Linux box for $25. Take a byte! http://www.raspberrypi.org.

Mikey Sklar. 2012. Analog Inputs for Raspberry Pi Using the MCP3008 — Adafruit Learning System. http://learn.adafruit.com/reading-a-analog-in-\and-controlling-\audio-volume-with-the-raspberry-pi/overview.

stackexchange.com. 2013. hardware – How fast is GPIO+DMA? Multi I2S input – Raspberry Pi Stack Exchange. http://raspberrypi.stackexchange.com/questions/9646/how-fast-is-gpiodma-multi-i2s-input.

Stéphane Letz et al. 2009. Walk-Through/User/NetJack2 Jack Audio Connection Kit – Trac. http://trac.jackaudio.org/wiki/WalkThrough/User/NetJack2.