FH AACHEN
UNIVERSITY OF APPLIED SCIENCES

Fachhochschule Aachen

Department of Electrical Engineering and Information Technology

MusiUeINet

Making Music over the Internet

Interdisciplinary Project

Nils Angelmann
Marcel Paturej
An-Phong Pham Dinh

Project Idea / Supervisor: Dipl.-Ing. Stephan Borucki

Aachen, Germany, July 2022

(English summary by Nils Angelmann, October 2024)

Contents

1. Introduction

This report summarizes the results of an interdisciplinary project realized at *FH Aachen* in 2022.

When Stephan Borucki encountered numerous problems while trying to remotely rehearse with his choir during the pandemic in 2020 and 2021, he had the idea of starting an interdisciplinary project. Being an excellent option for remote rehearsals, the free and open source software *Jamulus*[1] presented some challenges, however.
Thus it was made the goal of this project to find ways of making *Jamulus* more user friendly for musicians with no background in technology and to determine the decisive factors having the biggest influence on performance.

The initial plan included publishing the results shortly after the end of the project and writing instructions for recreating the test setups. Unfortunately, that never happened due to time constraints and a lack of interest in *Jamulus* after the pandemic had ended. Even though things may have already changed in the meantime, we would hereby like to publish at least our most important results. That might be helpful to someone who uses *Jamulus* with a larger group of musicians or would like to achieve the best possible performance.

All results of this project refer to the version of Jamulus that was up-to-date at the time of testing in 2022. Some results may thus be no longer applicable to newer versions.

[1] jamulus.io

2. Optimizations

In the first part of the project we tried to understand the influence of all the settings inside *Jamulus* as well as the used components in general, in order to achieve the best possible performance.

It quickly became obvious, that the biggest impediment to a great musical performance over the internet is latency, which should therefore be minimized. This however results in a trade-off between latency and stability.

2.1 Latency

In this context *latency* refers to the delay that occurs before a musician hears themself through their headphones.
A standout feature of Jamulus is synchronising those delays between all musicians, allowing all participants to be in time. As a result all other musicians have to "wait" for the musician with the highest latency. However, in comparison to applications designed for video calls Jumulus' latency still is considerably lower.

From sound recording it is well known that latency can have significant effects on a musician's performance. The amount of tolerable latency is up to debate. Based on the following examples it can be assumed that there is no definitive answer:
A) For a drummer hearing the sound of a drum stick hitting the drum delayed by 10 milliseconds it might already be challenging to steadily keep the tempo while playing an up-tempo song.
B) The members of a big orchestra or choir could be up to 15 meters apart, which equals a delay of 44 milliseconds given the comparatively low speed of sound (343 m/s).

As a first test it was decided to check whether the stated latency figure in Jamulus was accurate and could be relied on. Therefore, a burst signal was fed into an audio interface. By monitoring this input signal and the corresponding return signal on an oscilloscope, the latency reported in Jamulus could be confirmed (figure 2.1).

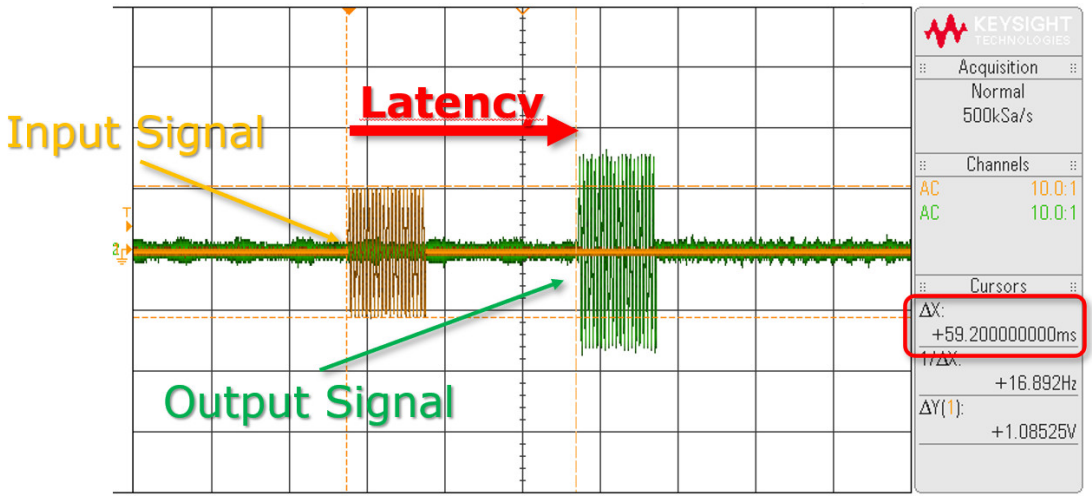


Figure 2.1: Latency measurement utilizing an oscilloscope and a signal generator

This overall latency through Jamulus is the sum of numerous factors. So, in order to minimize the latency it is necessary to understand the influence of each individual factor.

Operating System

A question that came up early, was whether the amount of background tasks on Windows (e.g. updates) might have an impact on Jamulus' performance/latency, which in return could be an advantage for Linux. However, a simple side by side test between a laptop with a fresh installation of *Windows 10* and a *Raspberry Pi 4* running the then latest version of *Raspberry Pi OS* showed no significant difference under identical conditions.
(This might not apply for a system with lots of bloatware and background tasks though.)

Audio Interface

Among musicians with an interest in recording their music, many different USB audio interfaces from various manufacturers can be found. All these devices have in common that they require a buffer to ensure a steady stream of data over the USB connection. For most audio interfaces the size of this buffer can be set in their corresponding driver as a number of samples (usually a power of 2). Setting the buffer too high leads to unnecessary latency while decreasing the buffer size too far can cause dropouts in the audio signal and should therefore be avoided.
The smallest possible buffer size for a stable operation highly depends on the model of audio interface. Once the ideal buffer size for a given audio interface has been found, its latency can be treated as a constant number.

$$Interface\ Latency = \frac{1}{Sample\ Rate} * Buffer\ Size * 2 \tag{2.1}$$

Example: At a common sample rate of 48,000 samples per second (48 kHz) a buffer size of 128 samples results in a latency of 2.67 ms per way or 5.33 ms round-trip.

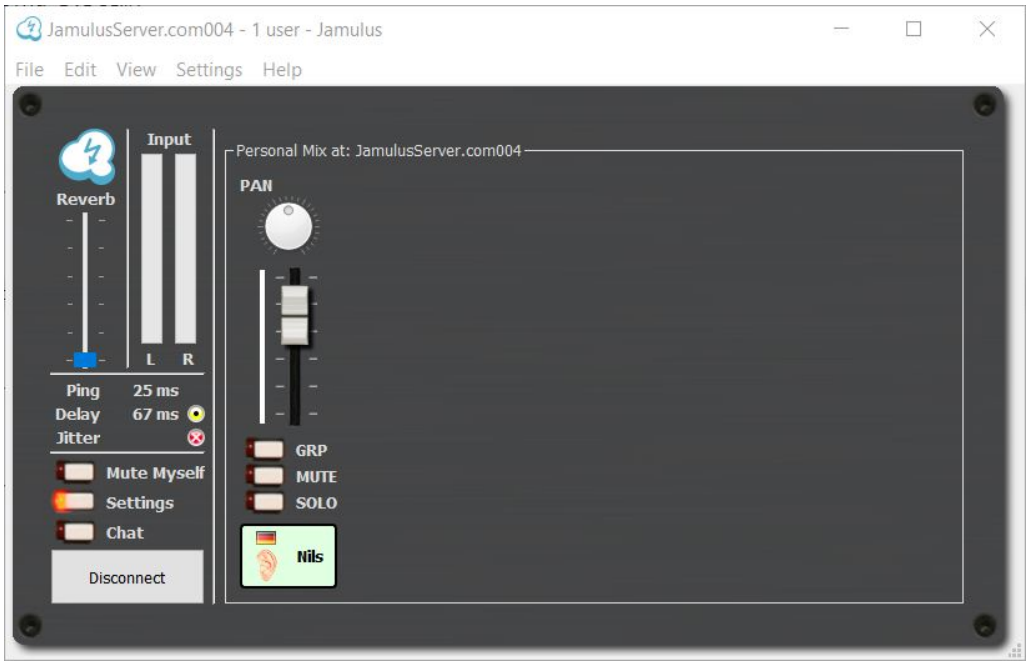


Figure 2.2: Jamulus' user interface: (server) ping and overall delay

Server Location

A fairly obvious cause for latency is the server's ping as also displayed by Jamulus (figure 2.2). This latency is simply caused by the time it takes to transmit data to and from the server. Due to the finite speed of light a noticeable delay is induced by any long enough distance.
When using public servers, it thus makes sense to use a server that is located in close distance to the musicians. Equally, when setting up a private server, e.g. for a remote choir rehearsal during the pandemic, a closely located server provider would be the best choice - even if one half around the globe may be cheaper.
For achieving the best possible performance, the server's distance to the nearest internet hub should also be considered.

Latency Calculation

By treating both ping and the audio interface's latency as known values the remaining causes for latency can be estimated using the average values of some practical experiments (with a local server):

$$Latency = Interface\ Latency + Ping + Buffers\ (+\ Signal\ Processing) \tag{2.2}$$

$$20\ ms = 5\ ms + 0\ ms + x \tag{2.3}$$

$$\Rightarrow x = 20\ ms - 5\ ms - 0\ ms = 15\ ms \tag{2.4}$$

This suggests a latency of roughly 15 ms due to jitter/network buffers and signal processing.

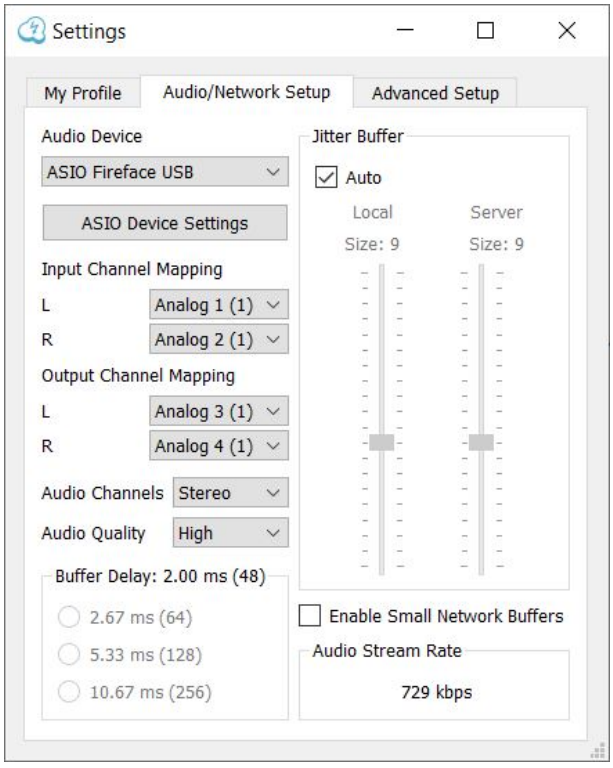


Figure 2.3: Settings menu in Jamulus: Jitter Buffer

Jitter/Network Buffer

While deactivating the *auto* function for the jitter buffer and manually lowering the buffer size seems to result in a slightly lower latency, the connection will most likely be unstable - noticeable as dropouts in the audio signal. The same applies to the option for *small network buffers* - reduced latency at the cost of transmission errors. Like already mentioned on the Jamulus website, changing these settings is absolutely not recommended.
The availability of these options does not really make sense to us.

Server Utilization

We realized a surprising fact when the an identical setup was tested at different times throughout the day: During the day, when the university was crowded, an overall latency of 45 ms was measured. In the early evening, when most people had already left, the delay dropped to 20 ms.
Since the server for those tests was a virtual machine, hosted on the university's server, this implies a big dependence on server utilization. A dedicated root server exclusively used for Jamulus should therefore provide a significant advantage in comparison to a shared or a virtual server.

2.2 Transmission Errors

Besides high latency, transmission errors can have a negative impact on the user experience. While feeding a continuous test signal into Jamulus and observing the returning signal on an oscilloscope, the following two different issues were found:

Clicks

The first issue is the sound of short clicks that - depending on their rate of appearance - can sound similar to a Geiger counter. While those short clicking sounds may be annoying and can certainly render any recordings unusable, they should generally not interrupt a rehearsal situation.

Dropouts

The second type of transmission errors are short dropouts of the audio signal, which have a significantly bigger impact on the musician's performance.

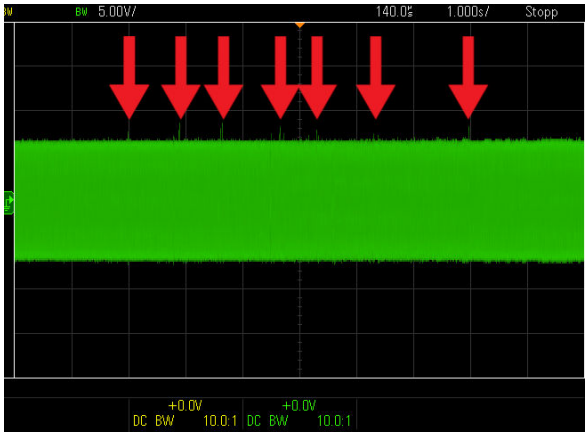


Figure 2.4: Short clicks

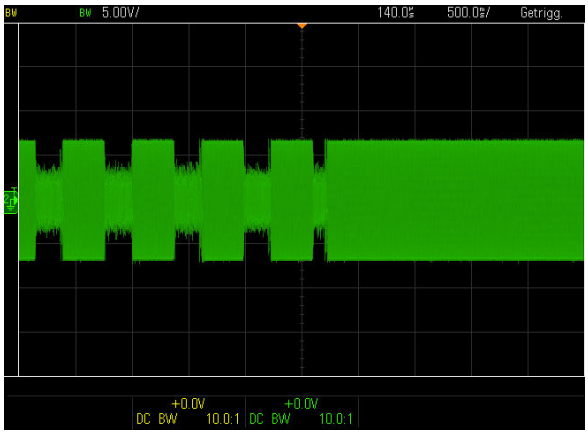


Figure 2.5: Obvious dropouts

Possible Explanation

While it was not possible to identify a definitive cause for the first type of transmission errors, the longer dropouts were only observed on devices that were connected to the internet using WiFi. The dropouts seem to be related to package losses of the WiFi connection. Therefore, a wired internet connection is highly recommended.

2.3 Test Setup

In order to quantify the determined transmission errors, a self-contained test device was developed. This device allows field testing without the need for specialized tools, such as an oscilloscope and a function generator. Different setups and internet connections can thus be easily compared.
Even though the primary goal was to measure transmission errors and although the latency reported by Jamulus was generally found to be correct, a way to measure latency was also implemented.

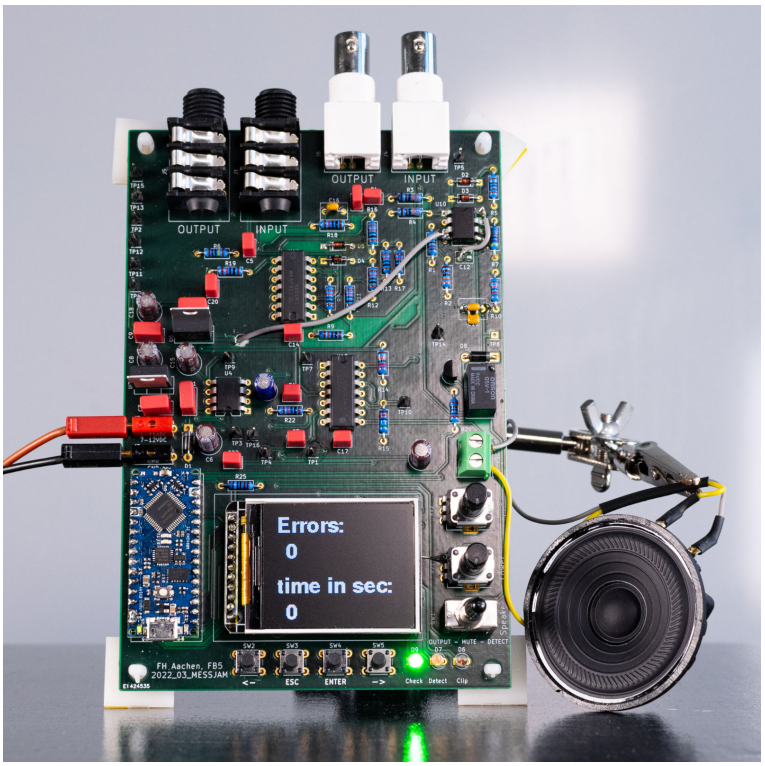


Figure 2.6: Prototype for a self-contained test device

The setup is based around an *Arduino Nano Every* microcontroller development board which is accompanied by some analog circuitry and can be connected to any regular audio interface. Similarly to the first test, latency is measured by sending a non-periodic signal (a simple step function in this case) and recording the time between sending and receiving this signal.
In order to detect transmission errors, a continuous sinusoidal signal, generated by an analog oscillator, gets fed through Jamulus. The returning signal gets rectified and filtered so that a window comparator can be used to decide whether the amplitude is within a carefully calibrated range. This circuit creates a simple (digital) high signal whenever a click or a dropout occurs. The Arduino is then used to count these detected errors.
Different setups can thus be compared on the basis of *transmission errors per minute* as the test device is capable of measuring even high rates of errors which could not be manually counted, e.g. by watching an oscilloscope.
Note: In terms of psychoacoustic aspects it might actually be questionable whether multiple clicks occurring shortly after each other should be counted individually or as **one** *perceived* error.
Also note: The small speaker can be used to inject the test signals into devices that do not feature a line input.

3. Ease of Use

The second goal of this project was to find ways to make Jamulus more accessible to musicians without any special hardware and/or limited technical knowledge.

3.1 Audio Interface

Relying on the built-in audio inputs and outputs of a computer is not a viable option as those cannot be directly accessed by Jamulus on a driver level.
The standard solution is an external USB audio interface as well as a microphone and a pair of headphones. While many musicians already have all the necessary equipment and know how to use it, this solution can present a challenge to anyone with no background in recording music.
The costs related to the acquisition of the necessary equipment can be a hindrance - especially when it comes to equipping an entire orchestra or choir - and providing technical assistance to a large amount of musicians can also be problematic.

Therefore, we suggest the usage of USB microphones, as well as a (probably already existing) pair of headphones. Also and perhaps even more importantly, with this limited amount of equipment the setup process is reduced to plugging in a USB cable.

In order to compare both options, we have tested two exemplary setups:

Setup A (figure 3.1)

- Audio interface: Behringer U-Phoria UM2 (50 €)
- Microphone: Behringer SL 85S (20 €)
- Headphones: Behringer HPM1100-BK (15 €)
- Table stand for microphone: Millenium DS-10 (10 €)
- Microphone cable: Cheap XLR cable (5 €)

Sum: 100 €

Setup B (figure 3.2)

- USB microphone: Mackie EM-USB[1] (60 €)
- Headphones: Behringer HPM1100-BK (15 €)

Sum: 75 €

Note: All prices were taken from Thomann.de (2024-09-17) and slightly rounded.

[1]Thomann's house brand has released the t.bone SC 430 which (at least visually) seems to be a re-branded version of the same microphone while being around 10 € cheaper

In our tests the performance of both setups (each on Windows and on Linux) was absolutely comparable and sufficient for rehearsal duties. Regarding the important factor of latency, there was no difference as both solutions performed best at a buffer size of 128 samples.

Figure 3.1: Setup A: USB audio interface with headphones and a microphone

Figure 3.2: Setup B: USB microphone with headphones

A word of warning: While there is a wide range of USB microphones available, only few models feature hardware controls, i.e. gain (microphone sensitivity) and headphone volume. Instead, most models rely on a specific software which might not be available for all operating systems and complicates the operation for the user in comparison to simple knobs.

3.2 Closed Entity

Another idea that came up during the project was the development of a purpose-built, self-contained unit that would make using Jamulus as easy as possible. Those pre-configured Jamulus clients could then be handed out to a group of musicians by one administrator.

Raspberrry Pi

Based on the successful tests with Linux, a single-board computer, i.e. a Raspberry Pi, seemed to be a good basis. Besides the small form factor and the competitive price, Linux allows for various customizations, e.g. removing any unused features and thereby improving the user experience. This would have been rather difficult on any Android or iOS device.
Windows would require more powerful hardware due to a bigger overhead, making a bigger footprint necessary.

Even though most tests at that time were performed with the current *Raspberry Pi 4*, the tests revealed as well that the older *Raspberry Pi 3* with 1 GB of RAM was indeed powerful enough to run a Jamulus session on *Raspberry Pi OS*. Thereby it was shown that 1 GB of RAM is sufficient and that a version with 2 GB might be a good choice considering future updates while any more RAM is not necessary.

Touchscreen

Considering Jamulus' simple user interface, a touchscreen is an elegant way of interacting with the channel sliders without the need of any external peripherals.

During tests an important disovery was made: Jamulus requires a vertical resolution of at least 600 pixels. Therefore, the *Waveshare 7inch HDMI LCD (C)*[2] with a resolution of 1024x600 px is a suitable option (figure 3.4), while the (B) variant[3] with a resolution of 800x480 px is not (figure 3.4).

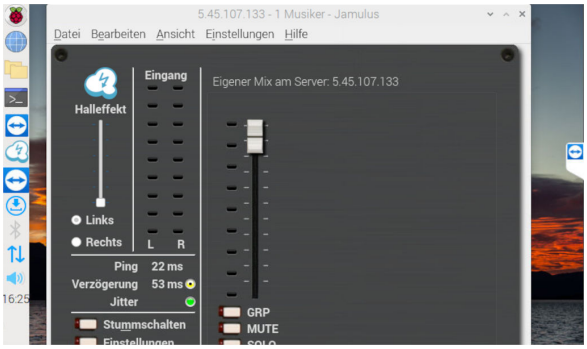


Figure 3.3: Touchscreen (B): 800x480 px

Figure 3.4: Touchscreen (C): 1024x600 px

Since we would suggest having an administrator do the basic configuration beforehand, the user will only need to enter their name. For this a virtual on-screen keyboard should be sufficient (figure 3.5). The *Matchbox-Keyboard* worked well in our tests[4].

[2] waveshare.com/7inch-hdmi-lcd-c.htm
[3] waveshare.com/7inch-hdmi-lcd-b.htm
[4] github.com/mwilliams03/matchbox-keyboard

Figure 3.5: Virtual keyboard: Matchbox-Keyboard

Audio Interfaces

In addition to the previously discussed USB audio interfaces and USB microphones (chapter 3.1) which can be used on any Linux device thanks to the *JACK Audio Connection Kit*[5], there are also specialized audio interfaces for the Raspberry Pi, such as the *HiFiBerry*[6] products. While the form factor would allow for an even tighter integration, there is currently no model that features a microphone input. The development of a Jamulus specific module that features a headphone output and one XLR microphone input would have been a potential solution. However, when it comes to existing solutions we would still recommend an external USB microphone for most applications.

Figure 3.6: Basic setup of a Jamulus client - Raspberry Pi 4 and touchscreen display

[5] jackaudio.org/
[6] hifiberry.com/

Teamviewer

To allow for remote administration of the clients, a tool like *TeamViewer*[7] could be used. In a test, remotely accessing a Raspberry Pi worked as intended and allowed for an easier configuration and setup than by using the touchscreen. However, to allow remote access *TeamViewer* would always need to be running on the client and this carries a serious security risk. Another option would be the manual start of the application in case assistance is needed which is not ideal in terms of usability.
As a solution, every client could have a physical *"HELP!"* button that semi-automatically starts *TeamViewer* and connects to the responsible administrator.

Virtualization

Another potential security risk might be caused by not updating the software regularly. Simply installing current updates could cause other issues, so a known stable version might be used over a prolonged period of time. Running Jamulus in a virtualized environment could theoretically reduce these risks. Since our tests showed serious problems with virtual installations this idea was discared.

Cost

Even though most features were working at the end of the project, a complete prototype has never been built due to the short duration of the project. So, the cost per client can only be estimated:

- Raspberry Pi 4 (Model B 2 GB)[8]: 46 €
- Waveshare 7inch HDMI LCD (C)[9]: 48 €
- microSDHC card (16 GB Class10)[10]: 8 €
- Raspberry Pi USB-C power supply[11]: 8 €
- Custom case (probably 3D printed): 5 €

Sum: 105 €

Note: All prices were taken from rasppishop.de and eckstein-shop.de (2024-09-17) and slightly rounded.

In total each Jamulus client would cost around 190 € including a USB microphone and a pair of headphones (Setup B from chapter 3.1).

[7] teamviewer.com/
[8] rasppishop.de/Raspberry-Pi-4-Modell-B-2GB-SDRAM
[9] eckstein-shop.de/Waveshare7inch
[10] rasppishop.de/Sandisk-microSDHC-16GB-Class10-mit-Raspberry-Pi-OS
[11] rasppishop.de/Raspberry-Pi-15W-USB-C-Netzteil-Schwarz-EU

4. Single-board computer

4.1 Alternatives

In the previous chapter we have discussed only the Raspberry Pi (4 Model B). However, since this project took place during the peak of the chip crisis and Raspberry Pis were extremely difficult and expensive to obtain, we also tested other single-board computers.

4.1.1 Raspberry Pi 3/4/5

As mentioned before, we successfully tested an older Raspberry Pi 3 with 1 GB of RAM. In conclusion, every model starting from generation 3 should work. The newer Raspberry Pi 5 was not yet released at the time of testing but it should work just as well.[1]

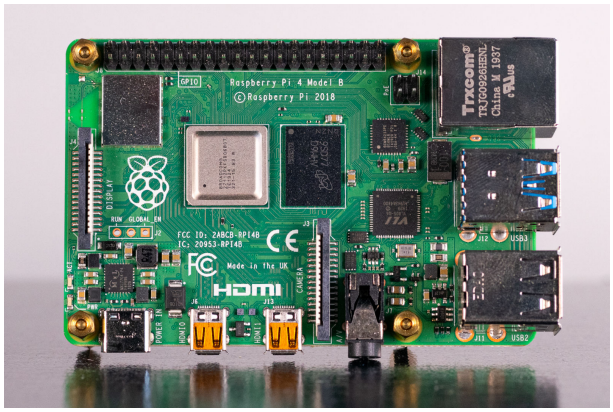


Figure 4.1: Raspberry Pi (4 Model B)

4.1.2 HardKernel ODROID XU4Q

The now discontinued *ODROID XU4Q* features an octa-core processor (Samsung Exynos 5422) and 2 GB of RAM. The manufacturer recommends Ubuntu as the operating system. While Jamulus works on this device, the tested touchscreen does not seem to be supported.[2]

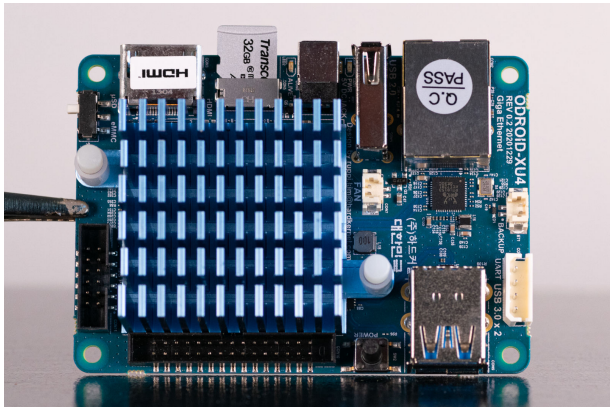


Figure 4.2: HardKernel ODROID XU4Q

[1] raspberrypi.com/products/
[2] hardkernel.com/shop/odroid-xu4q-special-price/

4.1.3 PINE A64-LTS

The slightly larger *PINE A64* (long term supply) features a quad-core cpu and 2 GB of RAM, making it roughly comparable to the Raspberry Pi 3. The recommended operating system *Armbian* is based on *Debian*. Unfortunately, the tested touchscreen does not seem to be compatible with this single-board computer either.

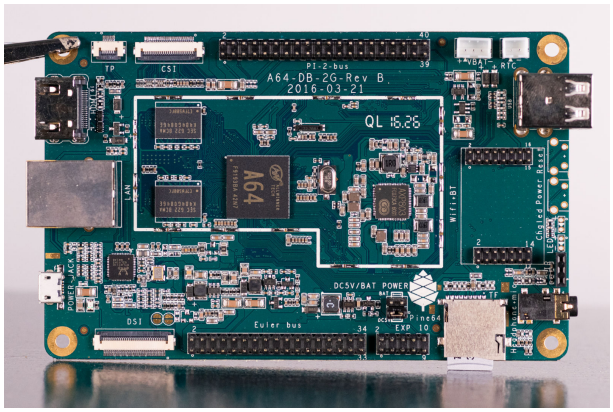


Figure 4.3: PINE A64-LTS

Since supply is no longer an issue, we do not recommend using any of the alternatives. None of those boards come close to the support of the Raspberry Pi's community and while some features worked fine for us, software support can be problematic and the installation process can be tedious.

4.2 Single-board server

Neither the ORDROID nor the PINE single-board computer could be used for the Jamulus client as previously planned. So, when we found out about Raspberry Pi's being used as Jamulus servers[3], we decided to try the same with the ODROID. Thanks to its powerful octa-core cpu it can in fact be used as a Jamulus server.
In general, using a single-board computer can be a cost-effective and energy-efficient option, when a private Jamulus server is needed.

[3] jamulus.io/de/kb/2020/03/28/Server-Rpi.html

5. Conclusion

Based on this project a number of recommendations can be given.

5.1 Ideal Jamulus Setup

In order to achieve the best performance with Jamulus...

- ...avoid using WiFi whenever possible.
- ...leave the network buffer settings inside Jamulus on *auto*.
- ...find the smallest, stable buffer size of your audio interface. A value of 128 samples has proven to be a good starting point.

If you need an audio interface...

- ...the Behringer UM2 is the cheapest recommendable option.
- ...consider a USB microphone for an easier setup.

When setting up your own server...

- ...avoid using a virtual server. A root server will most-likely perform significantly better.
- ...using a Rasbperry Pi can be cheaper than paying for a remote server.
- ...consider the phyisical distance between participating musicians and the server (ping).

5.2 Jamulus Client

If you like the idea of a self-contained Jamulus client...

- ...get the latest Raspberry Pi model. 2 GB of RAM are sufficient.
- ...do not buy a display with a vertical resolution of less than 600 pixels.
- ...consider ways of helping users remotely.

List of Figures

A. Appendix

Script for starting Jamulus & JACK
In order to make the start-up procedure on a Raspberry Pi as simple as possible, a script can be written.

Create a new bash script:

```
sudo nano /usr/bin/toggle-jamulus.sh
```

Content of that script:

```
#!/bin/bash
PIDJACK="$(pidof jackd)"
if [  "$PIDJACK" != ""  ]; then
  PIDJAMULUS="$(pidof Jamulus)"
  kill $PIDJAMULUS & kill $PIDJACK
else
  audioInterface=$(aplay -l | grep Audio | awk '{sub(":", "", $2); print $2}')
  jackd -R -dalsa -dhw:$audioInterface -r48000 -p128 -n2 & Jamulus --connect <IP>
fi
```

(Replace *-p128* with the optimal buffer size for your audio interface.)

Make the script executable:

```
sudo chmod +x /usr/bin/toggle-jamulus.sh
```

In order to create an icon in the taskbar, create another file:

```
sudo nano /usr/share/raspi-ui-overrides/applications/toggle-jamulus.desktop
```

Enter the following lines as the file's content:

```
[Desktop Entry]
Name=Toggle Jamulus
Comment=Toggle Jamulus
Exec=/usr/bin/toggle-jamulus.sh
Type=Application
Icon=Jamulus.png
Categories=Panel;Utility;MB
X-MB-INPUT-MECHANISM=True
```

To make the new icon visible, some changes to the user's taskbar settings are necessary. Open this file:

```
nano /home/pi/.config/lxpanel/LXDE-pi/panels/panel
```

Add the following lines to the end of the file:

```
Plugin {
  type=launchbar
  Config {
    Button {
      id=toggle-jamulus.desktop
    }
  }
}
```

After rebooting the Raspberrry Pi, a new icon should appear in the taskbar that allows to start and quit Jamulus and JACK in a user friendly way.

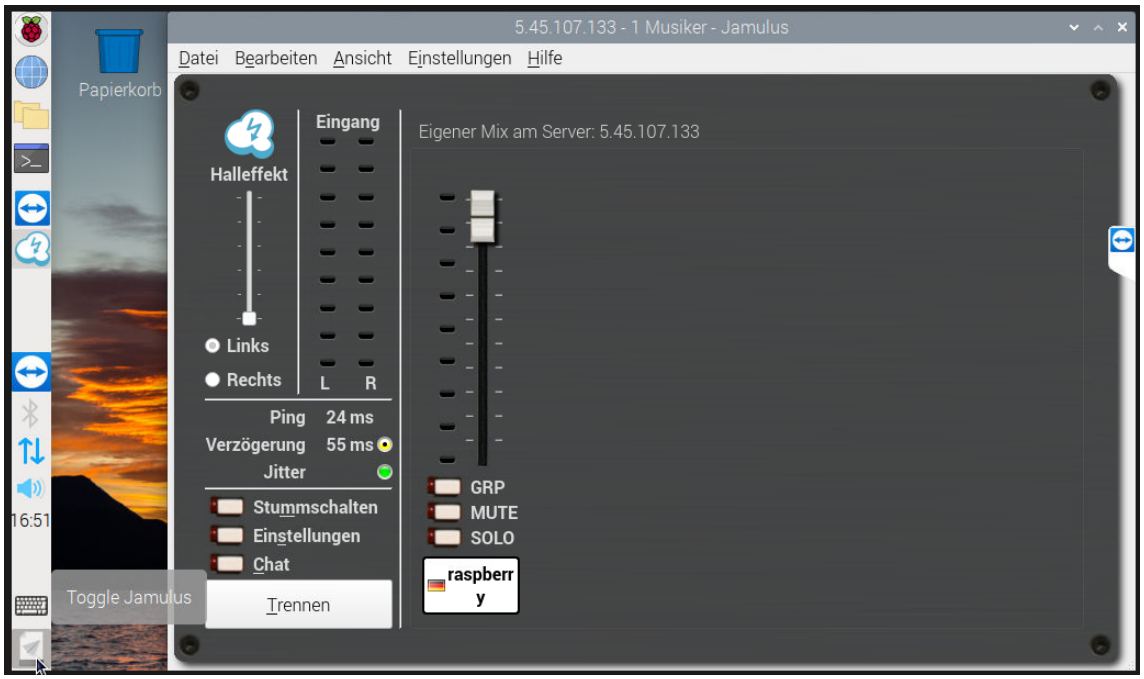


Figure A.1: Icon to start or quit Jamulus & JACK

Circuit for measuring transmission errors and latency
We encourage everyone with basic soldering skills to recreate our test device. The original schematic and PCB layout files (KiCad), as well as the Arduino code, are available on GitHub[1].

However, the original schematic and layout contain several errors. An improved schematic (with English comments) can be found on the next page. A possible way to contribute to this project would be a redesign of the PCB layout, based on this updated schematic (Rev. 2.2).

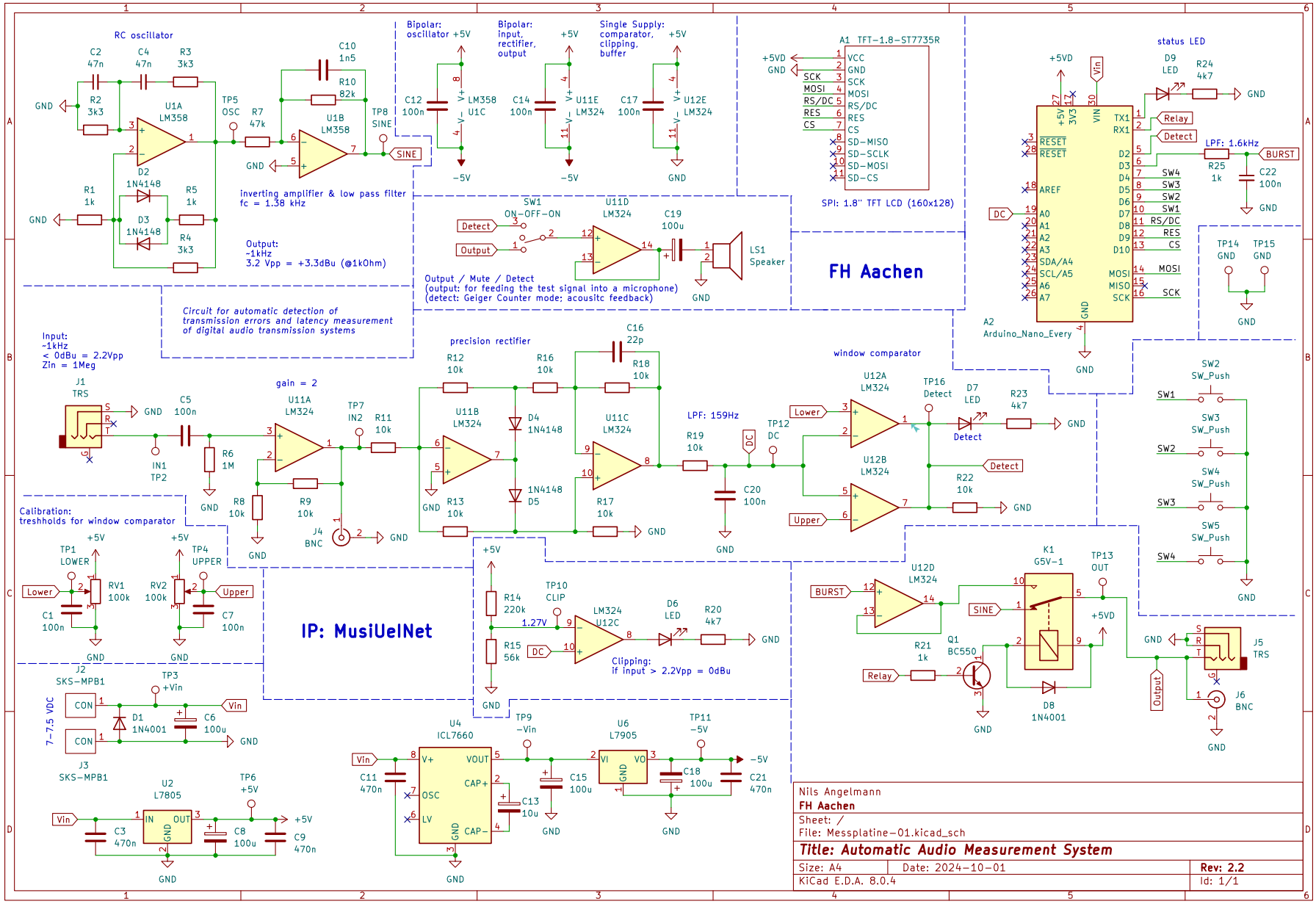
When using the original PCB layout (or Gerber files), the following changes need to be made:

- **Supply voltage**: 7 to 7.5 VDC rather than 7 to 12 VDC
- **Switch SW1**: Annotation flipped - should be output/mute/detect (left to right)
- **Polarity of electrolytic capacitors**: No visible marking - use GND plane as reference
- **Test points**: Only numbers - crossreference schematic for functions
- **Display footprint**: Missing an unused pin - ignore
- **DIP footprints**: Too wide - bend legs slightly appart, do not use sockets
- **U10 (op amp)**: V+ and V- swapped - bend legs up and connect using wire
- **D5**: wrong polarity - solder in with opposing polarity
- **Clipping comparator**: inputs swapped - turn LED around and connect anode to +5V
- **Display SPI**: MOSI and MISO swapped - bridge both Arduino pins (MISO is not needed)
- **LED brightness**: depending on the used LEDs adjust their current-limiting resistors

While most generic parts (resistors and capacitors) are cheaper to buy in bulk and the model is not important, it can be more challenging to find some specific parts:

- **Display**: https://www.reichelt.de/entwicklerboards-display-led-1-8-128x160-st7735r-debo-tft-1-8-p192139.html
- **TRS jack**: https://www.reichelt.de/klinkeneinbaubuchse-6-35-mm-stereo-neutrik-nr-j6hf-p116984.html
- **Power connector (red)**: https://www.reichelt.de/miniatur-pruefbuchse-2-mm-rot-mpb-1-rt-p12710.html
- **Power connector (black)**: https://www.reichelt.de/miniatur-pruefbuchse-2-mm-schwarz-mpb-1-sw-p12711.html
- **BNC connector**: https://www.reichelt.de/miniatur-pruefbuchse-2-mm-schwarz-mpb-1-sw-p12711.html
- **Relay (SPDT)**: https://www.reichelt.de/signalrelais-thd-5-vdc-1-a-1-wechsler-g5v-1-5dc-p243067.html
- **Loudspeaker**: https://www.reichelt.de/kleinlautsprecher-lsm-50m-k-f-0-5w-50ohm-lsm-50m-k-f-p145865.html
- **Switch (on-off-on)**: https://www.reichelt.de/kippschalter-6-a-125-vac-1x-ein-aus-ein-ms-167-p13141.html

[1] https://github.com/StephanBorucki/MusikUeberInternet/


RC oscillator
Bipolar: oscillator +5V
Bipolar: input, rectifier, output
Single Supply: comparator, clipping, buffer
inverting amplifier & low pass filter
fc = 1.38 kHz
Output:
~1kHz
3.2 Vpp = +3.3dBu (@1kOhm)
A1 TFT-1.8-ST7735R
SPI: 1.8" TFT LCD (160x128)
status LED
A2
Arduino_Nano_Every
SW1
ON-OFF-ON
Output / Mute / Detect
(output: for feeding the test signal into a microphone)
(detect: Geiger Counter mode; acousitc feedback)
LS1
Speaker
FH Aachen
Circuit for automatic detection of transmission errors and latency measurement of digital audio transmission systems
Input:
~1kHz
< 0dBu = 2.2Vpp
Zin = 1Meg
precision rectifier
gain = 2
LPF: 159Hz
window comparator
Calibration:
treshholds for window comparator
IP: MusiUelNet
Clipping:
if input > 2.2Vpp = 0dBu
7-7.5 VDC
Nils Angelmann
FH Aachen
Sheet: /
File: Messplatine-01.kicad_sch
Title: Automatic Audio Measurement System
Size: A4
Date: 2024-10-01
Rev: 2.2
KiCad E.D.A. 8.0.4
Id: 1/1

IV