How to set up a DSP box

based on

Raspberry Pi 4 and CamillDSP

v.1.0 2023-07-11

stemag 2023

1. Foreword

The purpose of this document is to show how to build an inexpensive Digital Signal Processor based on *Raspberry Pi 4* hardware and *CamillaDSP* software.
USB ports are used for input and output.
This little guide is just an attempt to put together all the information found on the subject mainly on *github.com*, *audiosciencereview.com* and *diyaudio.com*.
All operations are described as numbered sequences of commands to be given on the command line of the Raspberry OS shell. Such commands are written in bold italics and they can be copied and pasted directly in the RPi4 shell.
Keywords are written in italics.
The *vi* editor is indicated for editing files, but you can use any editor available on your Rpi4, e.g. *nano*.

To set up the DSP box you will need:

- A Raspberry Pi 4
- A 5V 3A power supply
- An internet connection
- A knowledge of the linux operating system

2. Disclaimer

What is described hereafter is the result of my personal experimentation. All the operations have been tested on my own system and proved to be fully working. However, please follow this guide at your own risk.

3. Acknowledgements

The sections concerning the installation of CamillaDSP and its GUI are just a synthesis of what is provided on the *github* pages of the author of this amazing software, Henrik Henquist.

https://github.com/HEnquist

Henrik's pages provide lots of information about the architecture, the configuration and the use of CamillaDSP and CamillaGUI.

The section concerning the set up of the capturing device was largely inspired by the guide published on diyaudio.com by member Daihedz.

https://www.diyaudio.com/community/threads/linux-usb-audio-gadget-rpi4-otg.342070/post-7379163

The section concerning handling of sample rate changes shows a use case of the great tool *gaudio_ctl* presented on *github.com* by member pavhofman.

https://github.com/pavhofman/gaudio_ctl

That section was much helped by the contributions of members *DeLub*, *audiofun* and *phofman* on *audiosciencereview.com*.

https://www.audiosciencereview.com/forum/index.php?threads/using-a-raspberry-pi-as-equaliser-in-between-an-usb-source-ipad-and-usb-dac.25414/

phofman also started and greatly contributed to a very informative thread on diyaudio.com

https://www.diyaudio.com/community/threads/linux-usb-audio-gadget-rpi4-otg.342070/

Many thanks to all the contributors of the above-mentioned threads.

Apologies for any other contributions I have forgot to mention.

4. Setting up Raspberry Pi OS

The latest version of Raspberry Pi OS Lite (64-bit) must be flashed on a microSD. This task can be conveniently accomplished by using the *Raspberry Pi Imager* tool.

https://www.raspberrypi.com/news/raspberry-pi-imager-imaging-utility/

In the Imager, before flashing, configure SSH access and WiFi access.

1. Once your Rpi4 finished booting. open a SSH session and update the operating system:

sudo apt update
sudo apt full-upgrade
sudo reboot

2. Enlarge the file system at maximum and complete the system configuration (e.g. with the local settings) using the *raspi-config* tool.

sudo raspi-config

5. Setting up CamillaDSP

1. Download the latest *CamillaDSP* distribution for Linux on Armv8 64-bit

cd
mkdir camilladsp
cd camilladsp
wget https://github.com/HEnquist/camilladsp/releases/download/v1.0.3-rebuild/camilladsp-linux-aarch64.tar.gz
tar -xvf camilladsp-linux-aarch64.tar.gz

NOTE: the folder of the latest version may differ

2. Create directories for IIR filter configurations and FIR filter coefficients

mkdir configs
mkdir coeffs

3. Create a valid configuration file (e.g. CDSP_Config.yml) under /home/pi/camilladsp/configs.

```
vi /home/pi/camilladsp/configs/CDSP_Config.yml
```

Refer to:
https://github.com/HEnquist/camilladsp
https://github.com/HEnquist/camilladsp/blob/master/stepbystep.md

4. Create a service to start *CamillaDSP* when the system starts up.

```
sudo vi /usr/lib/systemd/system/camilladsp.service
```

Add the following lines to this file:

```
[Unit]
Description=CamillaDSP Daemon
After=multi-user.target syslog.target
StartLimitIntervalSec=10
StartLimitBurst=10

[Service]
Type=simple
ExecStart=camilladsp --logfile /home/pi/camilladsp/camilladsp.log --loglevel error --
port 1234 --wait
Restart=always
RestartSec=1
StandardOutput=journal
StandardError=journal
SyslogIdentifier=camilladsp
User=root
Group=root
CPUSchedulingPolicy=fifo
CPUSchedulingPriority=10

[Install]
WantedBy=usb-gadget.target multi-user.target
```

5. Reload the *systemd* configuration, start the service by hand and check its status

```
sudo systemctl daemon-reload
sudo systemctl start camilladsp
sudo systemctl status camilladsp
```

6. Check that the camilladsp process is started. If all is well, enable the service and reboot:

```
ps -ef | grep camilladsp
sudo systemctl enable camilladsp.service
sudo reboot
```

After rebooting, check that CamillaDSP works as expected.

It may happen that if the DAC is not available, e.g. when switching the DAC input from USB to optical SP/DIF, *CamillaDSP* hangs in the INACTIVE state. In this case we must restore a valid configuration. To this end we can create a *udev* rule that triggers a script doing this job each time the USB DAC device is added to the system.
In the rule file we must specify the parameters *ID_VENDOR_ID* and *ID_MODEL_ID* of the DAC, which can be obtained from the *following* command:
usb-devices

"*Vendor*"corresponds to the *ID_VENDOR_ID* parameter.
"*ProdID*" corresponds to the *ID_MODEL_ID* parameter.

The code for the python script, named *restore_config.py,* is given in appendix to this document. That script assumes that CamillaDSP is started in "wait" mode.

7. Create a rules file under */etc/udev/rules.d*

sudo vi /etc/udev/rules.d/85-DAC.rules

Add the following line to this file, replacing XXXX and YYYY with the actual vendor id and model id of your DAC:

SUBSYSTEM=="usb", ACTION=="add", ENV{ID_VENDOR_ID}=="XXXX", ENV{ID_MODEL_ID}=="YYYY", RUN+="/bin/su pi -c '/usr/bin/python3 /home/pi/restore_config.py'"

8. Force reloading of *udev* rules and test if the added rule works as expected.

sudo udevadm control --reload

To install a CamillaDSP update just do the following:
rm ~/camilladsp/camilladsp-linux-aarch64.tar.gz
wget https://github.com/Henquist/camilladsp/releases/download/vX.Y.Z/camilladsp-linux-aarch64.tar.gz -P ~/camilladsp/
tar -xvf ~/camilladsp/camilladsp-linux-aarch64.tar.gz -C ~/camilladsp/
sudo systemctl restart camilladsp

9. Setting up CamillaGUI

1. Install dependencies

sudo apt update
sudo apt upgrade
sudo apt install python3-websocket python3-aiohttp python3-jsonschema python3-pip

2. Install the python libraries for *CamillaDSP*

pip install git+https://github.com/HEnquist/pycamilladsp.git@v1.0.0
pip install git+https://github.com/HEnquist/pycamilladsp-plot.git@v1.0.2

3. Install the GUI server

cd
mkdir camillagui
cd camillagui

```
wget https://github.com/HEnquist/camillagui-backend/releases/download/v1.0.1/
camillagui.zip
unzip camillagui.zip
```

4. Edit the CamillGUI configuration file (see: https://github.com/Henquist/camillagui-backend). In particular, directories are to be assigned appropriately. In addition, the *update_config_symlink* parameter must be set to true. This way, the current configuration will always be updated and you will find it again on reboot.

```
vi config/camillagui.yml
```

5. Create a service to start *CamillaGUI* when the system starts up.

```
sudo vi /usr/lib/systemd/system/camillagui.service
```

Add the following lines to this file:

```
[Unit]
Description=CamillaDSP Backend and GUI
After=multi-user.target

[Service]
Type=idle
ExecStart=/usr/bin/python3 /home/pi/camillagui/main.py
Restart=always
RestartSec=1
StandardOutput=journal
StandardError=journal
SyslogIdentifier=camillagui
CPUSchedulingPolicy=fifo
CPUSchedulingPriority=10
User=pi
Group=pi

[Install]
WantedBy=multi-user.target graphical.target
```

6. Reload the *systemd* configuration, start the service by hand and check its status

```
sudo systemctl daemon-reload
sudo systemctl start camillagui
sudo systemctl status camillagui
```

7. Check that the CamillaGUI is available and correctly running by pointing your browser at the following URL:
http://your-RPi4-IP-address:5000/

8. If all is well, enable the service and reboot:

```
sudo systemctl enable camillagui
sudo reboot
```

After rebooting, check that CamillaGUI works as expected.

To install an update to GamillaGUI just do the following:

```
rm -r /home/pi/camillagui/camillagui /home/pi/camillagui/camillagui.zip
```

```
wget https://github.com/HEnquist/camillagui-backend/releases/download/vX.X.X/camillagui.zip -P /home/pi/camillagui/
unzip /home/pi/camillagui/camillagui.zip -d /home/pi/camillagui
sudo service camillagui restart
```

9. Setting up the capture device

To capture the incoming digital audio signal we need a USB port working in “Device” mode. The four USB-A ports of the RPi4 can only work in “Host” mode. Instead, the USB-C port used to power the RPi4 can be configured to also work in 'Device' mode, which is what we are going to do.

We need a splitter to connect the RPi4 USB-C port to both the power supply and the digital audio source. To avoid electrical issues, the connector on the power supply side shall carry only DC voltage, whilst the one on the signal source side shall carry only data. I have successfully used this one:

https://it.aliexpress.com/item/1005003793429781.html?spm=a2g0o.store_pc_groupList.8148356.3.5d751d8eiQNhCY&pdp_npi=3%40dis%21EUR%21%E2%82%AC%205%2C13%21%E2%82%AC%205%2C13%21%21%21%21%21%4021038edf16890114755766486ed7f2%2112000027184842775%21sh%21IT%21875552785

1. Edit the boot configuration file to load gadget driver module at start-up

```
sudo vi /boot/config.txt
```

   Append the following line to this file:

```
dtoverlay=dwc2,dr_mode=peripheral
```

2. Reboot the RPi4

```
sudo reboot
```

3. Check that the *dwc2* module has been loaded

```
lsmod | grep dwc2
```

   The lines displayed should look like this:

```
dwc2 192512 0
roles 20480 1 dwc2
```

4. Create the gadget initialisation script. This script must be launched (automatically) when the RPi4 starts up. The code for this script, named *gadget_init.sh*, is given in appendix to this document.

```
cd
vi gadget_init.sh
```

5. Make the script executable and launch it.

```
chmod a+x gadget_init.sh
./gadget_init.sh
```

6. Check that gadget mode is active and that a new sound device is present.

```
cat /proc/asound/cards
```

The following soundcard should be listed:

```
4 [UAC2Gadget ]: UAC2_Gadget – UAC2_Gadget UAC2_Gadget 0
```

(the device and subdevice numbers may differ)

7. Check that the new device is available for 'capture' (UAC2 function):

```
arecord -l
```

The following lines should be displayed:

```
**** List of CAPTURE Hardware Devices ****
card 4: UAC2Gadget [UAC2_Gadget], device 0: UAC2 PCM [UAC2 PCM].
  Subdevices: 1/1
  Subdevice #0: subdevice #0
```

(the device and subdevice numbers may differ)

NOTE: Playback from USB Gadget to Host is disabled, so it will not appear in the list of playback devices shown by "aplay -l".

Further information for the UAC1 and UAC2 test can be found at:
https://www.kernel.org/doc/html/v6.1/usb/gadget-testing.html

8. Create a service for launching the gadget initialisation script at system start-up

```
sudo vi /usr/lib/systemd/system/gadget_init.service
```

Add the following lines to this file:

```
[Unit]
Description=Gadget Initialisation
After=multi-user.target

[Service]
ExecStart=/home/pi/gadget_init.sh
User=root
Group=root

[Install]
WantedBy=multi-user.target
```

10. Reload the *systemd* configuration, start the service manually and check that it has started.

```
sudo systemctl daemon-reload
sudo systemctl start gadget_init
sudo systemctl status gadget_init
```

9. If the service is listed with the status 'inactive' and exit code 0/SUCCESS, enable the service and reboot:

```
sudo systemctl enable gadget_init
sudo reboot
```

10. After rebooting, check that a gadget device is available.

11. Handling sample rate changes

This section describes how to install and how to use the *gaudio_ctl* tool.
This tool will trigger a user command with every change in the sample rate of the signal the gadget device is capturing. The current sample rate is passed as a parameter to the user command. In our case the command will launch a python script that adjusts the *samplerate* parameter in the configuration of CamillaDSP The *chunksize* parameter is adjusted as well as a function of the sample rate.
This method assumes that CamillaDSP is launched in "wait" mode.
For IIR filters only one configuration file is required for CamillaDSP to work (no need to have a configuration file for each sample rate).

1. Install the *rust* development environment (select option '1' and proceed with the installation):

```
curl --proto '=https' --tlsv1.2 -sSf https://sh.rustup.rs | sh
```

2. Configure the shell

```
source "$HOME/.cargo/env"
```

NOTE: To uninstall *rust*:

```
rustup self uninstall
```

3. Install *git* and the *alsa* development environment

```
sudo apt install git
sudo apt install libasound2-dev
```

4. Download and build *gaudio_ctl,* then copy the executable file to */usr/local/bin*

```
git clone https://github.com/pavhofman/gaudio_ctl
cd gaudio_ctl
cargo build --release
sudo cp ./target/release/gaudio_ctl /usr/local/bin
```

Each time the sample rate changes, *gaudio_ctl* will launch a python script that sends *CamillaDSP* the correct *samplerate* and *chunksize* parameters for the incoming signal. The code for such a script, named *set_samplerate.py*, is given in appendix to this document.

5. A daemon starting *gaudio_ctl* must be created as follows.

```
sudo vi /usr/lib/systemd/system/set_samplerate.service
```

Add the following lines to this file:

```
[Unit]
Description=Sample Rate Changer Daemon
After=multi-user.target

[Service]
ExecStart=gaudio_ctl --gadget-name "UAC2Gadget" --ccmd "/usr/bin/python3 /home/pi/set_samplerate.py {R}"
Restart=always
RestartSec=1
StandardOutput=journal
```

```
StandardError=journal
SyslogIdentifier=set_samplerate
CPUSchedulingPolicy=fifo
CPUSchedulingPriority=10
User=pi
Group=pi

[Install]
WantedBy=usb-gadget.target multi-user.target
```

6. Reload the *systemd* configuration, start the service manually and check its status.

```
sudo systemctl daemon-reload
sudo systemctl start set_samplerate
sudo systemctl status set_samplerate
```

If the service is active, enable the service and reboot:

```
sudo systemctl enable set_samplerate
sudo reboot
```

7. After rebooting, play tracks at different sample rates and check with *CamillGUI* that *CamillaDSP* is active and is capturing at the correct sample rate.

Appendix A. CamillaDSP config restore script

```
#!/usr/bin/python3
# restore_config.py
#
# RESTORING A VALID CONFIGURATION FOR CAMILLADSP
# This script must be launched whenever camilladsp hangs waiting for
# a valid config. This may happen if the DAC is disconnected,
# or if at the DAC the selected input is other than USB.
# When the DAC is reconnected, or the USB input is again selected,
# a udev event of the type "add" is triggered. By defining a udev rule that
# launches this script, a valid configuration is loaded into camilladsp when such
# event is triggered, so that camilladsp can run again.
# The previous valid configuration is picked up. If the previous configuration
# is not valid, a standard configuration is loaded from file.

import sys
import camilladsp

c = camilladsp.CamillaConnection("127.0.0.1", 1234)

msg = ""

try:
    c.connect()

    config = c.get_config()

    if config is not None:
        print("Current CamillaDSP Config is already valid.")
        sys.exit(0)

    config = c.get_previous_config()

    if config is not None:
        c.set_config(config)
        print("Previous CamillaDSP Config restored.")
        sys.exit(0)

    config = c.read_config_file("/home/pi/camilladsp/configs/CDSP_Config.yml")

    if config is not None:
        c.set_config(config)
        print("File CamillaDSP Config restored.")
        sys.exit(0)

    stop_reason = c.get_stop_reason()

    print("Config not restorable")

    sys.exit(stop_reason)


except ConnectionRefusedError as e:
    msg = "Can't connect to CamillaDSP, is it running? Error:" + str(e)
    retry = True
except camilladsp.CamillaError as e:
    msg = "CamillaDSP replied with error:" + str(e)
    retry = True
except IOError as e:
    msg = "Websocket is not connected:" + str(e)
    retry = True
finally:
    print(msg)
```

Appendix B. Gadget initialization script

```
#!/bin/sh
###################################
# gadget_init.sh
# Raspberry Pi OS on Raspberry Pi 4B
###################################
# This script is taken from the guide published by member Daihedz on diyadio.com
# Please adapt some parameters and directories according to your needs

CONFIGFS_ROOT=/sys/kernel/config
GDG_DIRNAME="audio-basic"

# Basics - better not to be changed, because of the standard nature of the values set
```

```
BCD_DEVICE=0x0100 # v.1.0.0
BCD_USB=0x0200 # USB2
ID_VENDOR=0x1d6b # Linux Foundation
ID_PRODUCT=0x0104 # 0x0104 for Multi Functional Gadget / 0x0101 for Audio Gadget

# Strings - likely/optionally to be adapted (except the first one)
STRG_LANGUAGE=0x409 # US English
STRG_MANUFACTURER="YourManufacturer" # adapt to your needs
STRG_PRODUCT="UAC2_Gadget"
STRG_SERIALNUMBER="000001" # adapt to your needs

# Configuration(s) - likely/optionally to be adapted
CONFIGURATION_CNF_1="YourConfigurationName" # adapt to your needs

# Functions – Adapt to your needs. The following is for a typical stereo audio device
AUDIO_CHANNEL_MASK_CAPTURE=3 # 1=Left 2=Right 3=Stereo 0=disables the device
AUDIO_CHANNEL_MASK_PLAYBACK=0 # Playback is disabled on USB-C port
AUDIO_SAMPLE_RATES_CAPTURE=44100,48000,88200,96000,176400,192000,352800,384000
AUDIO_SAMPLE_RATES_PLAYBACK=44100,48000,88200,96000,176400,192000,352800,384000
AUDIO_SAMPLE_SIZE_CAPTURE=4 # 1 for S8LE / 2 for S16LE / 3 for S24LE / 4 for S32LE
AUDIO_SAMPLE_SIZE_PLAYBACK=4

### Load the required kernel modules (and ev. overlays) ###

# libcomposite
modprobe libcomposite

### create the gadget ###

# create the gadget directory and change into it
cd "${CONFIGFS_ROOT}"/usb_gadget
mkdir -p $GDG_DIRNAME
cd $GDG_DIRNAME

# basics
echo $BCD_DEVICE > bcdDevice
echo $BCD_USB > bcdUSB
echo $ID_VENDOR > idVendor
echo $ID_PRODUCT > idProduct

# strings
mkdir -p strings/$STRG_LANGUAGE
echo $STRG_SERIALNUMBER > strings/$STRG_LANGUAGE/serialnumber
echo $STRG_MANUFACTURER > strings/$STRG_LANGUAGE/manufacturer
echo $STRG_PRODUCT > strings/$STRG_LANGUAGE/product

# configuration(s)
mkdir configs/c.1 # index mandatory for every configuration
mkdir -p configs/c.1/strings/$STRG_LANGUAGE
echo $CONFIGURATION_CNF_1 > configs/c.1/strings/$STRG_LANGUAGE/configuration

# functions
mkdir -p functions/uac2.usb0
echo $AUDIO_CHANNEL_MASK_CAPTURE > functions/uac2.usb0/c_chmask
echo $AUDIO_SAMPLE_RATES_CAPTURE > functions/uac2.usb0/c_srate
echo $AUDIO_SAMPLE_SIZE_CAPTURE > functions/uac2.usb0/c_ssize
echo $AUDIO_CHANNEL_MASK_PLAYBACK > functions/uac2.usb0/p_chmask
echo $AUDIO_SAMPLE_RATES_PLAYBACK > functions/uac2.usb0/p_srate
echo $AUDIO_SAMPLE_SIZE_PLAYBACK > functions/uac2.usb0/p_ssize

# associate functions to configurations
ln -s functions/uac2.usb0 configs/c.1/

# enable the gadget
ls /sys/class/udc > UDC
```

Appendix C.   Sample rate changer script

```
#!/usr/bin/python3
# set_samplerate.py
#
# Updates the sample rate and chunksize parameters in the configuration of CamillaDSP.
# This script must be launched at any sample rate change
# (a gaudio_ctl daemon will do the trick).
# The value passed in the command line  is used to update the sample rate in the current
configuration.
# Then the configuration is updated with the chunksize parameter calculated as a function
of sample rate.
# Finally, the updated configuration is uploaded to CamillaDSP via a websocket.
# If camillaDSP has not a current valid configuration, the configuratoin is loaded from the
previous
# is loaded configuration; if again the previous configurationn is non valid, the configuration
# is loaded from file.
# The sample rate is set to match the sample rate of the signal being captured.
# Note 1: camilladsp must be launched in wait mode.
# Note 2: the parameter samplerate in the standard configuration is not relevant,
#         as it is changed anyway soon after the file is loaded.
#
# This script derives from the work of @audiofun on ASR forum.

import sys
import camilladsp

c = camilladsp.CamillaConnection("127.0.0.1", 1234)

msg = ""

try:
    c.connect()
    rate = int(sys.argv[1])

    config = c.get_config()

    if config is None:
        config = c.get_previous_config()

        if config is None:
            config = c.read_config_file("/home/pi/camilladsp/configs/CDSP_Config.yml")

    if config is not None:
        config['devices']['samplerate'] = rate

        if rate <= 48000:
            chunksize = 1024
        elif rate <= 96000:
            chunksize = 2048
        else:
            chunksize = 4096

        config['devices']['chunksize'] = chunksize

        c.set_config(config)

        msg = "New settings: samplerate={} chunksize={}".format(rate, chunksize)
    else:
        msg = "No valid config: unable to set samplerate”

except ConnectionRefusedError as e:
    msg = "Can't connect to CamillaDSP, is it running? Error:" + str(e)
    retry = True
except camilladsp.CamillaError as e:
    msg = "CamillaDSP replied with error:" + str(e)
    retry = True
except IOError as e:
    msg = "Websocket is not connected:" + str(e)
    retry = True
finally:
    print(msg)
```