Set up a Linux Audio DSP UAC2 Rpi3A+ Gadget

Version:
23/06/17

Topic:
It will take approx. 30 minutes to set up a basic UAC2 gadget following part I of this HowTo. Be aware that despite starting from the "lite" version of the Raspberry Pi OS, the resulting system will include many standard OS features not really needed for an audio gadet setup. This standard setup will perform flawlessly and will completely satisfy the needs of most users.

This said, there is an optional, more extensive second section of this HowTo including four parts. Part II will deal with some tweaking options and modifications of the OS. Like getting rid of the horribly bloated WPA-supplicant (in it's Debian implementation). Or you e.g. may want to reduce the wear of the SD-card, e.g. by disabling journaling. Take care that every modification should come along with a real functional benefit. Successful modifying means gaining something at the cost of giving up some other functionality. Part III deals with CPU tweaking options potentially useful for audio. Part IV follows with a description of a quite complex setup including all goodies described before in Parts II and III. Part V is a collection of listings and other useful stuff.

System Actuality:
Raspberry Pi OS imager 1.7.5 / Debian Bullseye V.6.1 / CamillaDSP V.1.0.3

Ingredients:
Rpi3A+, Raspberry Pi OS Lite (64Bit), CamillaDSP.

Rpi3 vs. Rpi4:
In this above combination along with a Rpi3A+, the resulting UAC2 Gadget can easily be operated by connecting it to a single, standard USB-A port of a laptop. No need for a second connection providing extra power. The overall processing resources of a Rpi3A+ may at first look seem a bit moderate compared with other platforms, but will comfortably allow even very advanced DSP: And in most cases, there will be still enough resources and reserves to allow CPU clock frequency throttling while processing in steady-state. So there really is no need to resort to a more power hungry and hot Rpi4. You might instead loose the single-supply feature advantage in doing so.

About the formatting of this HowTo:
```
# Code is formatted in this liberation mono typeset
```
Text and code snippets formatted *LikeThis* is either *UserDefined*, *SystemSpecific* or *ChangingOverTime*, like *HomeDirectoryTrees* or *ReleaseVersions*. You will have to adapt it to your system and needs.

Real-world solution example:
In section V at the end of this HowTo there is a complete description of a functional, stripped-down UAC2 gadget. While processing audio as a headless system and still potentially responding to a ssh, it shows a very low kernel module and running services count.

For further improvement's sake, please … :
Some suggestions in this HowTo may not be really elegant, some maybe stupid, some redundant and some better solutions may exist. Feedback is welcomed, and probably best posted into p.hofman's the very informative diyaudio rpi4 otg thread for further general discussion: **https://www.diyaudio.com/community/threads/linux-usb-audio-gadget-rpi4-otg.342070/**

Disclaimer:
DSP might destroy your audio equipment and damage your auditory system by accidental excessive loudness if not correctly set up. Proceed at you own risk.

La Direttissima

Barebone UAC2 gadget:

```
$ sudo vi /boot/config.txt
```
→ add this statement: `dtoverlay=dwc2,dr_mode=peripheral`

```
$ touch /home/pi/gadget.sh
$ chmod u+x /home/pi/gadget.sh
```
→ copy and paste scripting code from part IV/L1
```
$ sudo /home/pi/gadget.sh
```

Autostart the UAC2 gadget mode at boot:

```
$ sudo touch /lib/systemd/system/gadget.service
```
→ copy and paste scripting code from part IV/L3a
```
$ sudo systemctl enable /lib/systemd/system/gadget.service
```

Same goes for autostart CamillaDSP mode at boot:

```
$ sudo touch /lib/systemd/system/gdsp.service
```
→ copy and paste scripting code from part IV/L3b
```
$ sudo systemctl enable /lib/systemd/system/gdsp.service
```

```
$ sudo reboot now
```

Content

Z. Useful links

Part I/V: The basic DSP UAC2 gadget setup

Content:
A. Install and update the basic OS
B. Setup the UAC2 mode
C. CamillaDSP
D. Make things work together

A: The basic Raspberry OS

As a very first step, install Raspberry Pi OS Lite (64-bit) onto a SD card, best by using the current available version of the Raspberry Pi Imager. Enable WiFi and SSH for the management of the Pi as a headless host. For this HowTo, a standard user name "pi" is assumed. Then startup the Raspberry Pi, log into it and run the update process.

Download from the Raspberry Pi OS website:

```
https://downloads.raspberrypi.org/imager/imager_latest.exe
```

apt
run this command to update the OS

```
$ sudo apt update
$ sudo apt upgrade
$ sudo reboot now
```

B. Setup the audio UAC2 / OTG

Maybe an appealing in-depth lecture to this topic first:

https://www.collabora.com/news-and-blog/blog/2019/02/18/modern-usb-gadget-on-linux-and-how-to-integrate-it-with-systemd-part-1/

https://www.collabora.com/news-and-blog/blog/2019/03/27/modern-usb-gadget-on-linux-and-how-to-integrate-it-with-systemd-part-2/

B.1.: Set up the UDC and the DWC2 kernel module

/boot/config.txt
Edit the RaspberryOS configuration file

The config.txt must contain this line:
`dtoverlay=dwc2,dr_mode=peripheral`

For a minimal headless approach, the config.txt might contain only these 3 lines:
```
arm_64bit=1
dtoverlay=dwc2,dr_mode=peripheral
gpu_mem=16
```

To preserve Rpi onboard audio devices functional instead, eventually keep any or both of these two lines:
`dtoverlay=vc4-kms-v3d` (enables the *vc4hdmi* device)
`dtparam=audio=on` (enables the *Headphones* device)

Checks:

```
$ lsmod | grep dwc2
```
This must now show the dwc2 and the roles module

```
$ ls -R /sys/class/udc/*
```

Must now show the UDC (USB device controller) with it's specific directory tree

```
$ cat /proc/asound/cards
```
The Headphones and the vc4hdmi audio devices might figure or might not figure
At this point, the gadget itself is not configured yet. Therefore it does not appear as an alsa soundcard device.

B.2.: Create the audio gadget functionality

The gadget is set up by using the configfs interface. This approach is up-to-date and versatile. It is a bit more complex to set up than had been the now deprecated and rigid g_audio (pseudo) kernel module approach. It is recommended not to use the g_audio approach anymore.

In this configfs interface approach, basically ...
- the audio gadget will be declared
- the configuration(s) will be declared
- the functions will be declared
- the functions will be associated with their configurations
- the gadget will be enabled

/home/pi/gadget.sh
create an executable shell script file for all the appropriate code.

To keep this section slim, the ready-to-paste script code for this file is found in part IV/L1 of this HowTo. Running this script file will initialize the gadget. You may adapt some parameters for your own settings and needs.

As for all shell files, in order to run this file as a command, it must be declared as executable

```
$ chmod u+x /home/pi/gadget.sh
```

Then run this file as root to create the UAC2Gadget

```
$ sudo /home/pi/gadget.sh
```

Checks:

```
$ cat /proc/asound/cards
```
This must show the UAC2Gadget after the script has been run

```
$ aplay -l
```
Show the resulting playback audio cards

```
$ arecord -l
```
Show the resulting caption audio cards

```
$ ls -R /sys/kernel/config/usb_gadget/*
```
Must show the active configuration of the usb_gadget

Reference - must read:
`https://www.kernel.org/doc/html/latest/usb/gadget_configfs.html`

Reference - for the complete set of functions see:
`https://www.kernel.org/doc/Documentation/ABI/testing/configfs-usb-gadget-uac2`

Reference - further reading:
`https://www.kernel.org/doc/html/`*`v6.1`*`/usb/gadget-testing.html`
v6.1 refers to the actual linux kernel version running on the gadget

C: CamillaDSP (CDSP) forever …

From now on in the further text notes, “CamillaDSP” will be abbreviated as “CDSP”

C.1.: Install CamillaDSP on your system

wget
run this command to download the CDSP tar

```
$ wget https://github.com/Henquist/camilladsp/releases/download/v1.0.3/camilladsp-linux-aarch64.tar.gz
```
v1.0.3 refers to the CDSP version to download

apt
run this command to install bsdtar

```
$ sudo apt install libarchive-tools
```

bsdtar
run this command to extract and install CDSP

```
$ bsdtar -xf camilladsp-linux-aarch64.tar.gz
$ sudo mv camilladsp /usr/local/bin
```

mkdir
run this command to set up a basic sub-directory tree for the CDSP auxiliary files. This is optional.

```
$ cd /home/pi
$ mkdir _camilladsp
$ mkdir _camilladsp/coeffs
$ mkdir _camilladsp/configs
```

Check:

```
$ camilladsp
```

C.2.: Configure CDSP to capture from the gadget's USB_input

/home/pi/*_camilladsp/configs/CDSP_Config*.yml
edit the CDSP configuration file

Make the values of the CDSP configuration file match with the gadget functions values of the /home/pi/gadget.sh init script. For e.g. c_srate=44100, c_chmask=3, c_ssize=2 might translate into ...

```
$ vi /home/pi/_camilladsp/configs/CDSP_Config.yml
...
devices
  capture_samplerate: 44100
  ...
  capture:
    channels: 2  # (chmask=3)
    device: hw:CARD=UAC2Gadget,DEV=0
    format: S16_LE # (c_ssize=2)
    type: Alsa
  playback:
    ...
...
```

C.3. First functional system test

At this point, a basic functional system is already set up. Test it for it’s correct, intended function.

```
$ sudo reboot now
$ sudo /home/pi/gadget.sh
$ camilladsp /home/pi/_camilladsp/configs/CDSP_Config.yml
```

If the system does not behave as expected, some debugging might be appropriate before going further.

/home/pi/gdsp.sh
Create this file

GDSP stands for GadgetDSP. For convenience, the audio startup process might be implemented into a shell script to allow for some extras. An example of such (a bit more elaborate file) is described in part IV/L2 of this HowTo.

D. Make things work together and autostart on bootup

D.1. UAC2 gadet mode service

Set up a service to initialize the audio gadet device at boot

/usr/lib/systemd/system/gadget.service
create this new file

```
$ sudo vi /usr/lib/systemd/system/gadget.service
...
[Service]
ExecStart=/home/pi/gadget.sh
Group=root
User=root
...
```

To keep this section slim, the ready-to-paste script code for this file is found in part IV/L3 of this HowTo, named/labeled as gadget.service. You may adapt some parameters for your own settings and needs.

Check:

```
$ sudo systemctl start gadget.service
```
This must have the same effect as starting gadget.sh as root - It must create and configure the UAC2 gadget

D.2. Audio processing service

Set up a service to start the gadget audio processing at boot.

/usr/lib/systemd/system/gdsp.service
create this new file

```
$ sudo vi /usr/lib/systemd/system/gdsp.service
...
[Service]
ExecStart=/home/pi/gdsp.sh
Group=root
User=root
...
```

To keep this section slim, the ready-to-paste script code for this file is found in part V/L3 of this HowTo, named/labeled as gdsp.service. You may adapt some parameters for your own settings and needs.

Make sure that the audio processing does start after the gadget init service has set up the audio gadget: CDSP has to find the gadget's functional audio device. Otherwise, it will crash. Therefore, inside of the /home/pi/gdsp.sh start script there first is a check for the correct functioning of the gadget device before then starting up the audio processing by CDSP. See Part IV/L2 for this kind of solution.

Check:

```
$ sudo systemctl start gdsp.service
```
This must have the same effect as starting gdsp.sh as root - It must start the DSP processing.

D.3. Make start the gadget.service and the gdsp.service at host boot

You may enable these services at startup as one of the very last steps, e.g. after having tested the full functionality of your basic setup. Fist check both your /home/pi/gadget.sh and your /home/pi/gdsp.sh shell scripts by test-starting them inside a root console, and then control the outcome.

Then, if both scripts run flawlessly, manually test-start the gadget.service and the gdsp.service by

```
$ sudo systemctl start gadget.service
$ sudo systemctl start gdsp.service
```

If these two services has proven to be functional, enable them at boot

```
$ sudo systemctl enable foo.service
```

If something goes wrong and for debug purpose, you may disable the boot-time startup behavior by ...

```
$ sudo systemctl disable foo.service
```

Running services can be stopped by ...

```
$ sudo systemctl stop foo.service
```

Part II/V: General (non Audio) OS tweaks and options

Content:

E. WiFi options

E.1. Set up iwd instead of (outdated and bloated) WPA-supplicant

In it's actual Debian implementation, WPA-supplicant depends on an incredible, and probably historically grown amount of dependencies. Occupying 35MB of disk space in total.

Prepare iwd to start after the next boot

```
$ sudo apt install iwd
$ sudo systemctl enable iwd.service
$ sudo systemctl start iwd.service
$ sudo iwctl
  - device list
  - station wlan0 scan
  - station wlan0 get-networks  - your network should figure on the list
  - exit
```

/var/lib/iwd/dhcpcd.*YourNetworkSSID*.psk
create a matching configuration file

```
$ sudo vi /var/lib/iwd/YourNetworkSSID.psk
...
[Security]
Passphrase=YourPassphrase
...
```

Disable WPA-supplicant

```
$ sudo systemctl disable wpa_supplicant
$ sudo reboot now
```

the system will reboot and establish a network connection through iwd

Check:

```
$ systemctl | grep iwd
```

Get rid of the WPA-supplicant installation

```
$ sudo apt purge wpasupplicant
$ sudo apt autoremove
$ sudo rm -rf /etc/wpa_supplicant
```

Comment:
As long as WPA-supplicant is running, iwd will not set up a `/var/lib/iwd/`*`YourNetworkSSID`*`.psk` file.
Therefore, first manually prepare a psk file for iwd to log into your network, then disable WPA-supplicant, then reboot.

E.2. Make the dhcp startup some seconds faster

/etc/dhcpcd.conf
append 'noarp' to this file to make dhcpcd startup faster by some seconds

```
$ sudo vi /etc/dhcpcd.conf
...
noarp
...
```

F. Uninstall useless programs (useless for your specific audio use)

Uninstalled programs do free space and will never be upgraded any longer. So this manages the SD card. Uninstalled programs don’t appear as running services either. Make your choice which programs you will get rid of.

apt
run apt to uninstall some programs

```
$ sudo apt purge foo foo foo foo
$ sudo apt autoremove
```

For a list of candidates for removal see part V/X.1. of this HowTo

You may also replace some programs by leaner versions, such as vim-tiny instead of vim-common.

Comment:
This might be an ongoing process. Every time an update/upgrade process will be performed, unexpected programs may appear in the update list you do not really want to keep.

G. Reduce the wear of the SD Card

Every write process wears the SD Card. Therefore, minimizing writes will lead to a longer life of the SD card.

G.1. Write temp and dynamic files into a RAM disk instead onto the SD card

/etc/fstab
append several lines to this file

```
$ sudo vi /etc/fstab
...
tmpfs  /tmp       tmpfs  noatime,nodev,nosuid,size=5M 0  0
tmpfs  /var/lock  tmpfs  noatime,nodev,nosuid,size=5M  0  0
tmpfs  /var/log   tmpfs  noatime,nodev,nosuid,size=50M  0  0
tmpfs  /var/run   tmpfs  noatime,nodev,nosuid,size=5M  0  0
tmpfs  /var/tmp   tmpfs  noatime,nodev,nosuid,size=5M  0  0
...
```

Comment:
In the same logic, do not set up a swap partition in fstab

G.2. Disable swapping to a file

dphys-swapfile
run this command

```
$ sudo dphys-swapfile swapoff
```

G.3. Minimize journaling

/etc/systemd/journald.conf
edit this file to minimize journaling

```
$ sudo vi /etc/systemd/journald.conf
...
Storage=none
MaxLevelStore=emerg
MaxLevelSyslog=emerg
MaxLevelKMsg=emerg
MaxLevelConsole=emerg
MaxLevelWall=emerg
ReadKMsg=no
Audit=no
...
```

G.4. Disable journaling on the ext4 file system

First insert the SD card onto a suitable linux host and then, on the host's console ...

```
$ sudo tune2fs -O ^has_journal /dev/mmcblk0p2
```

Remark:
-O like Oscar, not -0 like Zero

Check:

```
$ sudo debugfs -R features /dev/mmcblk0p2 | grep has_journal
```

H. The OS - reduce it's overall workload

A fresh install of Raspberry Pi OS lite 64Bit comes along with some 60 kernel modules loaded and some 140 active systemd units while 30 of them running at steady-state conditions. Most of these resources are not needed for the purpose of USB gadget audio DSP. Two options of unloading/stopping these programs are described - a dynamic one, and a static one. The dynamic approach will demand some processing time at every startup, but offers some flexibility and preserves the full functionality of the OS for maintenance. Instead, in the static approach, the modules may be statically prevented from loading at boot time.

In this HowTo, some lists of trash/keep candidates are already set up. These lists may need some individualization on other system. Any list of the trash or keep candidates may be iteratively gained by a trial-and-error method: Have your system run the application, and then one by one unload modules and disable services. If the application crashes, then you disabled a module or service (unit) to keep. Reset and go on trying ...

H.1. Kernel modules - the good, the bad, and the uglies ...

H.1.a. The dynamic method

At boot, all standard modules get loaded. Then, the unwanted/unneeded kernel modules get unloaded within a shell script file before audio processing starts.

/home/pi/gdsp.sh
The dynamic method is implemented within a function named os_kernel_trim()
See part IV/L2 of this HowTo for the details.

Central part is the declaration of list of modules to keep. See part V/X.2.b. of this HowTo for a "keep" list. This list is

declared as variables in the head section of the coding of /home/pi/gdsp.sh. Also all other variables are declared there in the head section.

Then, in the middle part of the script, where the functions are defined, there is the function

```
function os_krnl_trim()
```

This function implements the functionality to unload all loaded modules which do not figure in this list.

H.1.b. The static method

The static approach is to avoid loading the kernel modules at boot time as part of the (static) system configuration.

/etc/modprobe.d/blacklist.conf
create this new file

```
$ sudo vi /etc/modprobe.d/blacklist.conf
...
install foo /bin/false
install foo /bin/false
...
```

See part V/X.2.a. of this HowTo for a "trash" list of **modules to exclude** from being loaded

H.1.c. Rpi onboard sound device related kernel modules

Depending on enabling or disabling the onboard sound hardware, you may include or exclude the following two kernel modules:

Sound jack connector (Headphones):
snd_bcm2835

HDMI sound output:
snd_soc_hdmi_codec

H.2. Systemd units - services, sockets, timers, targets ...

H.2.a. The dynamic method

For the systemd units, there is a good starting point: While the system is functional in steady-state, any running unit is a candidate for the category which is to be kept running:

```
$ systemctl | grep "loaded active running" | cut -d"." -f1
```

Then you may compare this list against all units in "active" state

```
$ systemctl | grep "loaded active" | cut -d"." -f1
```

Your may then stop all "loaded active" units not being in "loaded active running" state. This already does a very good job, without any good-bad-ugly list. If you need further tuning, you may declare units to keep and units to stop.

/home/pi/gdsp.sh
The dynamic method is implemented within a function named os_sysd_trim()
See part IV/L2 of this HowTo for the details.

H.2.b. The static approach

The static approach is to avoid starting the services at boot time as part of the (static) system configuration, by masking the service.

`$ sudo systemctl mask foo.service` (=byebye forever ... unless you unmask them again)

or

```
$ sudo systemctl disable foo.service
```

See part V/X.3. of this HowTo for a list of services to stop from being started

Checks:

```
$ systemctl | grep "loaded active running"
```

```
$ systemctl | grep "loaded active"
```

Part III/V: Audio related tweaks and options

Content:

I: CPU settings

I. CPU settings and allocations

I.1. Set the CPU to a constant frequency

The regular ondemand CPU frequency switching governor may cause xruns during playback. This is because CPU frequency switching may come along with some ms of latency. Therefore, instead allow uncontrolled dynamic switching by the ondemand governor, better resort to the constant frequency performance and powersave governors to precisely toggle between high and low CPU demands.

apt
run this command

```
$ sudo apt install linux-cpupower
```

/home/pi/gdsp.sh
add some scripting

```
#!/bin/sh
...
### Variables declaration part of script
...
# CPU frequency min/max declaration
CPU_FREQ_LO=600000  # values in kHz!
CPU_FREQ_HI=1200000
...
### Functions part of the script
...
function CPU_freq_set() {
  # CPU frequency min/max values
  sudo cpupower frequency-set --min $CPU_FREQ_LO --max $CPU_FREQ_HI
  # Set CPU frequency to constant max value
  sudo cpupower frequency-set --governor performance
  sleep 0.5
  # Set CPU frequency to constant min value
  sudo cpupower frequency-set --governor powersave
}
...
### Main program flow part of the script
...
host_CPU_set &  # Must be run in background as a daemon
sleep 0.1
camilladsp config.yml
...
```

Checks:

```
$ sudo cat /sys/devices/system/cpu/cpufreq/policy0/cpuinfo_cur_freq

$ sudo cat /sys/devices/system/cpu/cpufreq/policy0/scaling_governor
```

Remark:
Starting a program might cause a high CPU load. Therefore for initializing, switch for a short time to a high CPU frequency, and then reduce the frequency for a more economic steady-state. As the ondemand scheduler is intended to do and would do. The above approach does exactly this, but then keeps the steady-state lower frequency to a constant value.

Remark II:
The actual implementation is a slightly different and a bit more elaborate, but follows the same principle. Have a look at the function cpu_freq_set in Part IV/L2 of this HowTo

I.2. Run audio programs (such as CDSP) on specific and optionally isolated CPU(s)

The following examples do isolate CPU 2 and 3 to independently run all threads of specified audio programs on these specific CPUs. Instead, let e.g. CPU 0 handle all IRQs

I.2.a. Assign programs to specific CPUs

/home/pi/gdsp.sh
edit this file, use the taskset command to assign program and CPUs

```
#!/bin/sh
...
sudo taskset -c 2-3 camilladsp configfile ...
...
```

Comment:
This assigns all threads of CDSP to CPUs 2 and 3. The system scheduler may still assign any other program to these CPUs. To have CDSP run as only programs on these CPUs, you will have to isolate these CPUs. See section E.3.b.

Comment II:
See section E.3.c.

I.2.b. CPU isolation

This best works together with the CPU assignment method described in the section E.3.a. above

/boot/cmdline.txt
add some instructions to the existing text line:

```
$ sudo vi /boot/cmdline.txt
...
... irqaffinity=0 nohz=on isolcpus=2-3 nohz_full=2-3 rcu_nocbs=2-3
...
```

Checks:

```
$ sudo cat /sys/devices/system/cpu/isolated

$ ps H -q $(pidof -s camilladsp) -o 'pid,tid,cls,rtprio,comm,pcpu,psr'
```

I.2.c. Pros and Cons

There are pro's, con's and compromises in terms of CPU assignment and isolation, as CPU isolation might not only be beneficial. The regular system scheduler is very effective in distributing all load between all available CPU resources. Some audio programs such as CDSP and also MPD run theirs process as several threads, and by standard all or them being more or less evenly distributed between all CPUs resources. Such minimizing the individual load per CPU, this allows the CPU to be operated at a lower frequency. In practice and as in one of the tested setups, CDSP runs as 3 threads and starts another 3 to 4 child threads. Now, squeezing all these threads onto a single CPU might indeed not be an optimal solution. The CPU frequency will need to be highish, or worst case the total load will even get too high for a

single CPU to handle it. As a workaround and as described in the above setup, two CPUs (2 and 3) might be isolated and assigned to all of the threads of CDSP and it's child threads. This approach still isolates the audio processing while letting the scheduler balance the load between CPU2 and CPU3. Furthermore, in the above example, all IRQ handling is affined to CPU0.

Part IV/V: A real-world system setup

K. The Testbed

The hardware consists of a Raspberry Pi 3A+ with a HifiBerry Digi2-Pro (HW 2.2) board. Data connection and power supply is both performed via one and the same USB_A to USB_A cable attached to the host computer. There is no additional supply through the USB_B connector needed.

The Raspberry Pi OS lite installation is best performed by the Raspberry Pi Imager, SSH and WiFi enabled for a headless system inside of an accessible WiFi zone.

Modifications on standard OS files:

/boot/cmdline.txt
contains the kernel isolation statements described in part III of this setup

```
... irqaffinity=0 nohz=on isolcpus=2-3 nohz_full=2-3 rcu_nocbs=2-3
```

/boot/config.txt
contains nothing but

```
arm_64bit=1
dtoverlay=dwc2,dr_mode=peripheral
gpu_mem=16
```

New and application specific files:

/home/pi/gadget.sh
content see below

/home/pi/gdsp.sh
content see below

/usr/lib/systemd/gadget.service
content see below

/usr/lib/systemd/gdsp.service
content see below

Two directories for the DSP functionality:

home/pi/*_camilladsp/coeffs*
contains the fir filters files

home/pi/*_camilladsp/configs*
contains the CDSP configuration file(s)

L. The Scripts

L.1. /home/pi/gadget.sh - the gadget init script

You may/must adapt the 10 red parameter values for your own settings and needs.

```
#!/bin/sh
#########################
# gadget.sh
# Audio Gadget setup
```

```
# RpiOS (Debian) / RPi 3+
##########################
#
# Daihedz
# 23/06/10/ 00:00


### Settings ###

# Directories - !!! must be adapted !!!
CONFIGFS_ROOT=/sys/kernel/config  # adapt to your machine
GDG_DIRNAME="audio-basic"  # adapt as you like

# Basics - better not to be changed, because values are standard
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  # no need to adapt - 0x409 is a standard value (for US English)
STRG_MANUFACTURER="DIYAudio"  # adapt as you like
STRG_PRODUCT="UAC2Gadget4448"  # adapt as you like
STRG_SERIALNUMBER="000001"  # adapt as you like

# Configuration(s) - likely/optionally to be adapted
CONFIGURATION_CNF_1="UAC2Config4448"  # adapt as you like

# Functions - !!! must be adapted to the audio format(s) to be processed !!!
AUDIO_CHANNEL_MASK_CAPTURE=3  # 1=Left 2=Right 3=Stereo 0=disables the device
AUDIO_CHANNEL_MASK_PLAYBACK=3
AUDIO_SAMPLE_RATES_CAPTURE=44100,48000
AUDIO_SAMPLE_RATES_PLAYBACK=44100,48000
AUDIO_SAMPLE_SIZE_CAPTURE=2  # 1 for S8LE / 2 for S16LE / 3 for S24LE / 4 for S32LE
AUDIO_SAMPLE_SIZE_PLAYBACK=2

### 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
```

L.2. home/pi/gdsp.sh - the Gadget DSP start script

The main program flow of this script (at the bottom of the file)
- sets the cpu clock to an initially high rate
- checks for the correct setup of the gadget
- shuts down some non-audio related OS load (kernel modules and systemd units)
- enters a loop
  - shuts down eventual erratic audio process residuals
  - starts the cpu governor in order to reduce the cpu clock for steady-state conditions
  - starts the CDSP processing
  - eventually match sampling rates after an eventual CDSP crash *
- continues to the begin of the loop

* Set stop_on_rate_change=true inside of the CDSP configuration file results in an auto-adaptive system in terms of eventual sampling rate changes e.g. at an SPDIF input.

The functionality is of this program flow is implemented within functions.

`function os_krnl_trim()` - unloads kernel modules
`function os_sysd_trim()` - stops systemd units
`function cpu_freq_set()` - controls the cpu governor
`function sw_gdgt_chk()` - checks for proper gadget setup
`function sw_prgs_kill()` - shuts down erratic program residues
`function sw_audio_prc()` - starts the audio processing
`function sw_rate_match()` - checks for and matches an eventual sampling rate mismatch

Program flow:

```
cpu_freq_set "init"
sw_gdgt_chk
os_krnl_trim
os_sysd_trim
while true; do
  sw_prgs_kill
  cpu_freq_set "lo" &
  sw_audio_prc
  cpu_freq_set "hi"
  sw_rate_match
done
```

This setup is a bit special in terms that it can be configured to resort to two CDSP instances for a split DSP handling for workload reasons. E.g. for a fist instance for input and data/format conversion, and a second instance for filtering. And also a bit for experimental purposes. A more typical system may come along with one single CDSP instance performing all the work. Another atypical approach might be the use a piped aplay backend.

You may/must adapt the 3 red parameter values for your own settings and needs.

```
#!/bin/bash
##############################
# gdsp.sh
# Gadget DSP Processor
# RpiOS / CamillaDSP / RPi 3+
##############################
#
# Daihedz
# 23/06/10/ 00:00
```

```
### Settings ###

# Host
cpu_freq_hi=1200000  # all  values in kHz
cpu_freq_lo=600000
snd_gdgt="UAC2Gadget"
prgs_run="aplay arecord camilladsp"

# CamillaDSP
cdsp_cnfg="/home/pi/_camilladsp/configs/UAC2_ESL63_SPDIF.yml"
cdsp_prt=1234
cdsp_cnfg_2="" # CamillaDSP dual mode - enabled by declaring a cdsp_2 configuration file
(otherwise set cdsp_2="")
cdsp_prt_2=0000

# OS kernel modules
modules_base="roles i2c_bcm2835 libcomposite regmap_i2c "
modules_gadget="dwc2 usb_f_uac2 u_audio"
modules_network="brcmfmac brcmutil cfg80211 rfkill"
#modules_network="" # networking might be disabled by setting value to ""
modules_snd="snd snd_bcm2835 snd_compress snd_pcm snd_pcm_dmaengine snd_soc_bcm2835_i2s
snd_soc_core snd_timer"
modules_wm8804="snd_soc_rpi_wm8804_soundcard snd_soc_wm8804 snd_soc_wm8804_i2c"  #
SPDIF_out might be disabled by setting value to ""
modules_stop=""

# OS systemd features
units_base="alsa-restore systemd-tmpfiles-setup"
units_stop="cron getty@tty1 rng-tools-debian systemd-journald systemd-timesyncd
triggerhappy"
targets_stop="cryptsetup nfs-client sockets swap time-set time-sync"


### Functions ###

## OS Dumping, leaning, cleaning ...

# Kernel
function os_krnl_trim() {
  # modules
  echo "script: info - unload some kernel modules ... "
  modules_prc=$modules_stop
  modules_chk=$modules_base" "$modules_gadget" "$modules_network" "$modules_snd"
"$modules_wm8804
  for module in $(lsmod | cut -d" " -f1 | tail -n +2) ; do  # list of loaded modules
    echo $modules_chk | grep -q $module &> /dev/null  # check module against the list of
modules to keep loaded
    if [ $? -eq 0 ]; then  # module is to keep loaded
      continue
    fi
    modules_prc=$modules_prc" "$module  # add module to processing ( the modules to
unload) list
  done
  for module in $modules_prc; do
    sudo modprobe -r $module  &> /dev/null
  done
}

# Systemd
function os_sysd_trim() {
  # targets
  echo "script: info - stop some targets ..."
  for target in $targets_stop ; do
    sudo systemctl stop $target.target &> /dev/null
  done
  # services - sockets - timers
  pools="timer socket service"  # three pools of systemd units to process
  for pool in $pools; do

echo ""
```

```
    echo "script: info - stop some systemd "$pool"s ... "
    # Establish processing list, sequentially for each pool
    units_prc=$units_stop
    units_chk=$(systemctl --state=running --type=$pool | grep "$pool" | cut -d"." -f1 |
tr -d " ")" "$units_base
    for unit in $(systemctl --state=active --type=$pool | grep "$pool" | cut -d"." -f1 |
tr -d " "); do  # list of active units
      echo $units_chk | grep -q $unit &> /dev/null  # check unit against the list of
units to keep active
      if [ $? -eq 0 ]; then   # unit might be to keep active per default
        echo $units_stop | grep -q $unit  &> /dev/null  # check unit against the list of
units to stop
        if [ $? -ne 0 ]; then   # unit is to be kept active, does not figure on the stop
list
          continue
        fi
      fi
      units_prc=$units_prc" "$unit  # add unit to processing (the units to stop) list
    done
    # Processing units
    for unit in $units_prc; do
      sudo systemctl stop $unit.$pool &> /dev/null
     done
  done
  # pam nologin issue for ssh into th headless system (as ssh is non-root)
  sudo rm /var/run/nologin &> /dev/null
}

## Host functions and status

# CPU clock
function cpu_freq_set() {
  if [ $1 = "lo" ]; then
  # CPU set to powersave - for steady-state audio processing
    sleep 0.3  # Delay for init of the audio pipe with max. ressources, then scale back
    sudo cpupower frequency-set --governor powersave &> /dev/null
  elif [ $1 = "hi" ]; then
    # CPU set to performance
    sudo cpupower frequency-set --governor performance &> /dev/null
  elif [ $1 = "init" ]; then
    # Set CPU clock governors frequency imits and set CPU to performance
    sudo cpupower frequency-set --min $cpu_freq_lo --max $cpu_freq_hi &> /dev/null
    sudo cpupower frequency-set --governor performance &> /dev/null
  fi
  # Info
  frq=$( sudo cat /sys/devices/system/cpu/cpu3/cpufreq/cpuinfo_cur_freq)
  ((frq /= 1000))
  echo "script: info - cpu frequency set to $frq MHz"
}

# Check for UAC gadget soundcard device to be fully functional
function sw_gdgt_chk() {
  echo "script: info - check for gadget device ..."
  # Part I - is it there?
  n=0
  while true; do
    (( n += 1 ))
     cat /proc/asound/cards | grep $snd_gdgt &> /dev/null
    if [ $? -eq 0 ]; then
      break
    else
      sleep 0.1
      if [ $n = 10 ] ; then
        echo "script: error - gadget device $snd_gdgt is not set up correctly"
        exit
      fi
    fi
  done
  # Part II: - functional?
  if [ "$(cat /proc/asound/$snd_gdgt/pcm0c/sub0/hw_params)" = "no setup" ]; then
    echo "script: error - gadget device $snd_gdgt is locked up in 'no_setup' state"
```

```
    echo "script: error - host reboot required"
    exit
  fi
  echo "script: info - $snd_gdgt ok"
}

# Clean up ev. programs residuals
function sw_prgs_kill() {
  echo "script: info - cleaning program residuals ..."
  # Clean programs
  for itm in $prgs_run; do
    if [ -n "$(pidof $itm)" ]; then
      sudo killall -9 $itm &> /dev/null
      while [ -n "$(pidof $itm)" ]; do
        sleep 0.01
      done
      echo "script: info - $itm"
    fi
  done
}

## Audio processing

# USB-in to SPDIF-out device processing
function sw_audio_prc() {
  echo "script: info - playback starting ..."
  if [ -z $cdsp_cnfg_2 ]; then
    # Single CDSP
    sudo taskset -c 2-3 nice -n -20 chrt -f 99 camilladsp $cdsp_cnfg -p $cdsp_prt -l warn
| \
    sudo taskset -c 2-3 nice -n -20 chrt -f 99 aplay -D hw:$sndcrd_out -c 2 -f S24_LE -r
96000 --mmap --nonblock --disable-channels --disable-format --disable-resample --disable-
softvol
  elif [ -a $cdsp_cnfg_2 ]; then
    # Tandem CDSP
    echo "script: info - camilladsp dual configuration"
    sudo taskset -c 2-3 nice -n -20 chrt -f 99 camilladsp $cdsp_cnfg -p $cdsp_prt -l warn
| \
    sudo taskset -c 2-3 nice -n -20 chrt -f 99 camilladsp $cdsp_cnfg_2 -p $cdsp_prt_2 -l
warn | \
    sudo taskset -c 2-3 nice -n -20 chrt -f 99 aplay -D hw:$sndcrd_out -c 2 -f S24_LE -r
96000 --mmap --nonblock --disable-channels --disable-format --disable-resample --disable-
softvol 2> /dev/null
  else
    echo "script: error - check camilladsp configuration file setup"
    exit
  fi
}

# Check for the incoming sampling rate and eventually adapt the (frontend) CDSP config
file to it
function sw_rate_match() {
  capt_rate=$(arecord -D hw:$snd_gdgt --dump-hw-params 2> >(grep -i "rate:") | cut -d ":"
-f2 | tr -d ' ')
  if [ -n "$capt_rate" ]; then
    echo "script: info - capture samplerate: $capt_rate Hz"
    grep -R "capture_samplerate: $capt_rate" $cdsp_cnfg &> /dev/null
    if [ $? -eq 1  ] ; then
      sed -i "/capture_samplerate/c\  capture_samplerate: \\$capt_rate" $cdsp_cnfg
    fi
  fi
}


### Main program ###

echo ""
# cpu governor limits and set max. CPU ressources
cpu_freq_set "init"
# gadget check
sw_gdgt_chk
```

```
# dump os components
os_krnl_trim
os_sysd_trim

while true; do
  echo ""
  echo "script: info - main main loop starting ..."
  # clean eventual program and functional residuals
  sw_prgs_kill
  # cpu governor set to lo for steady-state audio after a delay. Thus allowing init at
max. CPU ressources
  cpu_freq_set "lo" &
  # start audio processing
  sw_audio_prc
  # Beyond this point, either CamillaDSP has been stopped, or it crashed.
  # cpu set to max. ressources
  cpu_freq_set "hi"
  # eventually check for and fix sampling rate mismatch which might have caused
CamillaDSP to crash
  sw_rate_match
done
```

L.3. The systemd service init scripts

L.3.a. usr/lib/systemd/gadget.service - the gadget init autostart service file

You may/must adapt the 2 red parameter values for your own settings and needs.

```
[Unit]
Description=UAC2 Gadget init
After=multi-user.target

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

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

L.3.b. /usr/lib/systemd/gdsp.service - the gadget DSP start script autostart service file

You may/must adapt the 2 red parameter values for your own settings and needs. You may e.g. want to change the statement ExecStart to `ExecStart=camilladsp camillaconfigfile.yml` for a direct start of CDSP.

```
[Unit]
Description=CamillaDSP audio processing starter
After=multi-user.target

[Service]
CPUSchedulingPolicy=fifo
CPUSchedulingPriority=10
ExecStart=/home/pi/gdsp.sh
Group=root
User=root

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

Part V/V: Addendum

X. Lists of items

Take care: All these lists may be either incomplete, or contain units not belonging into them.

X.1. Programs to optionally remove

As described in part II/F:

avahi-daemon bash-completion bind9-host bluez bluez-firmware ca-certificates dphys-swapfile eject file gcc gettext-base gnupg libcamera-apps-lite libcamera0 man-db manpages ncurses-term ppp python3-libcamera python3-v4l2 tasksel usbutils zip

Comment: Getting rid of all these above listed programs and theirs dependencies will free more than 150MB of disk space. Running services drop by half from approx. 30 to approx. 15.

X.2. Kernel modules

As described in part II, section H.1.

X.2.a Kernel modules to eventually unload

Video related
bcm2835_codec bcm2835_isp bcm2835_v4l2 mc videobuf2_bcm2835_v4l2 videobuf2_common videobuf2_dma_contig videobuf2_memops videobuf2_vmalloc videobuf2_v4l2 videodev v4l2_mem2mem

Tuttifrutti
aes_arm64 aes_generic af_alg algif_aead algif_hash algif_skcipher backlight bluetooth bcm2835_mmal_vchiq btbcm btqca btsdio btrl cbc ccm ctr cmac drm drm_panel_orientation fuse garp gcm gf128mul ghash_generic hmac ip_tablesipv6 libaes llc md4 md5 raspberrypi_hwmon stp uio uio_pdrv_genirq vc_sm_cma vc4 x_tables

X.2.b Kernel modules to eventually keep loaded

basic OS modules
roles i2c_bcm2835 libcomposite regmap_i2c

gadget related modules
dwc2 usb_f_uac2 u_audio

network modules
brcmfmac brcmutil cfg80211 rfkill

audio/sound modules
snd snd_bcm2835 snd_compress snd_pcm snd_pcm_dmaengine snd_soc_bcm2835_i2s snd_soc_core snd_timer

SPDIF-out modules (HifiBerry Digi2 pro)
snd_soc_rpi_wm8804_soundcard snd_soc_wm8804 snd_soc_wm8804_i2c

X.3. Systemd units (services, sockets, timers) not to start

As described in part II, section H.2.

services to stop
apt-daily apt-daily-upgrade avahi-daemon console-setup cron dhphys-swapfile dhcpcd ifupdown-pre keyboard-setup kmod-static-nodes man-db ModemManager raspi-config rng-tools-debian rpi-eeprom-update rsyslog systemd-journal-flush systemd-journald systemd-random-seed systemd-remount-fs systemd-timesyncd systemd-udevd systemd-udev-trigger systemd-update-utmp triggerhappy
sockets to stop

apt-daily apt-daily-upgrade e2scrub_all fstrim logrotate systemd-fsckd systemd-initctl systemd-rfkill systemd-tmpfiles-dmp systemd-udevd-control systemd-udevd-kernel

targets to stop
cryptsetup nfs-client sockets swap time-set time-sync

timers to stop
apt-daily apt-daily-upgrade e2scrub_all fstrim logrotate systemd-tmpfiles-clean systemd-tmpfiles-dmp

Y. Useful system checks

Y.1. The UAC2 gadget functionality

```
$ lsmod | grep dwc2
$ ls -R /sys/class/udc/*
$ cat /proc/asound/cards
$ aplay -l
$ arecord -l
$ ls -R /sys/kernel/config/usb_gadget/*
```

Y.2. CamillaDSP

```
$ pidof camilladsp
$ ps -q $(pidof -s camilladsp) -eLo 'pid,tid,cls,rtprio,comm,pcpu,psr'
```

Y.3. Services and Modules

```
$ systemctl | grep running
$ systemctl | grep active

$ lsmod
```

Y.4. WiFi

```
$ ip route
$ systemctl | grep iwd
```

Y.5. CPU

```
$ sudo cat /sys/devices/system/cpu/cpufreq/policy0/cpuinfo_cur_freq
$ sudo cat /sys/devices/system/cpu/cpufreq/policy0/scaling_governor
$ sudo cat /sys/devices/system/cpu/isolated

...
```

Y.2314298 The little red LED there

```
$ isitlit
```

Z. Useful Links

Basics/Theory
https://www.kernel.org/doc/Documentation/ABI/testing/configfs-usb-gadget-uac2
https://www.kernel.org/doc/html/latest/usb/gadget_configfs.html
https://www.kernel.org/doc/html/v6.1/usb/gadget-testing.html

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

https://www.collabora.com/news-and-blog/blog/2019/02/18/modern-usb-gadget-on-linux-and-how-to-integrate-it-with-systemd-part-1/
https://www.collabora.com/news-and-blog/blog/2019/03/27/modern-usb-gadget-on-linux-and-how-to-integrate-it-with-systemd-part-2/

Software/Programs
https://downloads.raspberrypi.org/imager/imager_latest.exe
https://github.com/Henquist/camilladsp/releases/download/v1.0.3/camilladsp-linux-aarch64.tar.gz

Forums further discussion
https://www.diyaudio.com/community/threads/camilladsp-cross-platform-iir-and-fir-engine-for-crossovers-room-correction-etc.349818/

Rpi configuration
https://www.raspberrypi.com/documentation/computers/config_txt.html

Special thanks to H.Enquist (CamillaDSP) and P.Hofman (ALSA/Linux) and to many others having contributed

17.6.2023
Daihedz, a poster of diyaudio.com 's