Qualcomm Technologies, Inc.

Qualcomm Linux Audio Guide

80-70018-16 AA

April 4, 2025

Contents

1 Audio documentation **3**
- 1.1 Audio overview . . . 3
- 1.2 Enable audio . . . 3
- 1.3 Customize audio use case . . . 3
- 1.4 Troubleshoot audio . . . 4
- 1.5 Advanced audio features . . . 4

2 Audio overview **5**
- 2.1 Architecture . . . 6

3 Enable audio **8**
- 3.1 Enable audio with GStreamer . . . 15
- 3.2 Enable audio with PulseAudio . . . 16

4 Customize audio use case **27**
- 4.1 Customize at PAL level . . . 27
- 4.2 Enable TinyALSA-based applications . . . 35
- 4.3 Customize audio graph . . . 43

5 Troubleshoot audio **50**
- 5.1 Capture logs . . . 50
- 5.2 Analyze logs . . . 51

6 Advanced audio features **54**
- 6.1 Minimize echo and noise . . . 54
- 6.2 Compress offload . . . 57
- 6.3 Enable A2DP . . . 57
- 6.4 Minimize echo and noise . . . 59
- 6.5 Compress offload . . . 63

7 References **64**
- 7.1 Related documents . . . 64
- 7.2 Acronyms and terms . . . 64

1 Audio documentation

Use the audio subsystem to set up and customize audio devices.

1.1 Audio overview

Audio components

Learn more about available audio components such as the application processor and low-power AI.

Audio architecture

Review audio software architecture to find major audio components.

1.2 Enable audio

Set up audio hardware

Configure audio hardware for microphone and speaker connections.

Enable audio with GStreamer

Record and play back audio files in WAV, MP3, and FLAC formats.

Enable audio with PulseAudio

Record and play back audio files using the PulseAudio sound server.

1.3 Customize audio use case

Configure at the PAL level

Configure audio at the PAL level for PCM or hardware-accelerated formats.

Enable TinyAlsa-based applications

Configure TinyAlsa-based virtual mixer controls.

Customize audio graph

Add custom audio graphs to enable audio use cases.

1.4 Troubleshoot audio

Capture user space logs

Capture user space logs to identify audio issues.

Capture kernel logs

Capture kernel logs to identify audio issues.

Analyze logs for record and playback

Analyze audio driver logs to debug record and playback issues.

1.5 Advanced audio features

Enable VoIP ECNS

Reduce noise and echo in VoIP.

Compress offload playback

Decode compressed audio streams in low-power AI.

Enable A2DP source and sink

Encode or decode audio using A2DP source and sink.

2 Audio overview

The audio subsystem powered by low power AI (LPAI) delivers voice UI and audio experiences. It uses a dedicated hardware-based AI accelerator for machine learning-based work.

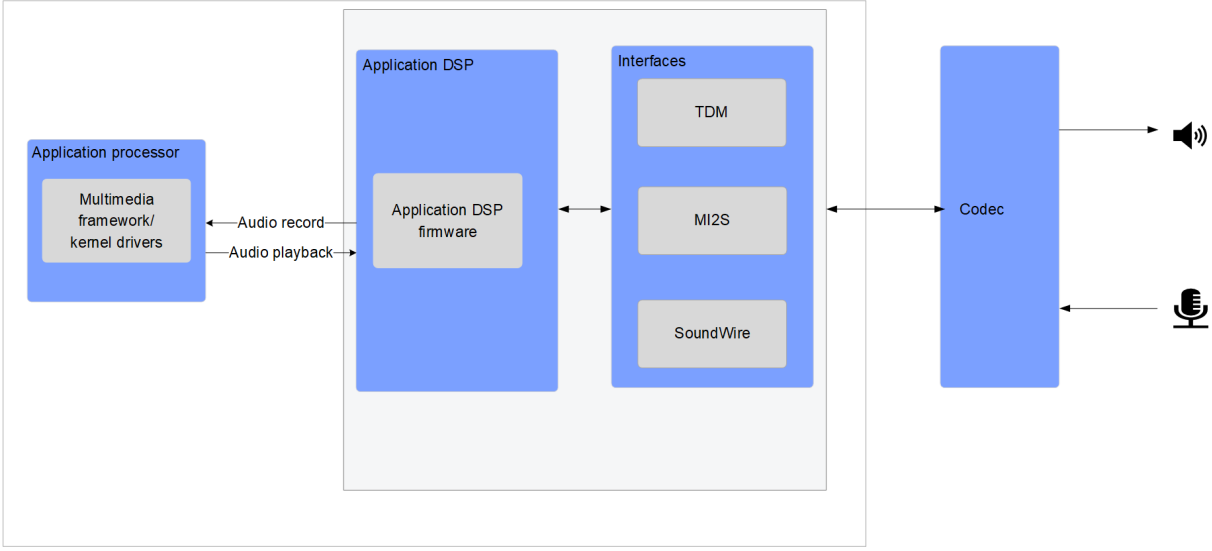


Figure1 Audio component overview

The audio system includes:

- **Application processor** – CPU that handles audio processing tasks. Tasks include:
  - Managing audio record and playback
  - Decoding audio formats
  - Using LPAI for postprocessing tasks
- **Low Power AI (LPAI)** – Subsystem that runs audio playback/record and voice-activation (VA) algorithms. It integrates with a dedicated Qualcomm® Hexagon™ Processor (QDSP6) and a low power island (LPI).
- **Audio codec** – Hardware that includes:
  - Analog-to-Digital Converter (ADC)

– Digital-to-Analog Converter (DAC)

These convert analog audio to digital, and vice versa.

- **Speaker AMP and microphone** – Devices that connect over I2S/TDM/SoundWire.

2.1 Architecture

The following figure shows the high-level audio software architecture.

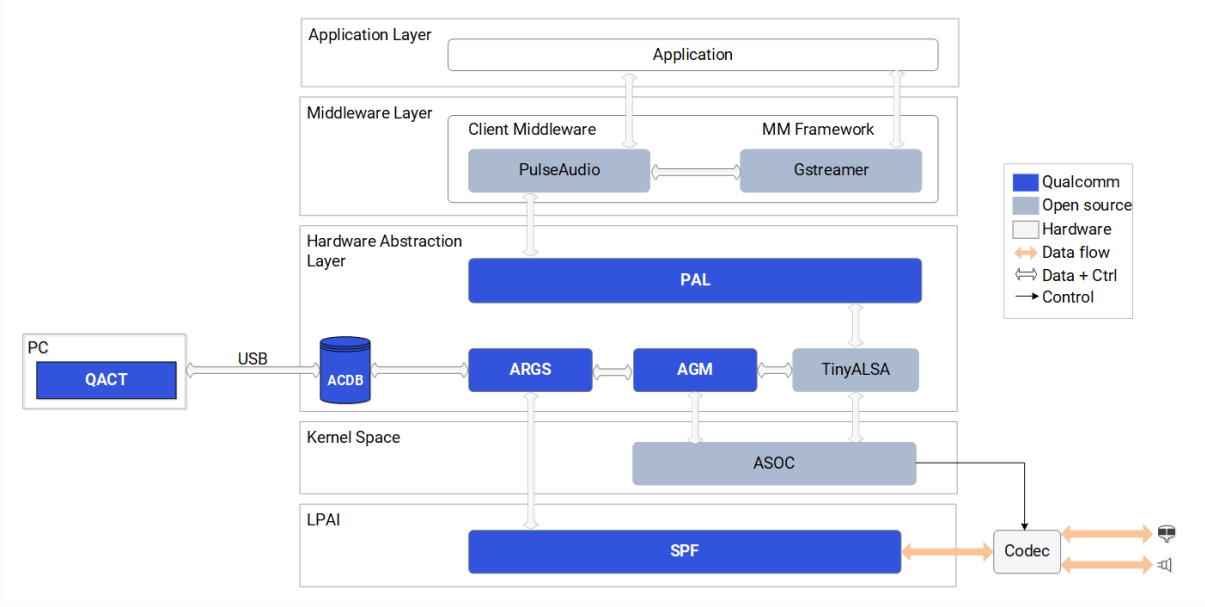


Figure2 High-level audio software architecture

The following are the major audio software architecture components:

PulseAudio

A sound server for POSIX OSes (mostly targeting Linux) that acts as a proxy and router between hardware device drivers and applications on one or many hosts.

Platform Abstration Layer (PAL)

Provides higher-level audio-specific APIs to access the audio hardware and drivers to enable audio features.

Audio Graph Manager (AGM)

Provides interfaces to allow TinyALSA-based mixer controls and PCM/compressed plug-ins to interact and enable audio features.

AudioReach Graph Service (ARGS)

Consists of the Graph Service Layer (GSL), Generic Packet Router (GPR), and acdb Management Layer (AML) modules. Handles initialization and creation of graphs, and creation of packets for sending a series of commands to the signal processing framework (SPF).

Audio calibration database (acdb)

Includes information about various audio use cases such as graphs, module calibration data, etc. The APPS processor parses acdb files to get the graph information used by SPF to enable the use case.

Signal Processing Framework (SPF)

Modular framework that runs on the LPAI DSP. It helps set up, configure, and run signal processing modules for audio features.

Qualcomm Audio Calibration Tool (QACT™ Platform)

PC-based software that provides a GUI to visualize, configure, and store audio graphs in the acdb for audio use cases.

3 Enable audio

This section shows how to configure hardware for microphone and speaker connections and gives steps to verify basic audio use cases.

QCS6490

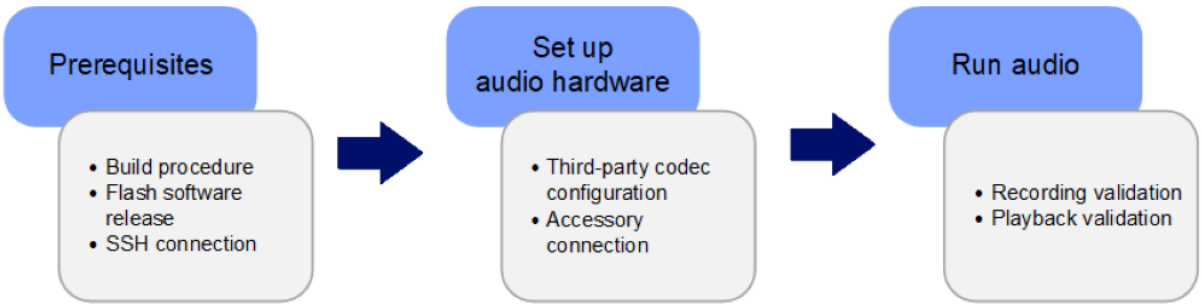


Figure1 Setup flow

Prerequisites

- Set up your infrastructure as described in the Qualcomm Linux Build Guide
- Flash the latest software release to the development board.
- Set up an SSH connection:
  1. Enable SSH in Permissive mode by performing the steps mentioned in Use SSH.
  2. Connect to the device:

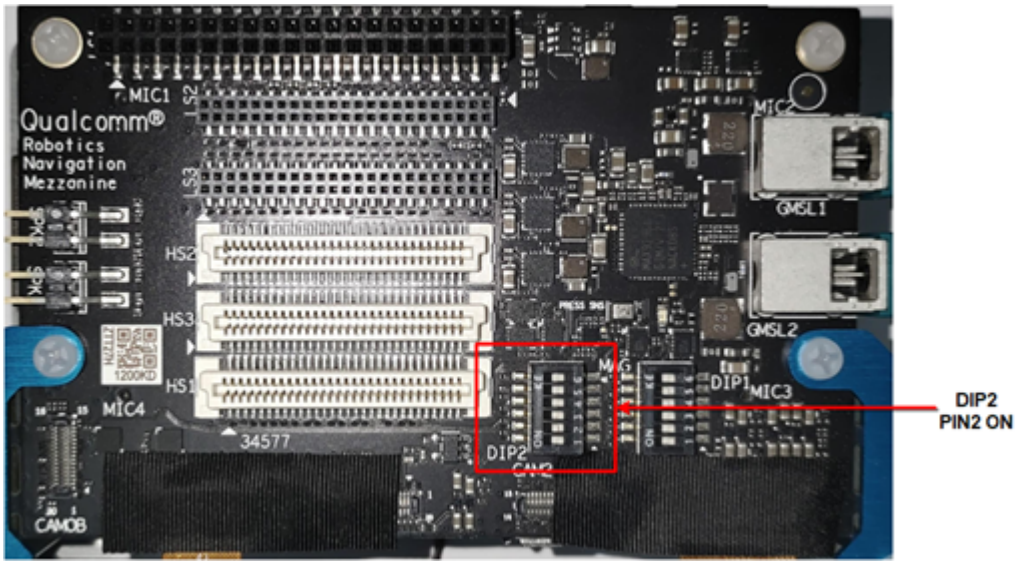
     ```
     ssh root@<device_IP_address>
     ```

     For example, if the IP address of the device is 10.92.160.222, run the following command:

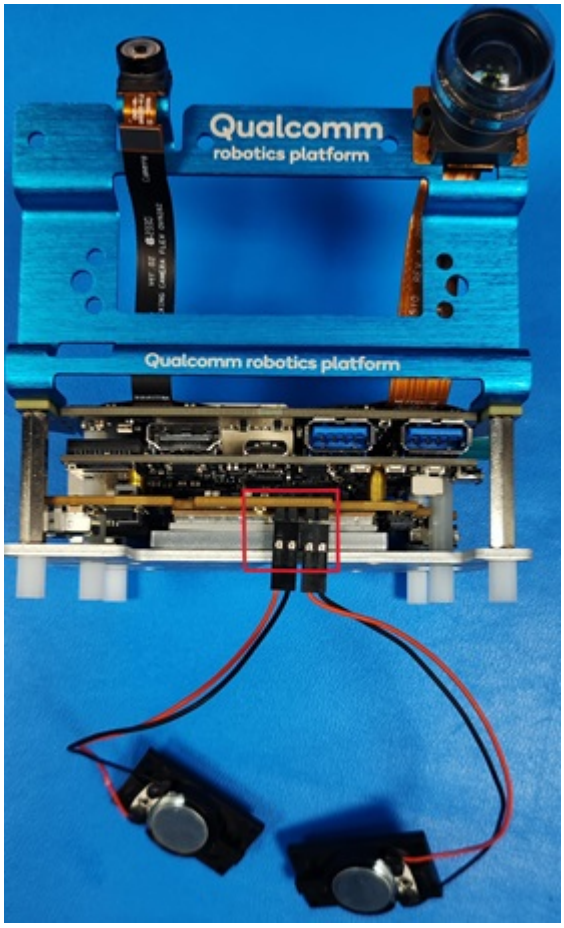
     ```
     ssh root@10.92.160.222
     ```

Set up audio hardware

1. To activate the digital microphone interface (DMIC) on the board, use DIP 2. Switch PIN 2 to the ON position.



2. Connect the speaker to the board.



QCS9075

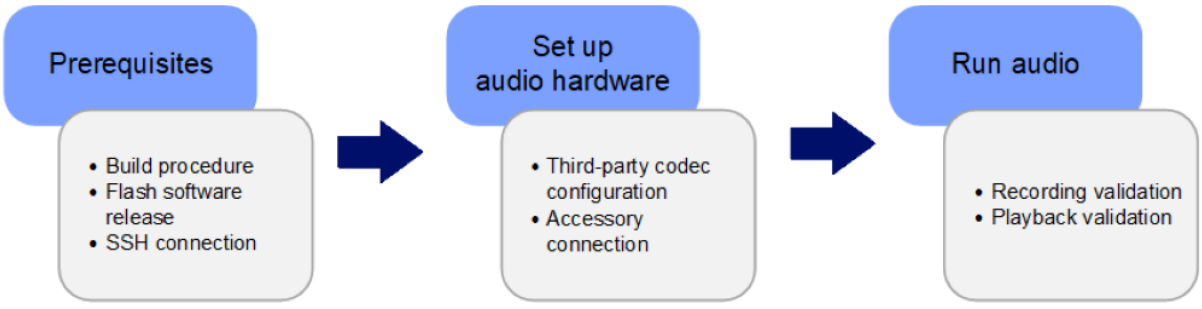


Figure2 Setup flow

Prerequisites

- Set up your infrastructure as described in the Qualcomm Linux Build Guide
- Flash the latest software release to the development board.
- Set up SSH connection:
  1. Enable SSH in Permissive mode by performing the steps mentioned in Use SSH
  2. Connect to the device by running the following command:

     ```
     ssh root@<device_IP_address>
     ```

     For example, if the IP address of the device is 10.92.160.222, run the following command:

     ```
     ssh root@10.92.160.222
     ```

Set up audio hardware

Connect the speaker to the board.

Note: EVT boards need hardware rework for audio functions to work.

QCS8275

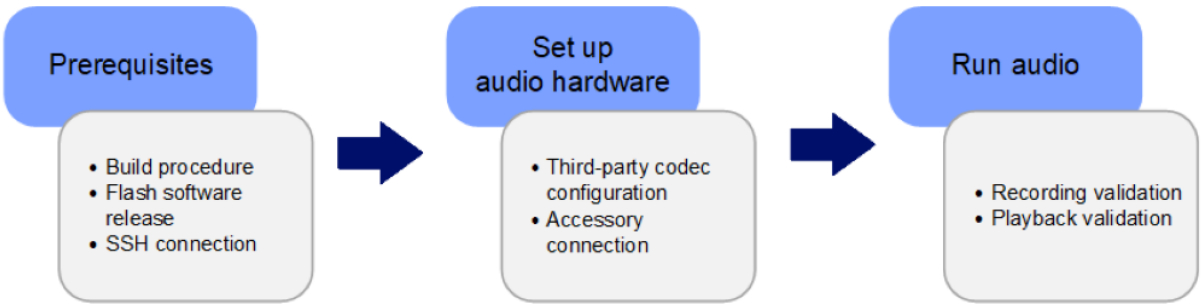


Figure3 Setup flow

Note: The Audio functionality requires firmware and license from NXP or other third-party codec. Mercury Codec from NXP is the default codec with QCS8275.

Prerequisites

- Set up your infrastructure as described in the Qualcomm Linux Build Guide
- Flash the latest software release to the development board.
- Set up an SSH connection.
  1. Enable SSH in Permissive mode. For instructions, see Use SSH
  2. Connect to the device:

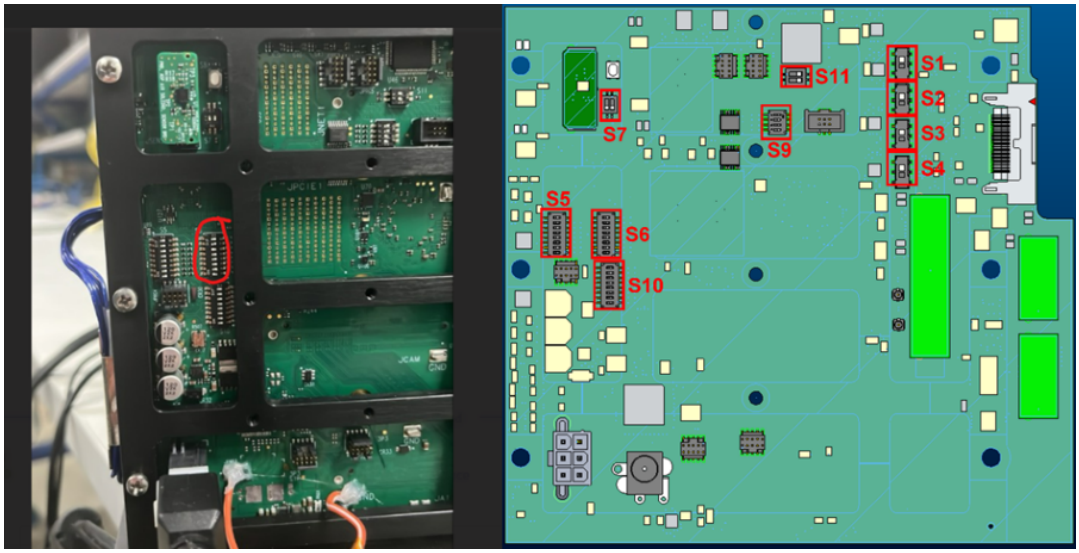
     ```
     ssh root@<device_IP_address>
     ```

     For example, if the IP address of the device is 10.92.160.222, run:

     ```
     ssh root@10.92.160.222
     ```

Set up audio hardware

1. Get the required license and firmware from NXP for Mercury Codec.
2. Power off the device.
3. Open the back panel of the device.
4. Set the DIP switch S6 bit7 to ON and the DIP switch S6 bit8 to OFF.



5. Power on the device.
6. Run the following commands:

```
ssh root@ip-addr
mount -o rw,remount /
```

7. Push the Mercury firmware received from NXP to the following location:

```
ssh root@ip-addr
scp -r mercury_firmware\. root@[ip-addr]:/etc/firmware/
```

8. Set the device in Flashing mode. Select 1 (Flash Mercury) followed by 7 with `dac_mer_testapp`.

```
ssh root@ip-addr
dac_mer_testapp
```

9. Flash the firmware:

```
mercuryflasher -f
```

If the above command fails, check the output of the following commands for debug.

Check if the ping works:

```
mercuryflasher -p
```

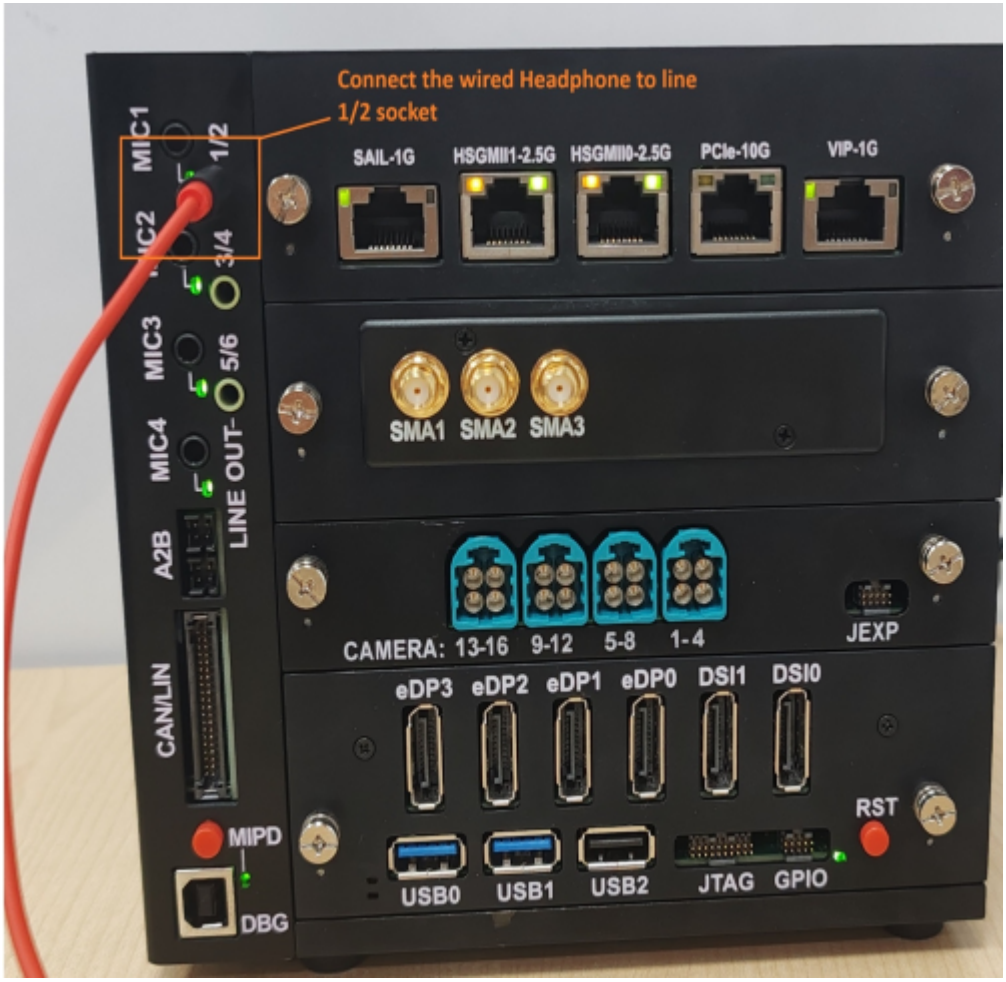
Check if the read works:

```
mercuryflasher -r
```

Note: The commands should all return success or 0a.

The Mercuryflasher tool doesn't work directly on the first candidate build. See the latest tool for details.

10. Power off the device.
11. Set the DIP switch S6 bit7 to OFF. Use the preceding figure as reference.
12. Power on the device.
13. For playback, connect the headphone/speaker device to the lineout 1/2 port as follows.



14. For recording, connect the microphone/headset device to the MIC1 port as follows.

3.1 Enable audio with GStreamer

To decode and encode audio using GStreamer apps, see the following.

- Audio decode example
- Audio encode example

Note: To check the GStreamer app for different chipsets, see Source information for multimedia

sample applications.

GStreamer is an open source multimedia framework. Qualcomm provides GStreamer plug-ins as part of the QIMP SDK.

GStreamer plugins

- GStreamer plug-ins for audio decoder and encoder are part of QIMSDK. Download the entire QIMSDK to use pulsesrc and pulsesink.
- The Qualcomm Intelligent Multimedia Product (QIMP) SDK Quick Start Guide describes how to download and build the QIMP SDK.

GStreamer sample apps

Sample GStreamer apps for audio use cases are part of the QIMP SDK. Before running sample apps, meet these prerequisite steps.

Run audio use cases with either command line or the GST app.

Play and record audio with the Gstreamer app

Use command line to run `gst-launch-1.0` for audio playback/capture.

Note: For QCS8275, enable playback and record using `dac_mer_testapp` as described in Enable audio.

3.2 Enable audio with PulseAudio

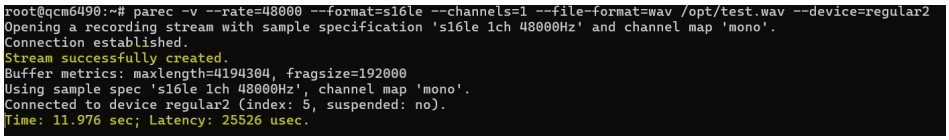
PulseAudio record

QCS6490

1. Set up PulseAudio recording:

```
parec -v --rate=48000 --format=s16le --channels=1 --
file-format=wav /opt/test.wav --device=regular2
```

   The command shell should resemble the following:



2. Press *Ctrl + C* to stop the recording.

   Supported formats are `s16le`, `s24le`, `s32le`, `s24-32le`, rate can be `8000`, `16000`, `22050`, `24000`, `32000`, `44100`, `48000`, `88200`, `96000`, `176400`, `192000`, `352800`, `384000`, `705600`, and `768000`, and channels can range from `1` to a maximum of `8`.

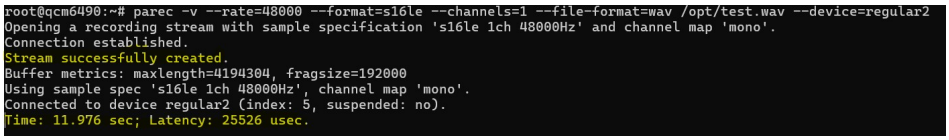
QCS9075

1.

   Set up PulseAudio recording:

```
parec -v --rate=48000 --format=s16le --channels=1 --
file-format=wav /opt/test.wav --device=regular2
```

   The command shell should resemble the following:



2. Press **Ctrl + C** to stop the recording.

   Supported formats are `s16le`, `s24le`, `s32le`, `s24-32le`, rate can be `8000`, `16000`, `22050`, `24000`, `32000`, `44100`, `48000`, `88200`, `96000`, `176400`, `192000`, `352800`, `384000`, `705600`, and `768000`, and channels can range from `1` to a maximum of `4`.

QCS8275

1.

To start a playback or record use case, configure the Mercury Codec and DAC.

Enable playback and capture:

```
ssh root@ip-addr
dac_mer_testapp
```

2. Select option 2 (SOC Mercury) followed by 1 (Enable Playback Setup) for playback and record use cases.
3. Run the following commands in a new command shell:

```
ssh root@ip-addr
parec -v --rate=48000 --format=s16le --channels=1 --
file-format=wav /opt/test.wav --device=regular2
```

The command shell should resemble the following:

```
root@qcm6490:~# parec -v --rate=48000 --format=s16le --channels=1 --file-format=wav /opt/test.wav --device=regular2
Opening a recording stream with sample specification 's16le 1ch 48000Hz' and channel map 'mono'.
Connection established.
Stream successfully created.
Buffer metrics: maxlength=4194304, fragsize=192000
Using sample spec 's16le 1ch 48000Hz', channel map 'mono'.
Connected to device regular2 (index: 5, suspended: no).
Time: 11.976 sec; Latency: 25526 usec.
```

4. Press *Ctrl + C* to stop the recording.

Supported formats are `s16le`, `s24le`, `s32le`, `s24-32le`, rate can be `8000`, `16000`, `22050`, `24000`, `32000`, `44100`, `48000`, `88200`, `96000`, `176400`, `192000`, `352800`, `384000`, `705600`, and `768000`, and channels can range from `1` to a maximum of `4`.

Volume settings for recording

QCS6490

1. Start audio capture using the `parec` utility.
2. Open a new command prompt in parallel. Open `ssh root@ip-addr` and run the following command to set the volume level:

```
pactl set-source-volume regular2 <volume level>
```

<value> varies from 0 to 65535.

QCS9075

1. Start capture using the parec utility.
2. Open a new command prompt in parallel. Open ssh root@ip-addr and run the following commands:

```
pactl set-source-volume regular2 <volume level>
```

<value> varies from 0 to 65535.

QCS8275

1. Start capture using the parec utility.
2. Open a new command prompt in parallel. Open ssh root@ip-addr and run the following commands:

```
pactl set-source-volume regular2 <volume level>
```

<value> varies from 0 to 65535.

PulseAudio playback

QCS6490

1. Push the .wav audio file to the device for playback:

```
scp test.wav root@[ip-addr]:/opt/
```

2. After pushing the test.wav file, enter the device shell using ssh root@ip-add.
3. Start playback:

```
paplay /opt/test.wav -v
```

The command shell should resemble the following:

```
root@qcm6490:/# paplay /opt/test.wav -v
Opening a playback stream with sample specification 's16le 2ch 48000Hz' and channel map 'front-left,front-right'.
Connection established.
Stream successfully created.
Buffer metrics: maxlength=4194304, tlength=384000, prebuf=380164, minreq=3840
Using sample spec 's16le 2ch 48000Hz', channel map 'front-left,front-right'.
Connected to device low-latency0 (index: 0, suspended: no).
Stream started.
Time: 8.194 sec; Latency: 2018945 usec.
```

QCS9075

1. Push the .wav file to the device:

   ```
   scp test.wav root@[ip-addr]:/opt/
   ```

2. After pushing the test.wav file, enter the device shell using `ssh root@ip-add`.
3. Start playback:

   ```
   paplay /opt/test.wav -v
   ```

   The command shell should resemble the following:

   ```
   root@qcm6490:/# paplay /opt/test.wav -v
   Opening a playback stream with sample specification 's16le 2ch 48000Hz' and channel map 'front-left,front-right'.
   Connection established.
   Stream successfully created.
   Buffer metrics: maxlength=4194304, tlength=384000, prebuf=380164, minreq=3840
   Using sample spec 's16le 2ch 48000Hz', channel map 'front-left,front-right'.
   Connected to device low-latency0 (index: 0, suspended: no).
   Stream started.
   Time: 8.194 sec; Latency: 2018945 usec.
   ```

QCS8275

1. Push the .wav file to the device:

   ```
   scp test.wav root@[ip-addr]:/opt/
   ```

2. After pushing the test.wav file, enter the device shell using `ssh root@ip-add`.
3. Use the following command for playback:

   ```
   paplay /opt/test.wav -v
   ```

   The command shell should resemble the following:

```
root@qcm6490:/# paplay /opt/test.wav -v
Opening a playback stream with sample specification 's16le 2ch 48000Hz' and channel map 'front-left,front-right'.
Connection established.
Stream successfully created.
Buffer metrics: maxlength=4194304, tlength=384000, prebuf=380164, minreq=3840
Using sample spec 's16le 2ch 48000Hz', channel map 'front-left,front-right'.
Connected to device low-latency0 (index: 0, suspended: no).
Stream started.
Time: 8.194 sec; Latency: 2018945 usec.
```

Volume settings for playback

QCS6490

1. Start audio playback on the speakers using the `paplay` utility.
2. Open a new command prompt in parallel. Open `ssh root@ip-addr` and run the following commands to set the volume level:

```
pactl set-sink-volume low-latency0 <volume level>
```

`<value>` varies from 0 to 65535.

QCS9075

1.

Start playback on speaker using the `paplay` utility.

2. Open a new command prompt in parallel. Open `ssh root@ip-addr` and run the following commands:

```
pactl set-sink-volume low-latency0 <volume level>
```

`<value>` varies from 0 to 65535.

QCS8275

1. Start playback on speaker using the `paplay` utility.
2. Open a new command prompt in parallel. Open `ssh root@ip-addr` and run the following commands:

```
pactl set-sink-volume low-latency0 <volume level>
```

`<value>` varies from 0 to 65535.

PulseAudio is a general purpose sound server. It runs as middleware between applications and hardware devices.

Find the source code for PulseAudio at `build-qcom-wayland/workspace/sources/pulseaudio`.

This section lists commonly used PulseAudio APIs. See the open source PulseAudio documentation for a complete description of all APIs .

For more information, see the PulseAudio API documentation

pa_stream_new

Creates a new, unconnected stream with the specified name and sample type.

```
pa_stream* pa_stream_new (
     pa_context * c,
     const char *name,
     const pa_sample_spec * ss,
     const pa_channel_map *map)
```

Parameter

c	Context in which to create the stream.
name	Stream name.
ss	Sample format.
map	Channel map. NULL for default.

Return value

pa_stream* handle

pa_stream_get_state

Returns the current stream state.

```
pa_stream_state_t pa_stream_get_state(
     const pa_stream *p)
```

Parameters

p	pa_stream handle.

Return value

Returns the pa_stream_state_t

pa_stream_get_device_name

Returns the sink or source name this stream connects to in the server.

```
const char* pa_stream_get_device_name(const pa_stream *s)
```

Parameters

s	pa_stream handle.

Return value

Device name

pa_stream_connect_playback

Connects the stream to a sink.

```
int pa_stream_connect_playback (
     pa_stream * s,
     const char * dev,
     const pa_buffer_attr *attr,
     pa_stream_flags_t flags,
     const pa_cvolume *volume,
     pa_stream *sync_stream)
```

Parameter

s	Stream to connect to a sink.
dev	Sink name to which to connect. NULL to let the server decide.
attr	Buffering attributes. NULL for default.
flags	More flags. 0 for default.
volume	Initial volume. NULL for default.
sync_stream	Synchronize this stream with the specified one. NULL for a standalone stream.

Return value

Returns zero on success.

pa_stream_connect_record

Connects the stream to a source.

```
int pa_stream_connect_record(
     pa_stream *s,
     const char * dev,
     const pa_buffer_attr * attr,
     pa_stream_flags_t flags)
```

Parameters

s	Stream to connect to a source.
dev	Source name to which to connect. NULL to let the server decide.
attr	Buffer attributes. NULL for default.
flags	More flags. 0 for default.

Return value

Returns zero on success.

pa_stream_write

Writes data to the server for playback streams.

```
int pa_stream_write(
     pa_stream * p,
     const void * data,
     size_t nbytes,
     pa_free_cb_t free_cb,
     int64_t offset,
     pa_seek_mode_t seek)
```

Parameters

p	Stream to use.
data	Data to write.
nbytes	Data length to write in bytes. Must be in multiples of the stream's sample specification frame size.
free_cb	Data clean-up routine. NULL to request an internal copy.
offset	Seeking offset. Must be 0 for upload streams. Must be in multiples of the stream's sample specification frame size.
seek	Seek mode. Must be PA_SEEK_RELATIVE for upload streams.

Return value

Returns zero on success.

pa_stream_set_write_callback

Sets the callback function that's called when new data can write to the stream.

```
void pa_stream_set_write_callback(
     pa_stream * p,
     pa_stream_request_cb_t cb,
     void *userdata)
```

Parameters

p	Stream to use.
cb	Callback function to set.
userdata	User data to send to callback.

Return value

None

pa_stream_set_read_callback

Sets the callback function that's called when new data is available from the stream.

```
void pa_stream_set_read_callback(
     pa_stream * p,
     pa_stream_request_cb_t cb,
     void *userdata)
```

Parameters

p	Stream to use.
cb	Callback function to set.
userdata	User data to send to callback.

Return value

None

pa_stream_disconnect

Disconnects a stream from a source/sink.

```
int pa_stream_disconnect(pa_stream *s)
```

Parameters

s	pa_stream handle.

Return value

Returns zero on success.

For more information, see the PulseAudio API documentation.

4 Customize audio use case

This section describes audio customizations, software configurations, interface customization, and custom module additions.

4.1 Customize at PAL level

PulseAudio audio manager

PulseAudio manages all audio applications, local and network streams, devices, filters, and audio inputs and outputs.

PulseAudio PAL plug-in

The PulseAudio PAL plug-in loads PAL. It allows the client to configure the audio devices and invoke various audio use cases.

The following are the components of the PulseAudio PAL plug-in:

Card	Represents a sound card, which is a group of supported profiles, ports, sinks, and sources. Card module uses include: • Load/unload PAL • Create/free a port supported by PAL • Create/free PAL card • Create/free sink or sources - Sinks are the playback path - Sources are the capture path
Sink	Configures the audio playback path. Sink module uses include: • Open/close a playback session based on the configuration specified by the module-pal-card • Create/free the sink thread for writing data for a PAL playback session • Set volume support • Get latency support • Routing support
Source	Configures of the audio capture path. Source module uses include: • Open/close a PAL record session based on the configuration specified by module-pal-card • Create/free PA sources • Create source thread for reading data from a PAL record session • Routing support

PAL APIs

PAL provides higher-level audio APIs to access audio hardware and drivers. This allows audio features including:

- Audio playback and record in PCM and compressed formats with various pre/postprocessing modules. This includes latency and power-sensitive use cases.
- Hardware accelerated encoding/decoding of audio formats.

Clients can interface with PAL using various APIs for controlling and configuring sessions and devices. PAL performs the following operations:

- Configures mixer controls to set up hardware codec devices and stream configurations
- Invokes TinyALSA APIs to open/start audio sessions

The PAL resource manager tracks all active sessions and devices to enable concurrencies. It parses and loads configuration of the following platform XML files:

- `Resource_manager.xml` – Device-to-backend mapping and policy making attributes
- `Card-defs.xml` – Virtual PCM and compress nodes, and their options

The PAL payload builder constructs the metadata payload for use case graphs. For more information about graphs, see Customize audio graphs.

PulseAudio uses the same PAL APIs for different audio use cases. Find the source code for the PAL module at:

`build-qcom-wayland/workspace/sources/qcom-pal/opensource/arpal-lx`

The following file gives all APIs exposed by the PAL module:

`build-qcom-wayland/workspace/sources/qcom-pal/opensource/arpal-lx/inc/PalApi.h`

The following are common PAL APIs.

pal_init

Initializes PAL, parses the related configuration files, and stores them in a local structure for use.

```
int32_t pal_init( )
```

Parameters

None

Return value

- 0 on success
- Error code on failure

pal_deinit

De-initializes PAL and frees up the resources allocated during initialization.

```
void pal_deinit()
```

Parameters

None

Return value

- 0 on success
- Error code on failure

pal_stream_open

Opens a stream with the specified configuration such as source/sink devices, media configuration, etc. Returns the stream handle upon success.

```
int32_t pal_stream_open(
     struct pal_stream_attributes *attributes,
     uint32_t no_of_devices,
     struct pal_device *devices,
     uint32_t no_of_modifiers,
     struct modifier_kv *modifiers,
     pal_stream_callback cb,
     uint64_t cookie,
     pal_stream_handle_t **stream_handle)
```

Parameters

attributes	Valid stream attributes obtained from pal_stream_open.
no_of_ devices	Number of audio devices with which to first start the stream.
devices	Array of pal_devices. Bases the size of the array on the number of devices specified by the client in no_of_devices.
no_of_ modifiers	Number of modifiers.
modifiers	Array of modifiers. Modifiers add more key-value pairs.
cb	Callback function associated with the stream. This callback function notifies any event.
cookie	Client data associated with the stream. The callback function returns this cookie.
stream_ handle	Updates with the valid stream handle if the operation is successful.

Return value

- 0 on success
- Error code on failure

pal_stream_start

Starts a stream.

```
int32_t pal_stream_start(
     pal_stream_handle_t *stream_handle)
```

Parameters

stream_handle	Valid stream handle from pal_stream_open.

Return value

- 0 on success
- Error code on failure

pal_stream_read

Reads the audio buffer captured from the audio source device.

```
ssize_t pal_stream_read(
     pal_stream_handle_t *stream_handle,
     struct pal_buffer *buf)
```

Parameters

stream_handle	Valid stream handle from pal_stream_open.
buf	Pointer to pal_buffer that has audio samples and metadata.

Return value

- Number of bytes read
- Error code on failure

pal_stream_write

Writes the audio buffer for stream rendering over a sink device.

```
ssize_t pal_stream_write(
     pal_stream_handle_t *stream_handle,
     struct pal_buffer *buf)
```

Parameters

stream_handle	Valid stream handle from pal_stream_open.
buf	Pointer to pal_buffer that has audio samples and metadata.

Return value

- Number of bytes read
- Error code on failure

pal_stream_stop

Stops a stream.

```
int32_t pal_stream_stop(
     pal_stream_handle_t *stream_handle)
```

Parameters

stream_handle	Valid stream handle from pal_stream_open.

Return value

- 0 on success
- Error code on failure

pal_stream_close

Closes a stream.

```
int32_t pal_stream_close(
     pal_stream_handle_t *stream_handle)
```

Parameters

stream_handle	Valid stream handle from pal_stream_open.

Return value

- `0` on success
- Error code on failure

Configuration files

Configure audio use cases at the PAL level by using the `mixer_paths`, `resourcemanager`, and `usecasekvmanager` XML files.

The following table shows the configuration files for different hardware versions:

QCS6490 configuration files

Variant	Sound card name	Conf file	mixer_paths.xml	ResourceManager.xml
Core Kit (Qualcomm RB3 Platform)	qcm6490-rb3-snd-card	qcm6490-rb3-snd-card.conf	mixer_paths_qcm6490_rb3.xml	resourcemanager_qcm6490_rb3.xml
Vision Kit	qcm6490-vision-snd-card	qcm6490-vision-snd-card.conf	mixer_paths_qcm6490_vision.xml	resourcemanager_qcm6490_vision.xml
Video Collab Kit	qcm6490-vc-snd-card	qcm6490-vc-snd-card.conf	mixer_paths_qcm6490_vc.xml	resourcemanager_qcm6490_vc.xml

QCS9075 configuration files

Variant	Sound card name	Conf file	mixer_paths.xml	ResourceManager.xml
Core Kit (RB8)	qcs9075-rb8-snd-card	qcs9075-rb8-snd-card.conf	mixer_paths_qcs9075_rb8.xml	resourcemanager_qcs9075_rb8.xml

QCS8275 configuration files

Variant	Sound card name	Conf file	mixer_paths.xml	ResourceManager.xml
Qualcomm IQ-8275 Beta Evaluation Kit	qcs8300-ridesx-snd-card	qcs8300-ridesx-snd-card.conf	mixer_paths_qcs8300_ridesx.xml	resourcemanager_qcs8300_ridesx.xml

Customize mixer paths XML file

Mixer control is a control variable exposed from the ALSA mixer to the user space. It allows the user space to access set and get functions and pass parameters to the ALSA mixer.

For example, the following is the entry for enabling the mono speaker device in the `mixer_path.xml` file. When playback triggers and the device selected is a speaker, the following mixer controls run with the help of the audio route helper class.

```
<path name="speaker">
<!--Mixer controls related to speaker need to be defined here-->
</path>
```

The platform uses the `mixer_paths_<sound-card-name>.xml` file as the mixer path. This file is in the `/etc/` folder on the target.

Customize resource manager XML file

The `Resourcemanager.xml` file includes all possible devices, use cases, and combinations. It also includes other configurations, module parameters, and global parameters.

The following is an example for a speaker device entry in the resource manager XML file. It has all configurations for the speaker device such as back-end name, channels, sample-rate, bit-width, etc.

```
<out-device>
     <id>PAL_DEVICE_OUT_SPEAKER</id>
     <back_end_name>CODEC_DMA-LPAIF_WSA-RX-0</back_end_name>
     <max_channels>2</max_channels>
     <channels>2</channels>
     <samplerate>48000</samplerate>
     <bit_width>16</bit_width>
     <snd_device_name>speaker</snd_device_name>
</out-device>
```

Customize usecasekvmanager XML file

The `Usecasekvmanager.xml` file has the GKV details for each use case. PAL uses this XML file to get the KV configuration for each use case and then uses that configuration to get graph information from the acdb files. This file is in the `/etc` folder on the device.

The following is an example of one stream and device graph key vector configuration.

Stream KV

```
<stream type="PAL_STREAM_LOW_LATENCY">
     <keys_and_values Direction="RX" Instance="1">
     <!-- STREAMRX - PCM_LL_PLAYBACK -->
```

```
     <graph_kv key="0xA1000000" value="0xA100000E"/>
     <!-- INSTANCE - INSTANCE_1 -->
     <graph_kv key="0xAB000000" value="0x1"/>
</keys_and_values>
```

Device KV

```
<!-- Speaker Device -->
<device id="PAL_DEVICE_OUT_SPEAKER">
     <keys_and_values>
     <!-- DEVICERX - SPEAKER -->
     <graph_kv key="0xA2000000" value="0xA2000001"/>
     </keys_and_values>
</device>
```

DevicePP KV

```
<!-- OUT Speaker DevicePPs -->
<devicepp id="PAL_DEVICE_OUT_SPEAKER">
<keys_and_values StreamType="PAL_STREAM_COMPRESSED,PAL_STREAM_LOW_
LATENCY">
     <!-- DEVICERX - SPEAKER -->
     <graph_kv key="0xA2000000" value="0xA2000001"/>
     <!-- DEVICEPP_RX - DEVICEPP_RX_AUDIO_MBDRC -->
     <graph_kv key="0xAC000000" value="0xAC000002"/>
</keys_and_values>
```

4.2 Enable TinyALSA-based applications

TinyALSA is a library that wraps the ALSA kernel interface into APIs that clients can invoke. It also gives a plug-in interface to emulate ALSA APIs.

Find TinyALSA source code at:

`build-qcom-wayland/workspace/sources/tinyalsa` and

`build-qcom-wayland/workspace/sources/tinycompress`.

The following figure shows the TinyALSA plug-in architecture.

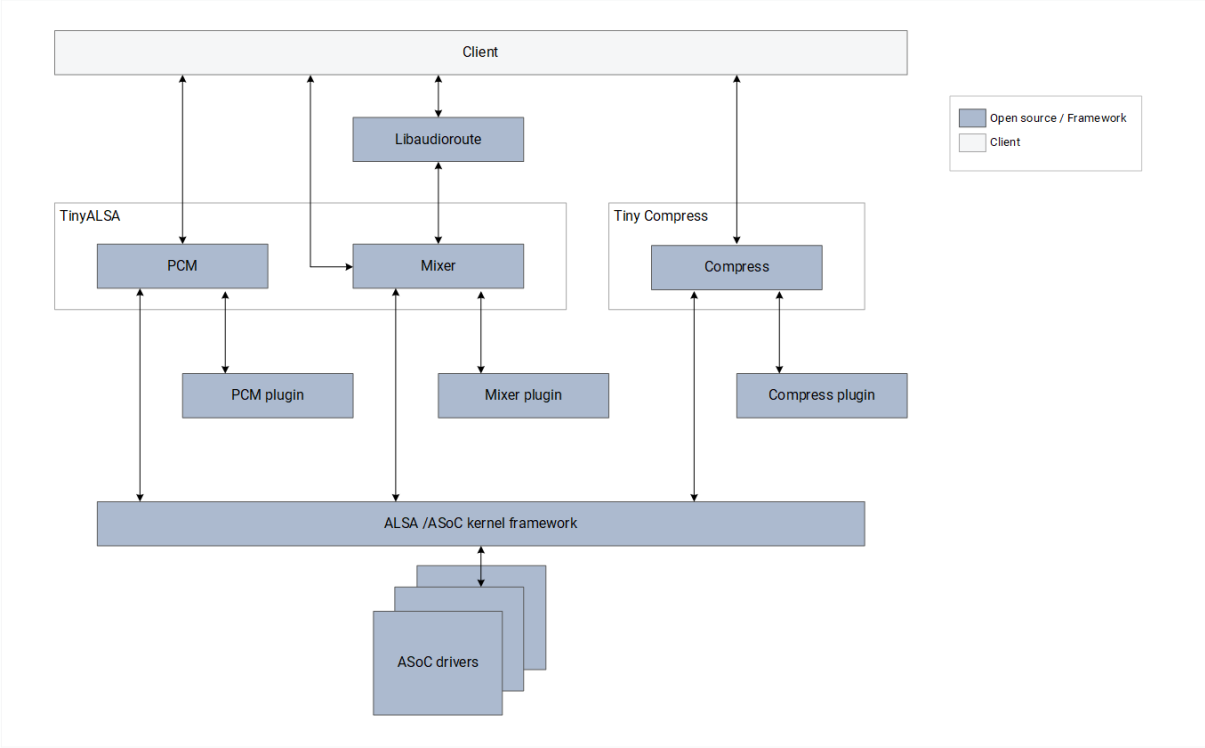


Figure1 TinyALSA plug-in architecture

The ALSA framework in the kernel exposes card and device nodes for PCM, compress, and mixer.

PCM, mixer, and compress plug-ins are TinyALSA plug-ins. They route all PCM, mixer, and compress calls from the application to plug-in specific implementation.

The plug-ins create a virtual sound card with PCM, compress, and mixer nodes. Virtual nodes map to on-device `.so` files (dynamically loadable shared objects).

The virtual sound card configuration is in the `card-defs.xml` file. This file is in the `/etc/` folder on the device.

TinyALSA APIs

The following are common TinyALSA APIs. See the open source TinyALSA documentation for a complete description of all APIs.

pcm_open

Opens a PCM audio device for input and output operations. Initializes a PCM device for communication, allowing read/write operations in audio data.

```
struct pcm *pcm_open(
     unsigned int card,
     unsigned int device,
     unsigned int flags,
     struct pcm_config *config)
```

Parameters

card	Card number.
device	Device number in the chosen card.
flags	Flags to configure the PCM device.
config	Structure variable that specifies audio stream parameters.

Return value

`pcm* handle`

pcm_is_ready

Checks if the PCM device is ready for input and output operations.

```
int pcm_is_ready(struct pcm *pcm)
```

Parameters

pcm	Pointer to the open PCM device.

Return value

- 0 on success
- Error code on failure

pcm_prepare

Prepares audio device input and output operations.

```
int pcm_prepare(
     struct pcm *pcm)
```

Parameters

pcm	Pointer to the open PCM device.

Return value

- 0 on success
- Error code on failure

pcm_start

Starts the PCM audio device for input and output operations.

```
int pcm_start(
     struct pcm *pcm)
```

Parameters

pcm	Pointer to the open PCM device.

Return value

- 0 on success
- Error code on failure

pcm_write

Writes audio data to the audio PCM device. Takes audio data as input and sends it to the PCM device for playback or processing.

```
int pcm_write(
     struct pcm *pcm,
     const void *data,
     unsigned int count)
```

Parameters

pcm	Pointer to the open PCM device.
data	Audio data to write.
count	Number of audio frames to write.

Return value

- 0 on success
- Error code on failure

pcm_read

Gets audio data from the PCM device, which allows the application to capture audio data from a microphone.

```
int pcm_read(
     struct pcm *pcm,
     void *data,
     unsigned int count)
```

Parameters

pcm	Pointer to the open PCM device.
data	Audio data to read.
count	Number of audio frames to read.

Return value

- 0 on success
- Error code on failure

pcm_stop

Stops the PCM audio device from further input and output operations.

```
int pcm_stop(
     struct pcm *pcm)
```

Parameters

pcm	Pointer to the open PCM device.

Return value

- 0 on success
- Error code on failure

pcm_close

Closes the PCM audio device. This releases the resources associated with the PCM device and releases the memory.

```
int pcm_close(
     struct pcm *pcm)
```

Parameters

pcm	Pointer to the open PCM device.

Return value

- 0 on success
- Error code on failure

Configure TinyALSA

For audio use cases from TinyALSA, configure the virtual mixer controls.

These controls, created by the mixer plug-in, set up audio use case graphs and modules. Most are byte array-based and are configured using the `mixer_ctl_set_array` API. Metadata (referred to as 'PCM100 metadata') is set using key-value (KV) pairs. For implementation details, see the `set_agm_audio_intf_metadata` API in `build-qcom-wayland/workspace/sources/qcom-agm/opensource/agm/plugins/tinyalsa/test/agmmixer.c`.

```
/**
            * Key Vector pair
            */
            struct agm_key_value {
            uint32_t key; /**< key */
            uint32_t value; /**< value */
            };
            /*Sample allocation for the key value pair*/
            gkv = calloc(num_gkv, sizeof(struct agm_key_value));
            ckv = calloc(num_ckv, sizeof(struct agm_key_value));
```

Sample code for the TinyALSA-based agmplay and agmcap utilities can be found at:

`build-qcom-wayland/workspace/sources/qcom-agm/opensource/agm/plugins/tinyalsa/test`.

To enable and execute audio use cases from TinyALSA:

1. Set the audio interface (backend) device configuration, including sample rate, channels, format, and data format.

   ```
   'CODEC_DMA-LPAIF_WSA-RX-0 rate ch fmt' 48000 2 2(PCM_16)
   ```

2. Set the metadata, including graph keys, cal keys for the device, and DevicePP.

   ```
   'CODEC_DMA-LPAIF_WSA-RX-0 metadata' bytes
   ```

3. Set the control to indicate that follow-on mixer configurations will set the metadata for the stream and StreamPP subgraphs. Zero indicates that subsequent commands are for the stream.

   ```
   'PCM100 control' Zero
                       'PCM100 metadata' bytes
   ```

4. Set the control to indicate that follow on mixer configurations will set the metadata for the DevicePP and stream-device subgraphs. `Codec_DMC-LPAIF_WSA-RX-0` indicates that the subsequent commands are for stream-device. `CODEC_DMA-LPAIF_WSA-RX-0` is one of the audio interfaces registered with the ALSA ASOC framework. A list of all audio interfaces can be found at `/proc/asound/pcm`.

   ```
   'PCM100 control' CODEC_DMA-LPAIF_WSA-RX-0
                       'PCM100 metadata' bytes
   ```

5. Retrieve all tags, module ID, and instance IDs associated with a given session between the stream and audio interface.

```
'PCM100 getTaggedInfo' bytes
```

6. Set the control to indicate that follow on mixer configurations will set the parameters for modules on the stream subgraph.

```
'PCM100 control' Zero
                    'PCM100 setParam' bytes
```

7. Set the control to indicate that follow on mixer configurations will set the parameters for modules on the StreamDevice and DevicePP subgraphs.

```
'PCM100 control' CODEC_DMA-LPAIF_WSA-RX-0
                    'PCM100 setParam' bytes
```

8. Connect the frontend (stream) with the backend (codec/audio interface).

```
'PCM100 connect'CODEC_DMA-LPAIF_WSA-RX-0
```

9. Open the PCM device.

```
pcm_open
```

10. Prepare the audio device for input and output operations.

```
pcm_prepare
```

11. Start the PCM audio device for input and output operations.

```
pcm_start
```

12. Write and read audio data from the audio PCM device.

```
pcm_write/pcm_read
```

13. Stop the PCM device.

```
pcm_stop
```

14. Close the PCM audio device.

```
pcm_close
```

All mixer controls for a virtual device can be fetched using:

```
ssh root@ip-addr
systemctl stop pulseaudio
```

```
tinymix set -D 100
```

4.3 Customize audio graph

Each audio use case is a graph with subgraphs of a specific type. Each subgraph has one or more functional software blocks (referred to as modules) that perform a specific function.

Audio graph terms

Audio graph terms

Use case	A graph of modules from source endpoint(s) to sink endpoint(s) that meets the product defined use case.
Graph	A logical interpretation of a group of one or more subgraphs connected together to create a specific use case.
Subgraph	A logical abstraction for a group of modules that connect and are manipulated as a single entity.
Container	Object that allows the system designer to group and run audio processing modules together in a single software thread.
Module	The smallest independent processing unit in the signal processing framework.
Key value (KV) pair	The individual key and associated values in a key vector. For example, a key can be a sound device and a value can be headphone, speaker, or some other sound device.
Key vector	Uniquely identifies a graph or subgraph through a set of KV pairs.
Graph key vector (GKV)	GKV is a unique identifier that gets a graph, which is represented by KV pairs. The graph or system designer associates a set of unique <keys> and <values> when creating a subgraph from the QACT UI canvas.
Calibration key vector (CKV)	CKV is a unique identifier that gets calibration data, which is represented by KV pairs. The graph or system designer associates a set of unique <keys> and <values> when storing calibration data.
Tag and tag key vector (TKV)	A tag is an identifier that sets runtime parameters for one or more modules. It allows updating module configurations (for example, enabling/disabling features like echo cancellation or equalization) in a graph at runtime.

Graph segments

An audio use case has the following segments.

The front-end represents stream and streamPP subgraphs, while the back-end represents the per-stream per-device (PSPD), devicePP, and device subgraphs.

Graph segments

Stream	Gives a data write/read interface and performs decoding and encoding of compressed data.
StreamPP	Has stream-based processing modules (for example, equalizer).
PSPD	Has a module to convert the stream media format to the device media format.
DevicePP	Has processing modules for sound device tuning.
Device	Hardware endpoint such as CodecDMA (SoundWire), I2S, or TDM port.

Once a front-end connects to a back-end using a routing mixer control, the full GKV forms by concatenating the subgraph GKVs and the CKVs assigned using mixer controls. Upon opening the front-end PCM or compress device, AGM invokes GSL APIs with concatenated GKVs and CKVs to set up the graph in SPF and apply calibration. At the same time, AGM opens a kernel PCM device corresponding to the connected back-ends to begin audio peripheral setup.

Sample audio graph

The following figure shows an example audio graph for a playback scenario.

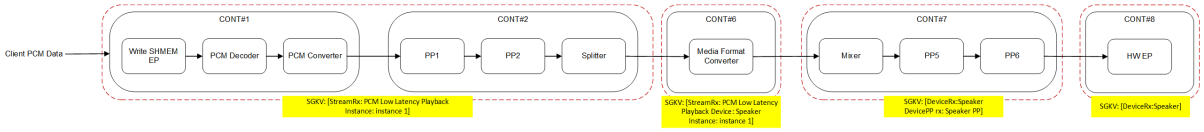


Figure2 Sample audio graph for playback

In this graph:

1. The stream subgraph has a write shared memory endpoint, PCM decoder, and PCM converter. The client passes PCM samples to write shared memory endpoint.
2. If conversion is necessary, the PCM converter converts PCM samples to a format supported by the stream-specific postprocessing modules.
3. Output of the stream subgraph is fed into the stream-device subgraph, which has the media format converter (MFC). MFC converts the stream-subgraph PCM to the device-subgraph PCM format.
4. After conversion, output of the stream-device subgraph is fed into the device PP subgraph for device-specific postprocessing. A mixer is placed at the beginning of subgraph to mix input streams.

5. Output of the devicePP subgraph is then fed into the device subgraph, which has a hardware endpoint module such as an I2S driver.

The following is the GKV for this example graph:

```
GKV1: <StreamRX1 KVs, StreamRX2 PP KVs, StreamRX1DeviceRX KVs,
DeviceRX PP KVs, DeviceRX KVs>

GKV2: <StreamRX2 KVs, StreamRX2 PP KVs, StreamRX2DeviceRX KVs,
DeviceRX PP KVs, DeviceRX KVs>
```

Audio graph manager

The audio graph manager (AGM) gives interfaces to allow TinyALSA-based mixer controls and PCM/compress plug-ins to interact and enable audio use cases. AGM runs as part of the PulseAudio service that runs in the user space.

AGM gives APIs for mixer plugins and PCM/compress APIs to set up audio use cases. It maintains many ALSA clients to set up use cases. AGM also manages front-end to back-end connections.

The following figure shows the AGM block at a high level.

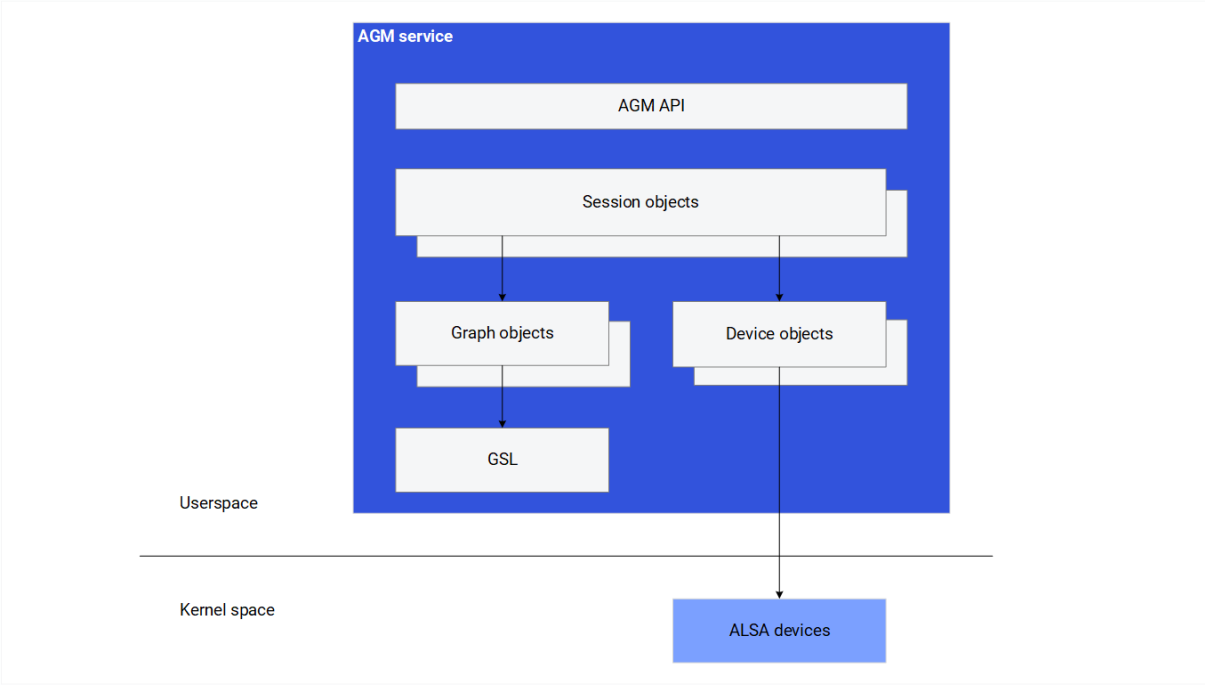


Figure3 High-level AGM software block

AGM objects

Object	Description
Session	• A session object is an audio playback or capture session. • Invoking session-specific mixer controls or APIs creates sessions. • Gives APIs for TinyALSA plug-ins to configure streams and manages state transitions of graph and device objects.
Graph	• A graph object is an audio use case. • Interacts with GSL to open, manage, and close graphs. • Gives APIs for graph creation, manages graphs, and configures stream and device endpoints.
Device	• A device object is an ALSA device from the ALSA sound card. • Enumerates available audio interfaces and gives device APIs for transitioning device states.

AudioReach graph services

The AudioReach™ Signal Processing Framework graph services (ARGS) consists of the graph service layer (GSL), generic packet router (GPR), and acdb management layer (AML). It handles initialization and creation of graphs, and creation of packets for sending series of commands to the SPF.

ARGS components

Component	Description
GSL	• The graph service layer (GSL) is a software driver for SPF which manages graphs, subgraphs, buffers, and configurations. • Loads and initializes graphs using graph key vectors (GKVs). • Handles data commands and SPF module calibration.
GPR	• The generic packet router (GPR) routes audio message packets across SPF and the graph service library. • Handles commands for constructing audio graphs and processing audio.
AML	• The acdb management layer (AML) gives get/set APIs to get and adjust data in acdb files. • Gives data abstractions and organization for how the audio driver and its components consume the calibration data.

Audio calibration database

acdb is a static database on the apps processor. It has all tuning/calibration parameters for the LPAI. The `*.acdb` file format organizes calibration data for various audio modules for various use cases.

Edit this file format using QACT (a PC tool) and place it on the device file system in the `/etc/acdbdata/` folder. During use case initialization or device switch, the AML queries the acdb database with a specified GKV and pushes the device calibration data to SPF.

Signal processing framework

Signal processing framework (SPF) runs in the LPAI subsystem and performs audio data processing.

The following figure gives a high-level overview of the functional blocks used in SPF.

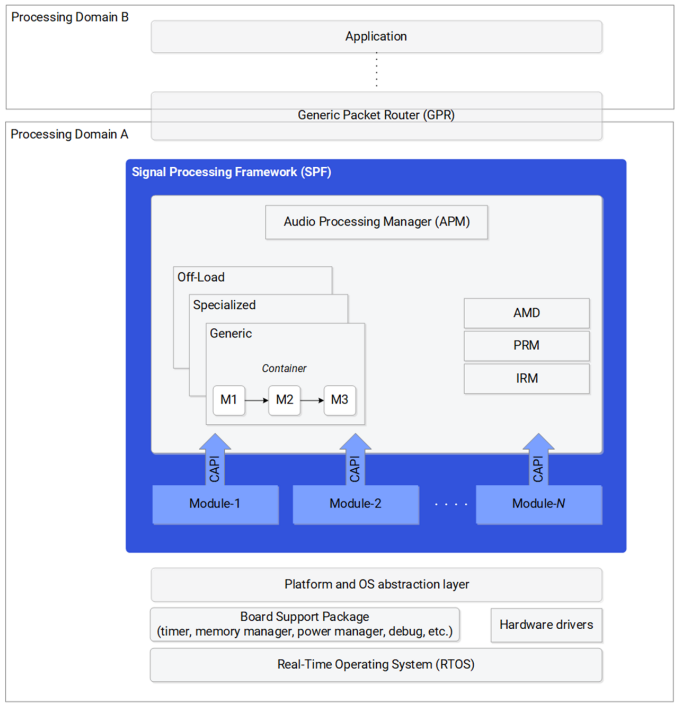


Figure4 High-level SPF software block

SPF components

Component	Description
APM	Audio processing manager (APM) sets up and manages the use case graphs in the SPF. It gives the standard APIs to the graph management library and APM client to set up and configure audio use cases.
Modules	A module is a functional block in the SPF. It performs real-time audio processing in the LPAI subsystem.
Containers	A container is a framework implementation that runs a group of data processing modules together in the same software thread. Each container runs in its own software thread.

5 Troubleshoot audio

This section gives commands for audio logging and debugging.

5.1 Capture logs

User space and kernel audio driver logs can help find issues.

User space logs	To capture user space logs:
	`ssh root@ip-addr`
	`mount -o remount,rw /`
	`cat /var/log/user.log`
	See User space logs for more instructions.

Kernel audio driver logs	To capture kernel logs:
	`ssh root@ip-addr dmesg`
	To disable kernel logs in a specific file:
	`echo -n "file <filename> -p" > /sys/kernel/debug/dynamic_debug/control`

Dynamic kernel logs	Dynamic logging is disabled by default. To enable it, add the `CONFIG_DYNAMIC_DEBUG` kernel configuration, recompile, and re-flash the device.
	To enable audio dynamic kernel logs:
	`ssh root@ip-addr`
	`mount -o rw,remount /`
	`mount -t debugfs none /sys/kernel/debug`
	`echo -n "file <filename> +p" > /sys/kernel/debug/dynamic_debug/control`
	See Enable dynamic debug for more instructions.

5.2 Analyze logs

Analyze user space and kernel audio driver logs for playback and record use cases.

Playback logs

The following log snippet shows the information collected for the playback use case.

```
//Open Low latency Playback stream. Details of each stream type can
be found at sources/audio/opensource/arpal-lx/inc/PalDefs.h
2022-04-28T18:02:08.748280+00:00 pulseaudio: pal_stream_open: 224:
Enter, stream type:1

//Verify the backend device, sample rate, bitwidth, channels etc
2022-04-28T18:02:08.748627+00:00 pulseaudio: setDeviceMediaConfig:
1056: CODEC_DMA-LPAIF_WSA-RX-0 rate ch fmt data_fmt 48000 2 2 1

//Start playback stream
2022-04-28T18:02:08.751947+00:00 pulseaudio: pal_stream_start: 338:
Enter. Stream handle 0xffff94001040K

//Map the metadata with kvh2xml.h file for playback usecase details.
2022-04-28T18:02:08.853157+00:00 pulseaudio: metadata_print: 82 key:
0xa1000000, value:0xa100000e//PCM_LL_PLAYBACK
2022-04-28T18:02:08.853395+00:00 pulseaudio: metadata_print: 82 key:
0xab000000, value:0x1
2022-04-28T18:02:08.853660+00:00 pulseaudio: metadata_print: 82 key:
0xa2000000, value:0xa2000001//Speaker
2022-04-28T18:02:08.853881+00:00 pulseaudio: metadata_print: 82 key:
0xac000000, value:0xac000002//DEVICEPP_RX_AUDIO_MBDRC

//Verify the graph opened for playback usecase
2022-04-28T18:02:08.856934+00:00 pulseaudio: print_graph_alias: 2334
GKV Alias 142 | StreamRX_PCM_LL_Playback_DeviceRX_Speaker_Instance_
Instance_1_DevicePP_Rx_Audio_MBDRC
//graph_open called
2022-04-28T18:02:08.859509+00:00 pulseaudio: graph_open: 709 graph_
handle 0x47534c

//Configure hardware endpoint module
2022-04-28T18:02:08.864386+00:00 pulseaudio: configure_hw_ep_media_
config: 636 entry mod tag c0000004 miid 43b1 mid 7001023
2022-04-28T18:02:08.864495+00:00 pulseaudio: configure_hw_ep_media_
config: 664 rate 48000 bw 16 ch 2, data_fmt 1
2022-04-28T18:02:08.864603+00:00 pulseaudio: configure_hw_ep_media_
```

```
config: 676 exit, ret 0

//graph_start entry
2022-04-28T18:02:08.867234+00:00 pulseaudio: graph_start: 899 entry
graph_handle 0x47534c
//Stream started
2022-04-28T18:02:08.867864+00:00 pulseaudio: pal_stream_start: 387:
Exit. status 0

//graph_stop entry
2022-04-28T18:02:25.037338+00:00 pulseaudio: graph_stop: 928 entry
graph_handle 0x47534c
//Stop the PAL stream once playback completes
2022-04-28T18:02:25.039923+00:00 pulseaudio: pal_stream_stop: 441:
Exit. status 0

//graph_close entry
2022-04-28T18:02:25.050944+00:00 pulseaudio: graph_close: 762 entry
handle 0x47534c
//Close the PAL stream
2022-04-28T18:02:25.054510+00:00 pulseaudio: pal_stream_close: 322:
Exit. status 0
```

Record logs

The following log snippet shows the information collected for the record use case.

```
//Open Recording stream for PAL_STREAM_RAW. Details of stream type
can be found at sources/audio/opensource/arpal-lx/inc/PalDefs.h
Apr 29 09:23:11 pulseaudio[862]: pal_stream_open: 224: Enter, stream
type:9

//Verify the backend device, sample rate, bitwidth, channels etc
Apr 29 09:23:11 pulseaudio[862]: setDeviceMediaConfig: 1056: CODEC_
DMA-LPAIF_VA-TX-0 rate ch fmt data_fmt 48000 1 2 1

//Start recording stream
Apr 29 09:23:11 pulseaudio[862]: pal_stream_start: 338: Enter. Stream
handle 0xffff6c001040K

//graph_open entry
Apr 29 09:23:11 pulseaudio[862]: graph_open: 709 graph_handle
0x47534c
```

```
//Metadata details to identify the usecase
Apr 29 09:23:11 pulseaudio[862]: metadata_print: 82 key:0xb1000000,
value:0xb1000009//RAW_RECORD
Apr 29 09:23:11 pulseaudio[862]: metadata_print: 82 key:0xa3000000,
value:0xa3000004//HANDSETMIC

//Verify the graph opened for recording usecase
Apr 29 09:23:11 pulseaudio[862]: print_graph_alias: 2334 GKV Alias 29
| DeviceTX_Handset_Mic_StreamTX_RAW_Record

//Configure hardware endpoint module
Apr 29 09:23:11 pulseaudio[862]: configure_hw_ep_media_config: 636
entry mod tag c0000005 miid 43af mid 7001024
Apr 29 09:23:11 pulseaudio[862]: configure_hw_ep_media_config: 664
rate 48000 bw 16 ch 1, data_fmt 1
Apr 29 09:23:11 pulseaudio[862]: configure_hw_ep_media_config: 676
exit, ret 0

//graph_start entry
Apr 29 09:23:11 pulseaudio[862]: graph_start: 899 entry graph_handle
0x47534c
//Stream recording started
Apr 29 09:23:11 pulseaudio[862]: pal_stream_start: 387: Exit. status
0


//graph_stop entry
Apr 29 09:23:26 pulseaudio[862]: graph_stop: 928 entry graph_handle
0x47534c
//Stop the PAL stream once user stops recording
Apr 29 09:23:26 pulseaudio[862]: D: [regular2] pal-source.c: pal_
stream_stop returned 0

//Close the PAL stream
Apr 29 09:23:26 pulseaudio[862]: pal_stream_close: 284: Enter. Stream
handle :0xffff6c001040K
//graph_close entry
Apr 29 09:23:26 pulseaudio[862]: graph_close: 762 entry handle
0x47534c
//Close the PAL stream
Apr 29 09:23:26 pulseaudio[862]: pal_stream_close: 322: Exit. status
0
```

6 Advanced audio features

QCS6490

6.1 Minimize echo and noise

Echo and noise problems are common in VoIP systems. Speech comes from the far-end speaker and echoes back with a time delay, causing perception problems. Echo cancellation decreases the echo from the far-end speaker during communication. Noise suppression decreases the noise from the microphone channel. The Fluence echo cancellation and noise suppression (ECNS) algorithm provides stationary and nonstationary noise suppression and echo cancellation. Acoustic echo is when echoes occur due to the acoustic path (acoustic coupling) between the loudspeaker and microphone of a device. It is important for hands-free and teleconferencing applications.

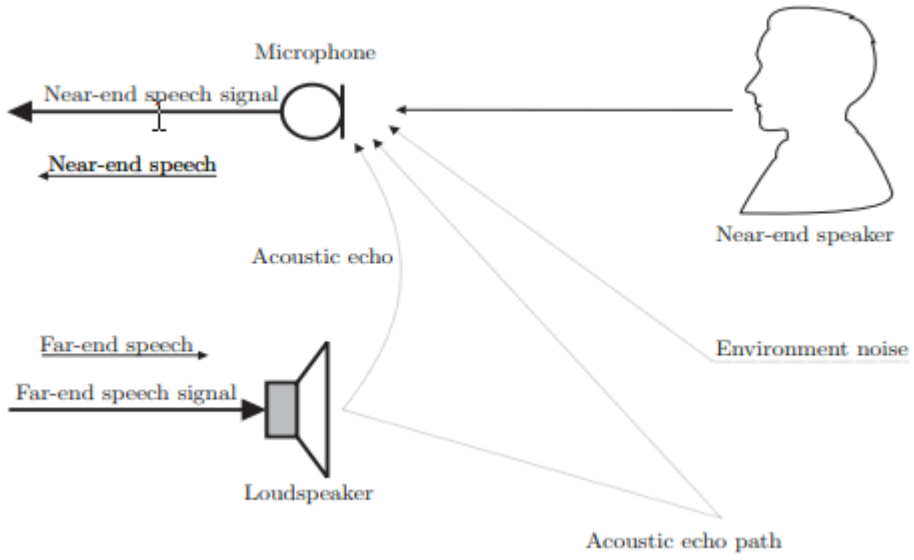


Figure1 Noise and acoustic echo

Echo Canceller – An adaptive filter that self-adjusts coefficients to cancel out echo. Every echo has an echo path, and is characterized by an impulse response. The echo

canceller adapts to the network echo path such that it cancels out the echo.

- **Noise Suppression** – Single mic echo canceller and noise suppressor (SMECNS) helps to suppress the surrounding stationary noise when using devices in noisy locations.

Enable SMECNS for recording

When recording, Fluence keeps speech quality in the recording path by suppressing background noise captured by the microphone.

For single-microphone recordings, only stationary noise suppression is possible. Stationary noise is where the frequency does not change over time, for example, road noise or white noise.

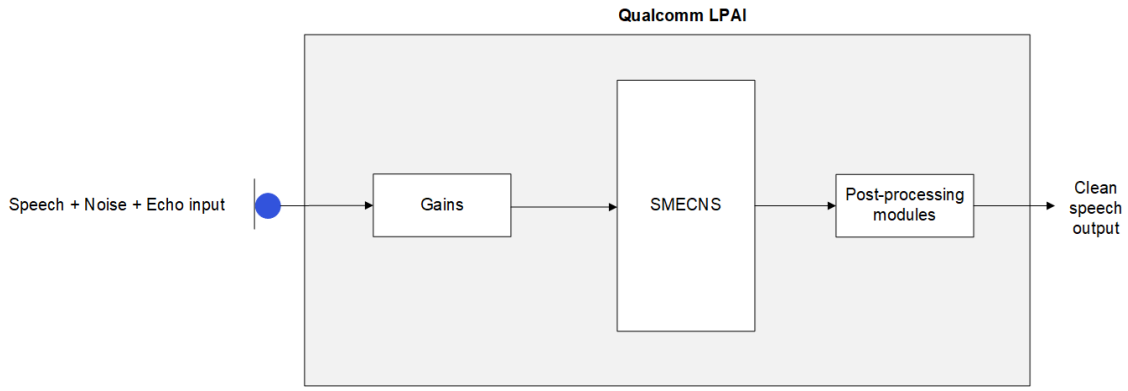


Figure2 SMECNS software block for recording

To enable the SMECNS in the recording path:

```
pactl set-source-port regular0 speaker-mic
parec --rate=48000 --format=s16le --channels=1 --file-format=wav
/opt/rec3.wav --device=regular0
```

Enable SMECNS for VoIP

Fluence reduces noise and eliminates echo in VoIP communication. It also suppresses noise and acoustic echo on the microphone signal.

The SDK supports a PulseAudio VoIP source and sink, which you can use when developing applications.

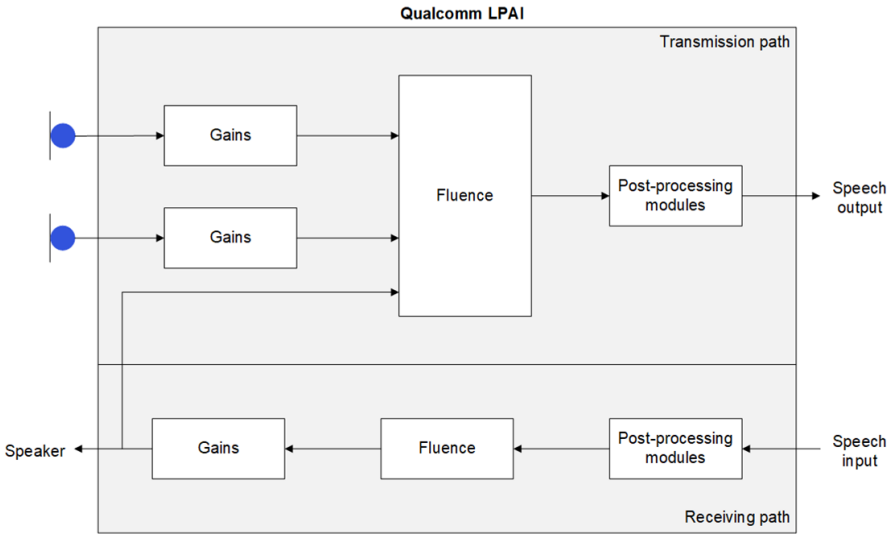


Figure3 SMECNS software block for VoIP

To enable the SMECNS in the VoIP path:

Set record source	`pactl set-source-port voip-tx0 speaker-mic` `parec --rate=48000 --format=s16le --channels=1 --file-format=wav /opt/rec3.wav --device=voip-tx0`
Set playback sink	`paplay /opt/<FileName>.wav --device=voip-rx0`

Note: Be sure to push a PCM file (`<FileName>.wav`) to the `/opt/` folder.

6.2 Compress offload

Compress offload allows the decoding of compressed audio streams in LPAI, removing the need for decoding in the Application processor. This lowers overall power usage relative to processing based on the Application processor.

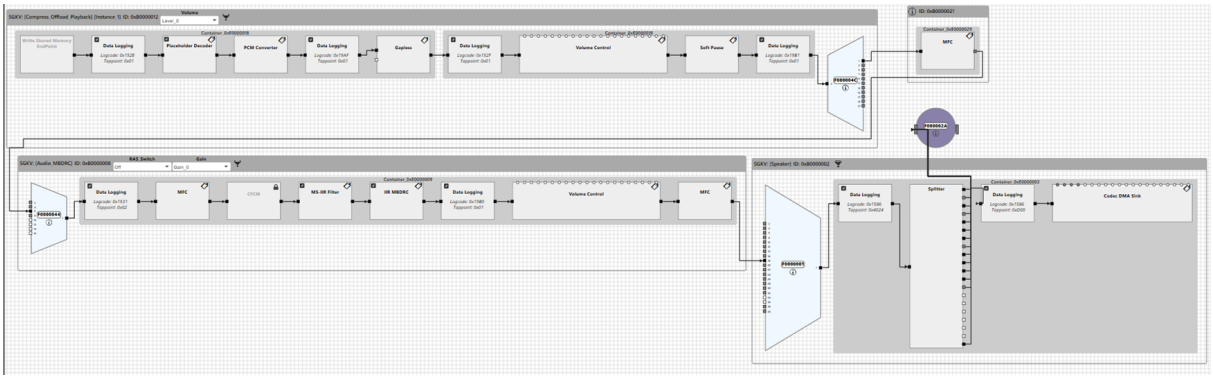


Figure4 Compress offload playback graph

Validate compression

Push an MP3 file using the following command:

```
scp test.mp3 root@[ip-addr]:/opt/
```

This example plays the `/opt/test.mp3` audio file with all default configurations in the compress offload path.

Validate audio playback through PulseAudio

```
#paplay -d offload0 --encoding=mpeg --raw /opt/test.mp3
```

6.3 Enable A2DP

The Advanced Audio Distribution Profile (A2DP) defines the protocols and procedures that control high-quality audio content sent over a Bluetooth® channel either in mono or stereo.

The QCS6490 chipset can be used as an A2DP Sink or A2DP Source.

A2DP Source

The A2DP Source device encodes and transmits digital audio streams over Bluetooth to a receiving device.

The following figure shows the data flow for an A2DP Source:

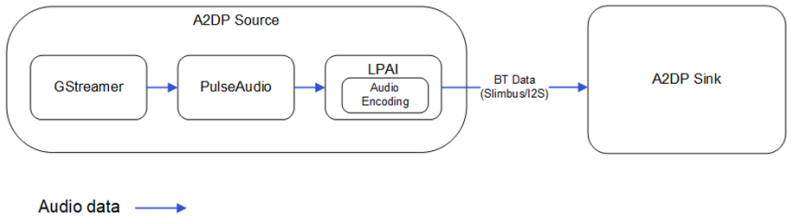


Figure5 A2DP Source data flow

The following is an acdb graph for an A2DP Source:

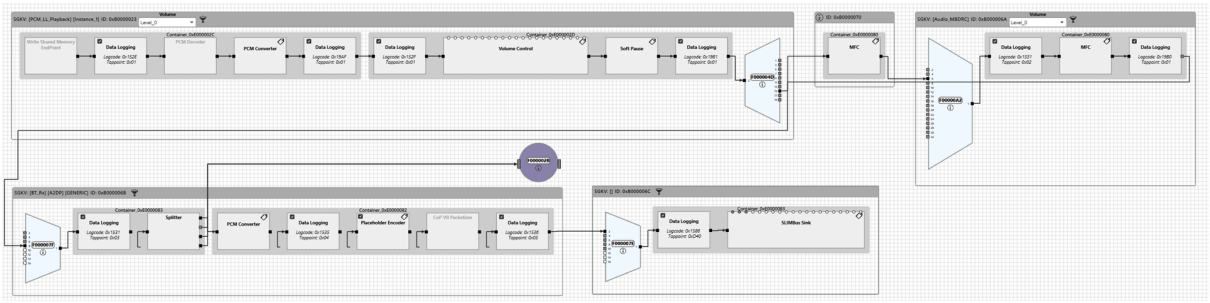


Figure6 A2DP Source acdb graph

A2DP Sink

An A2DP Sink is a device that receives an audio stream from a Source device, decodes the audio data, and plays it back.

The following figure shows the data flow for an A2DP Sink:

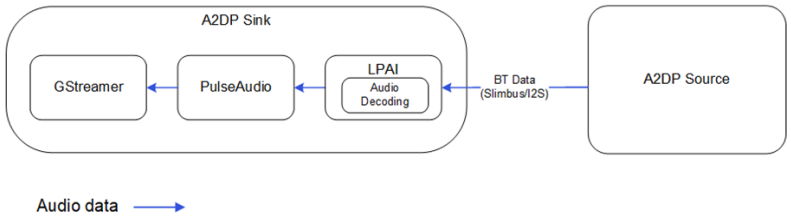


Figure7 A2DP Sink data flow

The following is an acdb graph for an A2DP Sink:

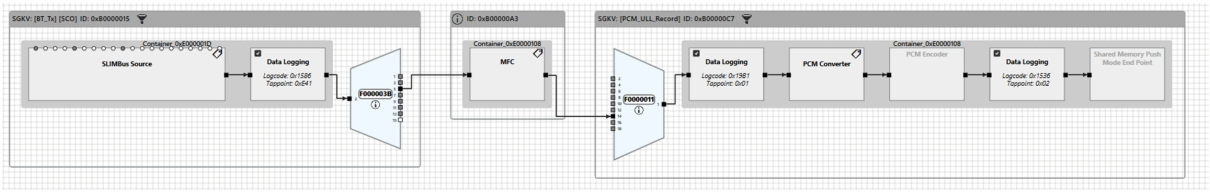


Figure8 A2DP Sink acdb graph

Validate A2DP

To validate the A2DP Sink and A2DP Source, see Qualcomm Linux Bluetooth Guide.

QCS9075/QCS8275

6.4 Minimize echo and noise

Echo and noise problems are common in VoIP systems. Speech comes from the far-end speaker and echoes back with a time delay, causing perception problems. Echo cancellation decreases the echo from the far-end speaker during communication. Noise suppression decreases the noise from the microphone channel. The Fluence echo cancellation and noise suppression (ECNS) algorithm provides stationary and nonstationary noise suppression and echo cancellation. Acoustic echo is when echoes occur due to the acoustic path (acoustic coupling) between the loudspeaker and microphone of a device. It is important for hands-free and teleconferencing applications.

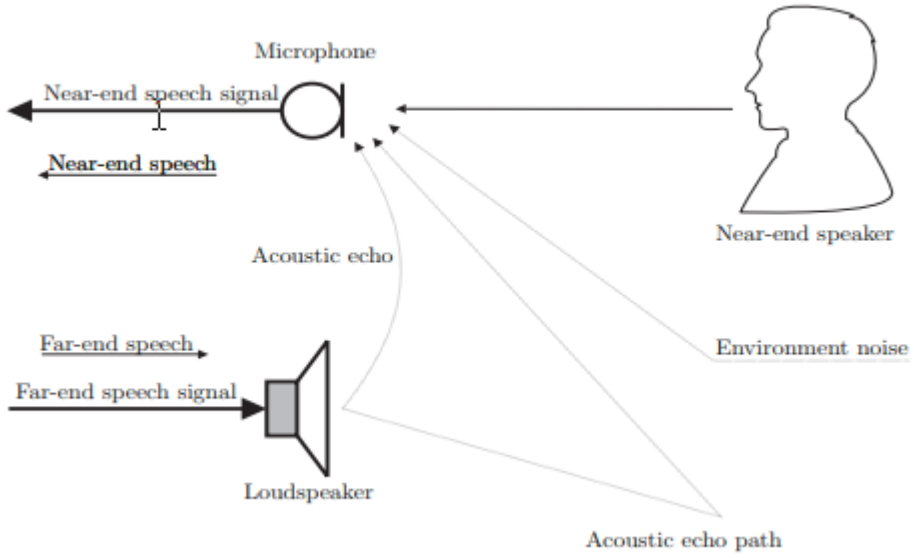


Figure9 Noise and acoustic echo

Echo Canceller – An adaptive filter that self-adjusts coefficients to cancel out echo. Every echo has an echo path, and is characterized by an impulse response. The echo canceller adapts to the network echo path such that it cancels out the echo.

- **Noise Suppression** – SMECNS helps to suppress the surrounding stationary noise when using devices in noisy locations.

Enable SMECNS for recording

When recording, Fluence keeps speech quality in the recording path by suppressing background noise captured by the microphone.

For single-microphone recordings, only stationary noise suppression is possible. Stationary noise is where the frequency does not change over time, for example, road noise or white noise.

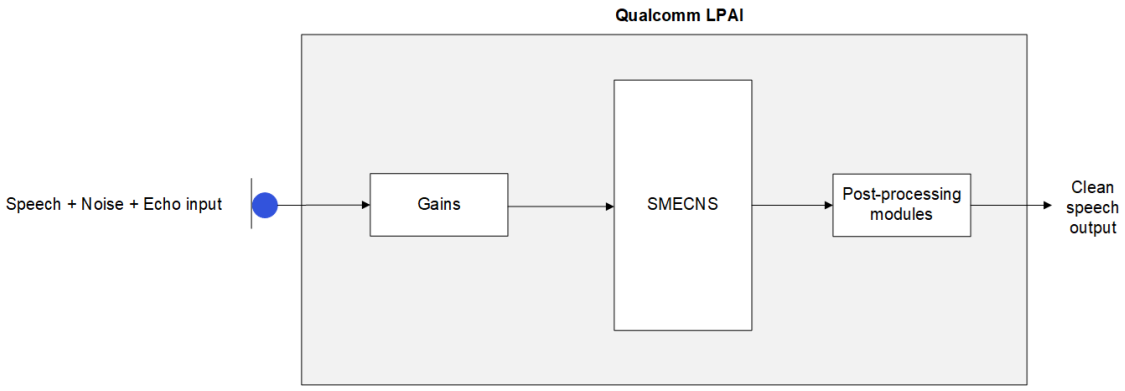


Figure10 SMECNS software block for recording

To enable the SMECNS in the recording path:

```
pactl set-source-port regular0 speaker-mic
parec --rate=48000 --format=s16le --channels=1 --file-format=wav
/opt/rec3.wav --device=regular0
```

Enable SMECNS for VoIP

Fluence reduces noise and eliminates echo in VoIP communication. It also suppresses noise and acoustic echo on the microphone signal.

The SDK supports a PulseAudio VoIP source and sink, which you can use when developing applications.

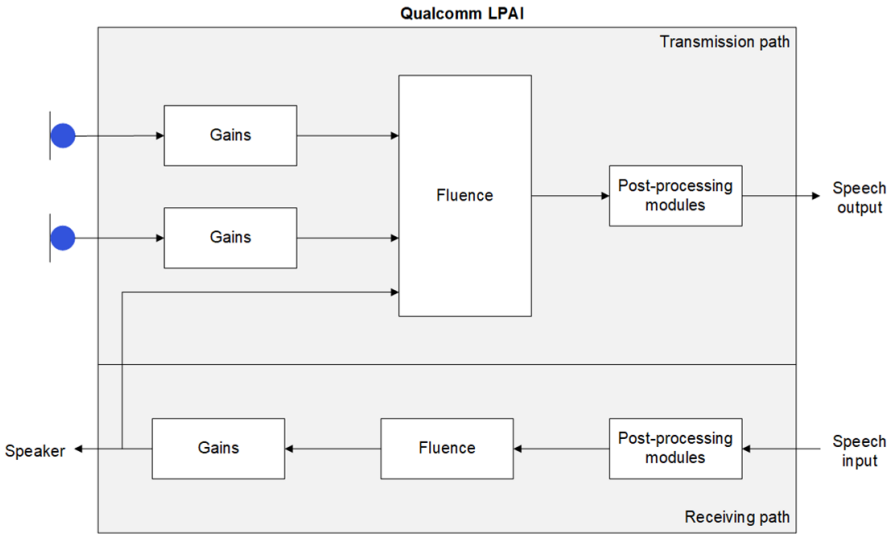


Figure11 SMECNS software block for VoIP

To enable the SMECNS in the VoIP path:

Set record source	`pactl set-source-port voip-tx0 speaker-mic` `parec --rate=48000 --format=s16le --channels=1 --file-format=wav /opt/rec3.wav --device=voip-tx0`
Set playback sink	`paplay /opt/<FileName>.wav --device=voip-rx0`

Note: Be sure to push a PCM file (`<FileName>.wav`) to the `/opt/` folder.

6.5 Compress offload

Compress offload allows the decoding of compressed audio streams in LPAI, removing the need for decoding in the Application processor. This lowers overall power usage relative to processing based on the Application processor.

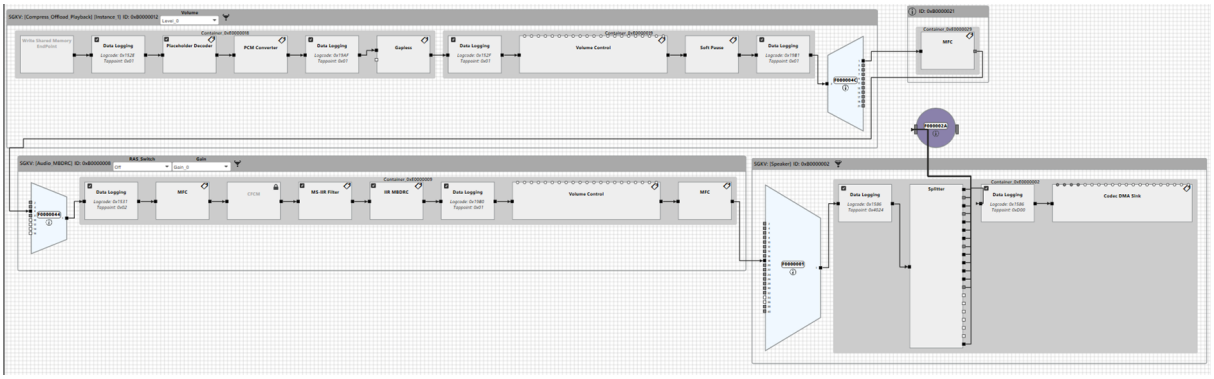


Figure12 Compress offload playback graph

Validate compression

Push an MP3 file using the following command:

```
scp test.mp3 root@[ip-addr]:/opt/
```

This example plays the `/opt/test.mp3` audio file with all default configurations in the compress offload path.

Validate audio playback through PulseAudio

```
#paplay -d offload0 --encoding=mpeg --raw /opt/test.mp3
```

Note: Make sure to use `dac_mer_testapp` before starting playback and record, as mentioned in Enable audio.

7 References

7.1 Related documents

Title	Resource
TinyALSA	https://github.com/tinyalsa/tinyalsa/tree/master
PulseAudio	https://www.freedesktop.org/software/pulseaudio/doxygen/index.html
AudioReach GitHub	https://github.com/audioreach
AudioReach documentation	https://audioreach.github.io/

7.2 Acronyms and terms

Acronym or Term	Definition
ACDB	Audio calibration database
AGM	Audio graph manager
ALSA	Advanced Linux Sound Architecture
Application DSP	Application digital signal processor
ARGS	AudioReach graph services
CKV	Calibration key vector
KV pair	Key Value pair
LPAI	Low power AI
PAL	Platform abstraction layer
PSPD	Per stream per device
QACT	Qualcomm audio calibration tool
SPF	Signal processing framework
TKV	Tag key vector

LEGAL INFORMATION

Your access to and use of this material, along with any documents, software, specifications, reference board files, drawings, diagnostics and other information contained herein (collectively this "Material"), is subject to your (including the corporation or other legal entity you represent, collectively "You" or "Your") acceptance of the terms and conditions ("Terms of Use") set forth below. If You do not agree to these Terms of Use, you may not use this Material and shall immediately destroy any copy thereof.

1) Legal Notice.

This Material is being made available to You solely for Your internal use with those products and service offerings of Qualcomm Technologies, Inc. ("**Qualcomm Technologies**"), its affiliates and/or licensors described in this Material, and shall not be used for any other purposes. If this Material is marked as "**Qualcomm Internal Use Only**", no license is granted to You herein, and You must immediately (a) destroy or return this Material to Qualcomm Technologies, and (b) report Your receipt of this Material to qualcomm.support@qti.qualcomm.com. This Material may not be altered, edited, or modified in any way without Qualcomm Technologies' prior written approval, nor may it be used for any machine learning or artificial intelligence development purpose which results, whether directly or indirectly, in the creation or development of an automated device, program, tool, algorithm, process, methodology, product and/or other output. Unauthorized use or disclosure of this Material or the information contained herein is strictly prohibited, and You agree to indemnify Qualcomm Technologies, its affiliates and licensors for any damages or losses suffered by Qualcomm Technologies, its affiliates and/or licensors for any such unauthorized uses or disclosures of this Material, in whole or part.

Qualcomm Technologies, its affiliates and/or licensors retain all rights and ownership in and to this Material. No license to any trademark, patent, copyright, mask work protection right or any other intellectual property right is either granted or implied by this Material or any information disclosed herein, including, but not limited to, any license to make, use, import or sell any product, service or technology offering embodying any of the information in this Material.

THIS MATERIAL IS BEING PROVIDED "AS IS" WITHOUT WARRANTY OF ANY KIND, WHETHER EXPRESSED, IMPLIED, STATUTORY OR OTHERWISE. TO THE MAXIMUM EXTENT PERMITTED BY LAW, QUALCOMM TECHNOLOGIES, ITS AFFILIATES AND/OR LICENSORS SPECIFICALLY DISCLAIM ALL WARRANTIES OF TITLE, MERCHANTABILITY, NON-INFRINGEMENT, FITNESS FOR A PARTICULAR PURPOSE, SATISFACTORY QUALITY, COMPLETENESS OR ACCURACY, AND ALL WARRANTIES ARISING OUT OF TRADE USAGE OR OUT OF A COURSE OF DEALING OR COURSE OF PERFORMANCE. MOREOVER, NEITHER QUALCOMM TECHNOLOGIES, NOR ANY OF ITS AFFILIATES AND/OR LICENSORS, SHALL BE LIABLE TO YOU OR ANY THIRD PARTY FOR ANY EXPENSES, LOSSES, USE, OR ACTIONS HOWSOEVER INCURRED OR UNDERTAKEN BY YOU IN RELIANCE ON THIS MATERIAL.

Certain product kits, tools and other items referenced in this Material may require You to accept additional terms and conditions before accessing or using those items.

Technical data specified in this Material may be subject to U.S. and other applicable export control laws. Transmission contrary to U.S. and any other applicable law is strictly prohibited.

Nothing in this Material is an offer to sell any of the components or devices referenced herein.

This Material is subject to change without further notification.

In the event of a conflict between these Terms of Use and the *Website Terms of Use* on www.qualcomm.com, the *Qualcomm Privacy Policy* referenced on www.qualcomm.com, or other legal statements or notices found on prior pages of the Material, these Terms of Use will control. In the event of a conflict between these Terms of Use and any other agreement (written or click-through, including, without limitation any non-disclosure agreement) executed by You and Qualcomm Technologies or a Qualcomm Technologies affiliate and/or licensor with respect to Your access to and use of this Material, the other agreement will control.

These Terms of Use shall be governed by and construed and enforced in accordance with the laws of the State of California, excluding the U.N. Convention on International Sale of Goods, without regard to conflict of laws principles. Any dispute, claim or controversy arising out of or relating to these Terms of Use, or the breach or validity hereof, shall be adjudicated only by a court of competent jurisdiction in the county of San Diego, State of California, and You hereby consent to the personal jurisdiction of such courts for that purpose.

2) Trademark and Product Attribution Statements.

Qualcomm is a trademark or registered trademark of Qualcomm Incorporated. Arm is a registered trademark of Arm Limited (or its subsidiaries) in the U.S. and/or elsewhere. The Bluetooth® word mark is a registered trademark owned by Bluetooth SIG, Inc. Other product and brand names referenced in this Material may be trademarks or registered trademarks of their respective owners.

Snapdragon and Qualcomm branded products referenced in this Material are products of Qualcomm Technologies, Inc. and/or its subsidiaries. Qualcomm patented technologies are licensed by Qualcomm Incorporated.