**Table of Contents**

[1. Overview 5](#_Toc530588276)

[1.1. Device Tree 8](#_Toc530588277)

[2. Terminologies 13](#_Toc530588278)

[3. API Specification 14](#_Toc530588279)

[3.1. APIs for PCM Interface 14](#_Toc530588280)

[3.2. APIs for hardware control 15](#_Toc530588281)

[3.3 APIs for for ASoC Platform interface 16](#_Toc530588282)

[3.4 APIs for for ASoC Platform driver 16](#_Toc530588283)

[3.5 Detail of APIs for PCM Interface 17](#_Toc530588284)

[3.5.1 snd\_adsp\_pcm\_open 17](#_Toc530588285)

[3.5.2 snd\_adsp\_pcm\_close 17](#_Toc530588286)

[3.5.3 snd\_adsp\_pcm\_hw\_params 17](#_Toc530588287)

[3.5.4 snd\_adsp\_pcm\_hw\_free 18](#_Toc530588288)

[3.5.5 snd\_adsp\_pcm\_prepare 18](#_Toc530588289)

[3.5.6 snd\_adsp\_pcm\_trigger 18](#_Toc530588290)

[3.5.7 snd\_adsp\_pcm\_ack 19](#_Toc530588291)

[3.5.8 snd\_adsp\_pcm\_pointer 19](#_Toc530588292)

[3.6 Detail of APIs for hardware control 20](#_Toc530588293)

[3.6.1 snd\_adsp\_control\_volume\_info 20](#_Toc530588294)

[3.6.2 snd\_adsp\_control\_sample\_rate\_info 20](#_Toc530588295)

[3.6.3 snd\_adsp\_control\_eqz\_switch\_info 20](#_Toc530588296)

[3.6.4 snd\_adsp\_control\_eqz\_info 20](#_Toc530588297)

[3.6.5 snd\_adsp\_control\_rdr\_out\_channel\_info 21](#_Toc530588298)

[3.6.6 snd\_adsp\_control\_volume\_get 21](#_Toc530588299)

[3.6.7 snd\_adsp\_control\_sample\_rate\_get 21](#_Toc530588300)

[3.6.8 snd\_adsp\_control\_eqz\_get 21](#_Toc530588301)

[3.6.9 snd\_adsp\_control\_eqz\_switch\_get 22](#_Toc530588302)

[3.6.10 snd\_adsp\_control\_rdr\_out\_channel\_get 22](#_Toc530588303)

[3.6.11 snd\_adsp\_control\_volume\_put 22](#_Toc530588304)

[3.6.12 snd\_adsp\_control\_sample\_rate\_put 22](#_Toc530588305)

[3.6.13 snd\_adsp\_control\_eqz\_put 23](#_Toc530588306)

[3.6.14 snd\_adsp\_control\_eqz\_switch\_put 23](#_Toc530588307)

[3.6.15 snd\_adsp\_control\_rdr\_out\_channel\_put 23](#_Toc530588308)

[3.7 Detail of APIs for ASoC Platform interface 24](#_Toc530588309)

[3.7.1 snd\_adsp\_pcm\_new 24](#_Toc530588310)

[3.8 Detail of APIs for ASoC Platform driver 24](#_Toc530588311)

[3.8.1 snd\_adsp\_probe 24](#_Toc530588312)

[3.8.2 snd\_adsp\_remove 24](#_Toc530588313)

[4 Sequence diagram 25](#_Toc530588314)

[4.1 Playback/TDM playback streams flow 25](#_Toc530588315)

[4.1.1 Open a renderer/TDM renderer substream 25](#_Toc530588316)

[4.1.2 Write data flow 26](#_Toc530588317)

[4.1.3 Close a playback/TDM playback substream 27](#_Toc530588318)

[4.2 Capture/TDM capture streams flow 28](#_Toc530588319)

[4.2.1 Open a capture/TDM capture substream 28](#_Toc530588320)

[4.2.2 Read data flow 29](#_Toc530588321)

[4.2.3 Close a capture/TDM capture substream 30](#_Toc530588322)

[4.3 Volume Control Flow 31](#_Toc530588323)

[4.3.1 Get volume value from hardware 31](#_Toc530588324)

[4.3.2 Set volume value to hardware 32](#_Toc530588325)

[4.4 Sample Rate Converter Control Flow 33](#_Toc530588326)

[4.4.1 Get sample rate of hardware 33](#_Toc530588327)

[4.4.2 Set sample rate of hardware 34](#_Toc530588328)

[4.5 Output Channel Control Flow 35](#_Toc530588329)

[4.5.1 Get output channel of Renderer 35](#_Toc530588330)

[4.5.2 Set output channel of Renderer 36](#_Toc530588331)

[4.6 Equalizer Control Flow 37](#_Toc530588332)

[4.6.1 Get status of Equalizer control 37](#_Toc530588333)

[4.6.2 Enable Equalizer control 38](#_Toc530588334)

[4.6.3 Get Equalizer parameters of hardware 39](#_Toc530588335)

[4.6.4 Set Equalizer parameters of hardware 40](#_Toc530588336)

[4.7 MIX Control Flow 42](#_Toc530588337)

[5. List of Usage 43](#_Toc530588338)

[5.1. Playback 43](#_Toc530588339)

[5.1.1. Playback stream 43](#_Toc530588340)

[5.1.2. Setting Playback Volume 44](#_Toc530588341)

[5.1.3. Setting Output Sample Rate 44](#_Toc530588342)

[5.1.4. Setting Output Channel 45](#_Toc530588343)

[5.1.5. MIX function 45](#_Toc530588344)

[5.2. Capture 48](#_Toc530588345)

[5.2.1. Capture stream 48](#_Toc530588346)

[5.2.2. Setting Capture Volume 48](#_Toc530588347)

[5.2.3. Setting Input Sample Rate 49](#_Toc530588348)

[5.3. TDM Playback 49](#_Toc530588349)

[5.3.1. TDM Playback stream 49](#_Toc530588350)

[5.3.2. Setting TDM Playback Volume 50](#_Toc530588351)

[5.3.3. Setting TDM Output Sample Rate 50](#_Toc530588352)

[5.4. TDM Capture 51](#_Toc530588353)

[5.4.1. TDM Capture stream 51](#_Toc530588354)

[5.4.2. Setting TDM Capture Volume 52](#_Toc530588355)

[5.4.3. Setting TDM Input Sample Rate 52](#_Toc530588356)

[5.5. Equalizer 53](#_Toc530588357)

[5.5.1. Equalizer for Playback 54](#_Toc530588358)

[5.5.2. Equalizer for Capture 56](#_Toc530588359)

[6 Appendix 59](#_Toc530588360)

[6.1 Error code 59](#_Toc530588361)

[6.2 Structure and type definitions 60](#_Toc530588362)

[6.2.1 snd\_adsp\_control structure 61](#_Toc530588363)

[6.2.2 snd\_adsp\_base\_info structure 62](#_Toc530588364)

[6.2.3 snd\_adsp\_playback structure 62](#_Toc530588365)

[6.2.4 snd\_adsp\_record structure 63](#_Toc530588366)

[6.2.5 snd\_adsp\_card structure 63](#_Toc530588367)

[6.2.6 snd\_adsp\_tdm\_playback structure 63](#_Toc530588368)

[6.2.7 snd\_adsp\_tdm\_record structure 63](#_Toc530588369)

**List of Figures**

[Figure 1‑1 Overview architecture of ADSP driver. 6](#_Toc530588370)

[Figure 4‑1 Open flow for playback/TDM playback stream 25](#_Toc530588371)

[Figure 4‑2 Write data flow 26](#_Toc530588372)

[Figure 4‑3 Close flow for playback/TDM playback stream 27](#_Toc530588373)

[Figure 4‑4 Open flow for capture/TDM capture stream 28](#_Toc530588374)

[Figure 4‑5 Read data flow 29](#_Toc530588375)

[Figure 4‑6 Close flow for capture/TDM capture stream 30](#_Toc530588376)

[Figure 4‑7 Flow of getting volume information from hardware 31](#_Toc530588377)

[Figure 4‑8 Flow of setting volume of hardware 32](#_Toc530588378)

[Figure 4‑9 Flow of getting sample rate information from hardware 33](#_Toc530588379)

[Figure 4‑10 Flow of setting sample rate of hardware 34](#_Toc530588380)

[Figure 4‑11 Flow of getting Renderer output channel information 35](#_Toc530588381)

[Figure 4‑12 Flow of setting Renderer output channel 36](#_Toc530588382)

[Figure 4‑13 Flow of Equalizer control’s status getting 37](#_Toc530588383)

[Figure 4‑14 Flow of Equalizer activation setting 38](#_Toc530588384)

[Figure 4‑15 Flow of getting Equalizer parameters of hardware 39](#_Toc530588385)

[Figure 4‑16 Flow of setting Equalizer parameters of hardware 41](#_Toc530588386)

[Figure 4‑17 Flow of activating MIX control usage 42](#_Toc530588387)

[Figure 5‑1 Data path for playback 43](#_Toc530588388)

[Figure 5‑2 Data path when mixing 2 streams 45](#_Toc530588389)

[Figure 5‑3 Data path when mixing 3 streams 46](#_Toc530588390)

[Figure 5‑4 Data path when mixing 4 streams 47](#_Toc530588391)

[Figure 5‑5 Data path for capture stream 48](#_Toc530588392)

[Figure 5‑6 Data path for multichannel 49](#_Toc530588393)

[Figure 5‑7 Data path for recording multichannel stream 51](#_Toc530588394)

**List of Tables**

[Table 1‑1 Device tree files 8](#_Toc530588395)

[Table 1‑2 Required property for ADSP sound node 9](#_Toc530588396)

[Table 3‑1: List of API functions for PCM interface 14](#_Toc530588397)

[Table 3‑2: List of API functions for Volume control and Sample Rate control 15](#_Toc530588398)

[Table 3‑3 List of API functions for Equalizer control 15](#_Toc530588399)

[Table 3‑4 List of API functions for ASoC Platform interface 16](#_Toc530588400)

[Table 3‑5 List of API for ASoC Platform driver 16](#_Toc530588401)

[Table 5‑1 Target environment for each use case. 43](#_Toc530588402)

[Table 5‑2 Detailed information of ADSP ALSA sound card 43](#_Toc530588403)

[Table 5‑3 List of channel number conversation 45](#_Toc530588404)

[Table 5‑4 Detailed information of ADSP ALSA sound card 48](#_Toc530588405)

[Table 5‑5 User setting parameters for Parametric Equalizer 53](#_Toc530588406)

[Table 5‑6 User setting parameters for Graphic Equalizer 53](#_Toc530588407)

[Table 6‑1 Error code for ALSA callback functions 59](#_Toc530588408)

[Table 6‑2 Structures or type definition are defined in ALSA middle layer. 60](#_Toc530588409)

[Table 6‑3 Structures defined in ADSP ALSA Driver 61](#_Toc530588410)

[Table 6‑4 snd\_adsp\_control structure information 61](#_Toc530588411)

[Table 6‑5 snd\_adsp\_base\_info structure information 62](#_Toc530588412)

[Table 6‑6 snd\_adsp\_playback structure information 62](#_Toc530588413)

[Table 6‑7 snd\_adsp\_record structure information 63](#_Toc530588414)

[Table 6‑8 snd\_adsp\_card structure information 63](#_Toc530588415)

[Table 6‑9 snd\_adsp\_tdm\_playback structure information 63](#_Toc530588416)

[Table 6‑10 snd\_adsp\_tdm\_record structure information 63](#_Toc530588417)

ADSP ALSA Driver Specification

RCG3AHPDL4101ZDO

Rev. 1.00

Apr 26, 2018

# Overview

This material describes detailed information of ALSA APIs implemented in ADSP ALSA Driver:

* APIs for PCM data control (**PCM interface**).
* APIs for hardware control (**Control interface**).

Above interfaces belong to ALSA middle layer. They need to define **PCM callbacks** and **Control callbacks** in ADSP ALSA driver.

* APIs for platform interface.
* APIs for platform driver.

ALSA SoC core

Shared memory area

ADSP

Audio applications

*User side*

*Kernel side*

*Hardware* side

ALSA Lib

Renderer/Capture/ Equalizer/TDM Renderer/TDM Capture plugin

*PCM parameters*

*Extension parameters (volume, SRC, Equalizer)*

PCM interface

Control interface

*Kernel* side

ALSA framework supported by Linux

ALSA SoC core

Shared memory, ADSP hardware, codec

ADSP Driver Extension

PCM data

Communication

Platform driver, Machine driver, Codec driver

Audio application layer

***Legend:***

Plugins (Capture, Renderer, Equalizer, TDM Capture, TDM Renderer)

The target of this document is in side of red square.

ALSA Middleware Layer

Platform driver (rcar\_adsp\_sound)

ADSP ALSA

ADSP Driver Extension

CPU DAI

Machine driver

Codec driver

Codec DAI

Codec DAI

Codec DAI

ADSP ALSA

CPU DAI and Codec DAI

PCM Interface, Control Interface

Proxy Extension Interface

Proxy Extension Interface

ADSP Driver

Codec

ADSP Driver

Codec

Codec

CPU DAI

CPU DAI

Machine driver

Machine driver

Figure 1‑1 Overview architecture of ADSP driver.

* **Audio applications (aplay, arecord, amixer, etc)**:

The user applications that support to play or record sound by using ALSA library.

* **ALSA Lib**:

The ALSA library APIs are the interface to the ALSA drivers.

* **ALSA Middle Layer**:

It is a set of libraries which APIs gives applications access to the sound card drivers. And it can be broken down into the major interfaces such as control interface, PCM interface, raw MIDI interface, timer interface, sequencer interface and mixer interface.

* **ALSA SoC core:**

It is part of ALSA Framework and does processing of PCM data

* **ADSP ALSA**:

It is an ALSA device driver, implements to register a sound card for ADSP device. It provides callback functions for the native supports from ALSA framework to perform both playback and record. For playback/TDM playback, it receives PCM data from user app and transfers to ADSP Renderer plugin/ADSP TDM Renderer plugin. For record, it receives PCM data from ADSP Capture plugin/ADSP TDM Capture plugin and transfers to user app. The equalization function can be integrated into playback and record by routing between Equalizer and Renderer plugin, and between Equalizer and Capture plugins.

* **CPU DAI**:

DAI stands for Digital Audio Interface. CPU DAI is the interface for the platform driver to communicate with other drivers.

* **Platform driver**:

This is used to register ADSP sound card into ASoC framework. It holds ADSP ALSA driver, ADSP Driver Extension and ADSP sound card.

* **Codec driver**:

It represents interface for codecs.

* **Codec DAI**:

The DAI for codecs to communicate with other drivers

* **Machine driver**:

The ASoC machine (or board) driver is the code that glues together the platform driver and codec driver.

* **Proxy Extension Interface**:

APIs of methods through which ADSP Driver Extension communicates with shared memory area in Hardware side.

* **Shared memory area**:

Shared memory is a memory area which can be read and written by both CPU and ADSP.

* **ADSP**:

It is an audio DSP hardware unit. It provides ADSP framework which has the capability to control and execute multiple plugins (Renderer/Capture/Equalizer/TDM Renderer/TDM Capture) for playback, record, TDM and equalization. The communication between ADSP side and CPU side is performed by the interrupt, and the shared memory area.

## Device Tree

Below table describes which DTS files need to be considered to update when using ADSP sound driver for playback/record.

As Salvator-X/XS and Ebisu does not support TDM multi-channel, it is checked only in Starter-Kit and Kingfisher environment. Since the Starter-Kit and Kingfisher environment are not supported by the standard, it is necessary for users to prepare by yourself.

Table 1‑1 Device tree files

|  |  |  |
| --- | --- | --- |
| Target CPU | Target board | Device tree files (for example : Yocto v3.15.0) |
| R-Car H3 | Salvator-X/XS | arch/arm64/boot/dts/renesas/r8a7795-es1-salvator-x.dts  arch/arm64/boot/dts/renesas/r8a7795-salvator-x.dts  arch/arm64/boot/dts/renesas/r8a7795-salvator-xs.dts  each include below:  arch/arm64/boot/dts/renesas/r8a7795.dtsi,  arch/arm64/boot/dts/renesas/salvator-common.dtsi |
| R-Car M3 | Salvator-X/XS | arch/arm64/boot/dts/renesas/r8a7796-salvator-x.dts  arch/arm64/boot/dts/renesas/r8a7796-salvator-xs.dts  each include below:  arch/arm64/boot/dts/renesas/r8a7796.dtsi,  arch/arm64/boot/dts/renesas/salvator-common.dtsi |
| R-Car M3N | Salvator-X/XS | arch/arm64/boot/dts/renesas/r8a77965-salvator-x.dts  arch/arm64/boot/dts/renesas/r8a77965-salvator-xs.dts  each include below:  arch/arm64/boot/dts/renesas/r8a77965.dtsi,  arch/arm64/boot/dts/renesas/salvator-common.dtsi |
| R-Car E3 | Ebisu | arch/arm64/boot/dts/renesas/r8a77990-es10-ebisu.dts  include below:  arch/arm64/boot/dts/renesas/r8a77990.dtsi |

Table 1‑2 Required property for ADSP sound node

|  |  |
| --- | --- |
| Property name | Value |
| compatible | "renesas,rcar\_adsp\_sound\_gen3" |

* Example setting route:

1. Use ADSP sound with MIX function on Salvator-XS/H3:

* Select Machine driver: Simple SCU Card or Audio Graph SCU Card which support MIX function. This example will use Audio Graph SCU Card for illustration.

(Refer to *Documentation/devicetree/bindings/sound/simple-scu-card.txt* or *Documentation/devicetree/bindings/sound/audio-graph-scu-card.txt* for more detail setting of sound card)

* Select Codec driver: AK4613 codec which supports I2C format and controls audio chip on Salvator-XS/H3 board.

(Refer to *Documentation/devicetree/bindings/sound/ak4613.txt* for more information)

* Add ADSP sound node to device tree.

The node is placed in the root node of *arch/arm64/boot/dts/renesas/r8a7795.dtsi*

/ {

…

rcar\_adsp\_sound: adsp\_sound {

compatible = "renesas,rcar\_adsp\_sound\_gen3";

status = "disabled";

};

…

};

* Add routing between ADSP sound driver and codec driver.

Update routing configuration in *arch/arm64/boot/dts/renesas/salvator-common.dtsi*

Define DAI indexes for each Mixing port, each DAI connects to AK4613’s DAI as below:

&rcar\_adsp\_sound {

status = "okay";

/\* Multiple DAI \*/

#sound-dai-cells = <1>;

ports {

#address-cells = <1>;

#size-cells = <0>;

adsp\_port0: port@0 {

reg = <0>;

adsp\_endpoint0: endpoint {

remote-endpoint = <&ak4613\_endpoint0>;

dai-format = "left\_j";

};

};

adsp\_port1: port@1 {

reg = <1>;

adsp\_endpoint1: endpoint {

remote-endpoint = <&ak4613\_endpoint1>;

dai-format = "left\_j";

};

};

adsp\_port2: port@2 {

reg = <2>;

adsp\_endpoint2: endpoint {

remote-endpoint = <&ak4613\_endpoint2>;

dai-format = "left\_j";

};

};

adsp\_port3: port@3 {

reg = <3>;

adsp\_endpoint3: endpoint {

remote-endpoint = <&ak4613\_endpoint3>;

dai-format = "left\_j";

};

};

};

};

Keep rcar-sound card connects to AK4613 codec with 4th end point:

&rcar\_sound {

…

ports {

…

rsnd\_port0: port@0 {

reg = <0>;

rsnd\_endpoint0: endpoint {

remote-endpoint = <&ak4613\_endpoint4>;

…

};

The codec DAIs are also connect to ADSP’s DAIs, and rcar-sound’s DAI:

ak4613: codec@10 {

compatible = "asahi-kasei,ak4613";

…

ports {

#address-cells = <1>;

#size-cells = <0>;

ak4613\_endpoint0: endpoint@0 {

reg = <0>;

remote-endpoint = <&adsp\_endpoint0>;

};

ak4613\_endpoint1: endpoint@1 {

reg = <1>;

remote-endpoint = <&adsp\_endpoint1>;

};

ak4613\_endpoint2: endpoint@2 {

reg = <2>;

remote-endpoint = <&adsp\_endpoint2>;

};

ak4613\_endpoint3: endpoint@3 {

reg = <3>;

remote-endpoint = <&adsp\_endpoint3>;

};

ak4613\_endpoint4: endpoint@4 {

reg = <4>;

remote-endpoint = <&rsnd\_endpoint0>;

};

};

};

Update sound node to routing ADSP sound driver and codec AK4613 driver:

sound\_card: sound {

compatible = "audio-graph-scu-card";

label = "rcar-sound";

prefix = "ak4613";

routing = "ak4613 Playback", "Playback0",

"ak4613 Playback", "Playback1",

"ak4613 Playback", "Playback2",

"ak4613 Playback", "Playback3",

"Capture0", "ak4613 Capture",

"Capture1", "ak4613 Capture",

"Capture2", "ak4613 Capture",

"Capture3", "ak4613 Capture",

"ak4613 Playback", "DAI0 Playback",

"DAI0 Capture", "ak4613 Capture";

dais = <&adsp\_port0

&adsp\_port1

&adsp\_port2

&adsp\_port3

&rsnd\_port0>;

};

[Note]

- If current kernel-source configured HDMI audio for rcar-sound. It means rcar-sound is routed to HDMI codec, and AK4613 codec. In such cases, integrating ADSP sound is done by removing HDMI audio from rcar-sound.

For example, remove below content in *arch/arm64/boot/dts/renesas/r8a7795-salvator-xs.dts*

&sound\_card {

dais = <&rsnd\_port0 /\* ak4613 \*/

&rsnd\_port1 /\* HDMI0 \*/

&rsnd\_port2>; /\* HDMI1 \*/

};

- The device index of R-Car sound (*rcar\_sound*) changed to 4 as above configurations.

So, it is necessary to add the index number when setting its control.

# Terminologies

ALSA Advanced Linux Sound Architecture (ALSA) is a software framework and part of the Linux kernel that provides an application programming interface (API) for sound card device drivers.

Sound card The term “sound card” in this material is external audio interfaces used for audio applications to communicate with audio hardware.

Stream A PCM interface consists of PCM playback and capture streams and each pcm stream consists of one or more pcm substreams.

Substream A substream correspond to a PCM file opened or recorded.

# API Specification

## APIs for PCM Interface

Below table presents APIs used for PCM interface.

Table 3‑1: List of API functions for PCM interface

|  |  |  |
| --- | --- | --- |
| Number | API functions | Description |
| 1 | snd\_adsp\_pcm\_open | Register a Capture/Renderer plugin or a TDM Capture/Renderer plugin. It also registers Equalizer plugin in case of Capture/Renderer if Equalizer is used. It also gets range of hardware parameter into substream. |
| 2 | snd\_adsp\_pcm\_close | Unregister Capture/Renderer plugin or TDM Capture/Renderer plugin. It also unregisters Equalizer plugin in case of Capture/Renderer if Equalizer is used. |
| 3 | snd\_adsp\_pcm\_hw\_params | This callback is used to allocate ALSA buffer to transfer data. |
| 4 | snd\_adsp\_pcm\_hw\_free | This callback is used to release the allocated ALSA buffer. |
| 5 | snd\_adsp\_pcm\_prepare | Initialize hardware data pointer. Capture/Renderer and Equalizer (if Equalizer is used) or TDM Capture/Renderer will be set up to go to running state. |
| 6 | snd\_adsp\_pcm\_trigger | Start, stop, resume, suspend pcm substream. |
| 7 | snd\_adsp\_pcm\_ack | This callback is called in read/write operation. It calls transfer function to transfer data from ALSA buffer to ADSP buffer and vice versa. |
| 8 | snd\_adsp\_pcm\_pointer | Return hardware data position in the buffer in frames. |

## APIs for hardware control

Below table presents APIs used for Control interface. The interface includes Volume control, Sample Rate control, Equalizer control.

Table 3‑2: List of API functions for Volume control and Sample Rate control

|  |  |  |
| --- | --- | --- |
| Number | API function | Description |
| 1 | snd\_adsp\_control\_volume\_info | Get detail information on volume control |
| 2 | snd\_adsp\_control\_sample\_rate\_info | Get detail information on sample rate control |
| 3 | snd\_adsp\_control\_rdr\_out\_channel\_info | Get detail information on Renderer output channel control |
| 4 | snd\_adsp\_control\_volume\_get | Get volume setting value |
| 5 | snd\_adsp\_control\_sample\_rate\_get | Get sample rate output setting value |
| 6 | snd\_adsp\_control\_rdr\_out\_channel\_get | Get Renderer output channel setting value |
| 7 | snd\_adsp\_control\_volume\_put | Set volume value |
| 8 | snd\_adsp\_control\_sample\_rate\_put | Set sample rate output value |
| 9 | snd\_adsp\_control\_rdr\_out\_channel\_put | Set Renderer output channel |

Table 3‑3 List of API functions for Equalizer control

|  |  |  |
| --- | --- | --- |
| Number | API function | Description |
| 1 | snd\_adsp\_control\_eqz\_switch\_info | Get detail info on the equalizer switch control. |
| 2 | snd\_adsp\_control\_eqz\_switch\_get | Get information about equalizer activation. |
| 3 | snd\_adsp\_control\_eqz\_switch\_put | Enable or disable equalizer control. |
| 4 | snd\_adsp\_control\_eqz\_info | Get detailed info on the equalizer control. |
| 5 | snd\_adsp\_control\_eqz\_get | Get equalizer parameters. |
| 6 | snd\_adsp\_control\_eqz\_put | Set equalizer parameters. |

## APIs for for ASoC Platform interface

Below table shows APIs used for ASoC Platform interface

Table 3‑4 List of API functions for ASoC Platform interface

|  |  |  |
| --- | --- | --- |
| Number | API function | Description |
| 1 | snd\_adsp\_pcm\_new | Registers control interface for ADSP soundcard and pre-allocates ALSA buffer |

## APIs for for ASoC Platform driver

Below table shows APIs used for ASoC Platform driver

Table 3‑5 List of API for ASoC Platform driver

|  |  |  |
| --- | --- | --- |
| Number | API function | Description |
| 1 | snd\_adsp\_probe | Register platform device and add ADSP sound card component into ASoC framework |
| 2 | snd\_adsp\_remove | Unregister platform driver and remove ADSP sound card component from ASoC framework |

## Detail of APIs for PCM Interface

### snd\_adsp\_pcm\_open

FD\_DRV\_ALSA\_001

|  |  |  |
| --- | --- | --- |
| snd\_adsp\_pcm\_open | | |
| Synopsis | This callback is used for record/playback streams and TDM streams.  In Capture/Renderer case, it registers a Capture/Renderer plugin. It also registers Equalizer plugin if Equalizer switch is set to 1. Get range of hardware parameter into substream.  In TDM Capture/Renderer case, it registers a TDM Capture/Renderer plugin. It also gets range of hardware parameter into substream. | |
| Syntax | static int snd\_adsp\_pcm\_open(struct snd\_pcm\_substream \*substream) | |
| Parameter | struct snd\_pcm\_substream \*substream | Pointer to a pcm substream |
| Return value | 0 | Success |
| -EINVAL | The registering of Capture/Renderer or Equalizer plugin, or a TDM Capture/Renderer plugin fails. |

[Covers: RD\_001]

### snd\_adsp\_pcm\_close

FD\_DRV\_ALSA\_002

|  |  |  |
| --- | --- | --- |
| snd\_adsp\_pcm\_close | | |
| Synopsis | This callback is used for record/playback streams and TDM streams.  In Capture/Renderer case, it unregisters the Capture/Renderer plugin. It also unregisters Equalizer plugin if Equalizer switch is set to 1.  In TDM Capture/Renderer case, it unregisters the TDM Capture/Renderer plugin. | |
| Syntax | static int snd\_adsp\_pcm\_close(struct snd\_pcm\_substream \*substream) | |
| Parameter | struct snd\_pcm\_substream \*substream | Pointer to a pcm substream |
| Return value | 0 | Success |
| -EINVAL | The unregistering of the Capture/Renderer or Equalizer plugin, or TDM Capture/Renderer plugin fails. |

[Covers: RD\_001]

### snd\_adsp\_pcm\_hw\_params

FD\_DRV\_ALSA\_003

|  |  |  |
| --- | --- | --- |
| snd\_adsp\_pcm\_hw\_params | | |
| Synopsis | This callback is used to allocate ALSA buffer to transfer data. It also figures out the expiration time of high-resolution timer. | |
| Syntax | static int snd\_adsp\_pcm\_hw\_params(struct snd\_pcm\_substream \*substream, struct snd\_pcm\_hw\_params \*hw\_params) | |
| Parameter | struct snd\_pcm\_substream \*substream | Pointer to a pcm substream |
| struct snd\_pcm\_hw\_params \*hw\_params | Hardware parameter is set up by the application |
| Return value | 0 | Success |
| -ENOMEM | Cannot allocate ALSA buffer |

[Covers: RD\_001]

### snd\_adsp\_pcm\_hw\_free

FD\_DRV\_ALSA\_004

|  |  |  |
| --- | --- | --- |
| snd\_adsp\_pcm\_hw\_free | | |
| Synopsis | This callback is used to deallocate ALSA buffer. | |
| Syntax | static int snd\_adsp\_pcm\_hw\_free(struct snd\_pcm\_substream \*substream) | |
| Parameter | struct snd\_pcm\_substream \*substream | Pointer to a pcm substream |
| Return value | 0 | Always return 0 |
| -EINVAL | Cannot deallocate ALSA buffer |

[Covers: RD\_001]

### snd\_adsp\_pcm\_prepare

FD\_DRV\_ALSA\_005

|  |  |  |
| --- | --- | --- |
| snd\_adsp\_pcm\_prepare | | |
| Synopsis | Initialize hardware data position with value 0 then set default volume value is 100% unless user set volume before. In this function, Capture/Renderer plugin or TDM Capture/Renderer is set up to go to running state. In Capture/Renderer case, if Equalizer switch is enabled, it is also set up, then set route Capture/Renderer with Equalizer. This callback is also called when data overrun or underrun error occurs. | |
| Syntax | static int snd\_adsp\_pcm\_prepare(struct snd\_pcm\_substream \*substream) | |
| Parameter | struct snd\_pcm\_substream \*substream | Pointer to a pcm substream |
| Return value | 0 | Success |
| -EINVAL | Cannot set parameters to Capture/Renderer or Equalizer plugin, or TDM Capture/TDM Renderer plugin  Cannot allocate ADSP buffer pool  Cannot route Equalizer and Capture/Renderer,  ADSP plugin initialization fails. |

[Covers: RD\_001]

### snd\_adsp\_pcm\_trigger

FD\_DRV\_ALSA\_006

|  |  |  |
| --- | --- | --- |
| snd\_adsp\_pcm\_trigger | | |
| Synopsis | This is called to start, stop, resume, suspend pcm substream. | |
| Syntax | static int snd\_adsp\_pcm\_trigger(struct snd\_pcm\_substream \*substream, int idx) | |
| Parameter | struct snd\_pcm\_substream \*substream | Pointer to a pcm substream |
| int idx | Start, resume, suspend, stop command |
| Return value | 0 | Success |
| -EINVAL | Invalid command, cannot send fill buffer command to ADSP when performing capture, TDM capture functions |

[Covers: RD\_001]

### snd\_adsp\_pcm\_ack

FD\_DRV\_ALSA\_007

|  |  |  |
| --- | --- | --- |
| snd\_adsp\_pcm\_ack | | |
| Synopsis | This callback is called in read/write operation. It calls transfer function to transfer data from ALSA buffer to ADSP buffer and vice versa. | |
| Syntax | static int snd\_adsp\_pcm\_ack(struct snd\_pcm\_substream \*substream) | |
| Parameter | struct snd\_pcm\_substream \*substream | Pointer to a pcm substream. |
| Return value | 0 | Success |
| -EINVAL | Cannot transfer data from ALSA buffer to ADSP buffer, or runtime error has been trigger. |

[Covers: RD\_001]

### snd\_adsp\_pcm\_pointer

FD\_DRV\_ALSA\_008

|  |  |  |
| --- | --- | --- |
| snd\_adsp\_pcm\_pointer | | |
| Synopsis | Return data position on the buffer in frames. | |
| Syntax | static snd\_pcm\_uframes\_t snd\_adsp\_pcm\_pointer(struct snd\_pcm\_substream \*substream) | |
| Parameter | struct snd\_pcm\_substream \*substream | Pointer to a pcm substream |
| Return value | pointer | Position offset in frames on hardware buffer. |

[Covers: RD\_001]

## Detail of APIs for hardware control

* APIs get detailed information from the control

### snd\_adsp\_control\_volume\_info

FD\_DRV\_ALSA\_009

|  |  |  |
| --- | --- | --- |
| snd\_adsp\_control\_volume\_info | | |
| Synopsis | Get detailed information on volume control in playback, capture, TDM playback, TDM capture cases | |
| Syntax | static int snd\_adsp\_control\_volume\_info(struct snd\_kcontrol \*kcontrol, struct snd\_ctl\_elem\_info \*uinfo) | |
| Parameter | struct snd\_kcontrol \*kcontrol | Pointer to control instance |
| struct snd\_ctl\_elem\_info \*uinfo | Pointer to info structure of volume control |
| Return value | 0 | Always return 0 |

[Covers: RD\_001]

### snd\_adsp\_control\_sample\_rate\_info

FD\_DRV\_ALSA\_010

|  |  |  |
| --- | --- | --- |
| snd\_adsp\_control\_sample\_rate\_info | | |
| Synopsis | Get detailed information on sample rate control in playback, capture, TDM playback, TDM capture cases | |
| Syntax | static int snd\_adsp\_control\_sample\_rate\_info(struct snd\_kcontrol \*kcontrol, struct snd\_ctl\_elem\_info \*uinfo) | |
| Parameter | struct snd\_kcontrol \*kcontrol | Pointer to control instance |
| struct snd\_ctl\_elem\_info \*uinfo | Pointer to info structure of sample rate control |
| Return value | 0 | Always return 0 |

[Covers: RD\_001]

### snd\_adsp\_control\_eqz\_switch\_info

FD\_DRV\_ALSA\_011

|  |  |  |
| --- | --- | --- |
| snd\_adsp\_control\_eqz\_switch\_info | | |
| Synopsis | Get detailed info on the equalizer switch control in playback, capture cases | |
| Syntax | static int snd\_adsp\_control\_eqz\_switch\_info(struct snd\_kcontrol \*kcontrol, struct snd\_ctl\_elem\_info \*uinfo) | |
| Parameter | struct snd\_kcontrol \*kcontrol | Pointer to control instance |
| struct snd\_ctl\_elem\_info \*uinfo | Pointer to info structure of the equalizer switch control |
| Return value | 0 | Always return 0 |

[Covers: RD\_001]

### snd\_adsp\_control\_eqz\_info

FD\_DRV\_ALSA\_012

|  |  |  |
| --- | --- | --- |
| snd\_adsp\_control\_eqz\_info | | |
| Synopsis | Get detailed info on the equalizer control in playback, capture cases | |
| Syntax | static int snd\_adsp\_control\_eqz\_info(struct snd\_kcontrol \*kcontrol, struct snd\_ctl\_elem\_info \*uinfo) | |
| Parameter | struct snd\_kcontrol \*kcontrol | Pointer to control instance |
| struct snd\_ctl\_elem\_info \*uinfo | Pointer to info structure of the equalizer control |
| Return value | 0 | Always return 0 |

[Covers: RD\_001]

### snd\_adsp\_control\_rdr\_out\_channel\_info

FD\_DRV\_ALSA\_013

|  |  |  |
| --- | --- | --- |
| snd\_adsp\_control\_rdr\_out\_channel\_info | | |
| Synopsis | Get detailed info of Renderer output channel control in playback case | |
| Syntax | static int snd\_adsp\_control\_rdr\_out\_channel\_info(struct snd\_kcontrol \*kcontrol, struct snd\_ctl\_elem\_info \*uinfo) | |
| Parameter | struct snd\_kcontrol \*kcontrol | Pointer to control instance |
| struct snd\_ctl\_elem\_info \*uinfo | Pointer to info structure of Renderer output channel |
| Return value | 0 | Always return 0 |

[Covers: RD\_001]

### snd\_adsp\_control\_volume\_get

FD\_DRV\_ALSA\_014

|  |  |  |
| --- | --- | --- |
| snd\_adsp\_control\_volume\_get | | |
| Synopsis | Get the PCM volume rate setting value in playback, capture, TDM playback, TDM capture cases | |
| Syntax | static int snd\_adsp\_control\_volume\_get(struct snd\_kcontrol \*kcontrol, struct snd\_ctl\_elem\_value \*ucontrol) | |
| Parameter | struct snd\_kcontrol \*kcontrol | Pointer to control instance |
| struct snd\_ctl\_elem\_value \*ucontrol | Pointer to volume value |
| Return value | 0 | Success |
| -EINVAL | Can’t get parameter information from ADSP. |

[Covers: RD\_001]

### snd\_adsp\_control\_sample\_rate\_get

FD\_DRV\_ALSA\_015

|  |  |  |
| --- | --- | --- |
| snd\_adsp\_control\_sample\_rate\_get | | |
| Synopsis | Get sample rate value of hardware output in playback/TDM playback case and sample rate of hardware input in capture/TDM capture case. | |
| Syntax | static int snd\_adsp\_control\_sample\_rate\_get(struct snd\_kcontrol \*kcontrol, struct snd\_ctl\_elem\_value \*ucontrol) | |
| Parameter | struct snd\_kcontrol \*kcontrol | Pointer to control instance |
| struct snd\_ctl\_elem\_value \*ucontrol | Pointer to sample rate value |
| Return value | 0 | Success |
| -EINVAL | Can’t get parameter information from ADSP. |

[Covers: RD\_001]

### snd\_adsp\_control\_eqz\_get

FD\_DRV\_ALSA\_016

|  |  |  |
| --- | --- | --- |
| snd\_adsp\_control\_eqz\_get | | |
| Synopsis | Get parameters info of equalizer control in playback, capture cases | |
| Syntax | static int snd\_adsp\_control\_eqz\_get(struct snd\_kcontrol \*kcontrol, struct snd\_ctl\_elem\_value \*ucontrol) | |
| Parameter | struct snd\_kcontrol \*kcontrol | Pointer to control instance |
| struct snd\_ctl\_elem\_value \*ucontrol | Pointer to equalizer parameters. |
| Return value | 0 | Success |
| -EINVAL | Can’t get parameter information from ADSP. |

[Covers: RD\_001]

### snd\_adsp\_control\_eqz\_switch\_get

FD\_DRV\_ALSA\_017

|  |  |  |
| --- | --- | --- |
| snd\_adsp\_control\_eqz\_switch\_get | | |
| Synopsis | Get status of Equalizer control in playback, capture cases | |
| Syntax | static int snd\_adsp\_control\_eqz\_switch\_get(struct snd\_kcontrol \*kcontrol, struct snd\_ctl\_elem\_value \*ucontrol) | |
| Parameter | struct snd\_kcontrol \*kcontrol | Pointer to control instance |
| struct snd\_ctl\_elem\_value \*ucontrol | Pointer to equalizer switch. |
| Return value | 0 | Always return 0 |

[Covers: RD\_001]

### snd\_adsp\_control\_rdr\_out\_channel\_get

FD\_DRV\_ALSA\_018

|  |  |  |
| --- | --- | --- |
| snd\_adsp\_control\_rdr\_out\_channel\_get | | |
| Synopsis | Get value of Renderer output channel in playback case | |
| Syntax | static int snd\_adsp\_control\_rdr\_out\_channel\_get(struct snd\_kcontrol \*kcontrol, struct snd\_ctl\_elem\_value \*ucontrol) | |
| Parameter | struct snd\_kcontrol \*kcontrol | Pointer to control instance |
| struct snd\_ctl\_elem\_value \*ucontrol | Pointer to Renderer output channel value |
| Return value | 0 | Success |
| -EINVAL | Can’t get value of output channel from Renderer plugin |

[Covers: RD\_001]

### snd\_adsp\_control\_volume\_put

FD\_DRV\_ALSA\_019

|  |  |  |
| --- | --- | --- |
| snd\_adsp\_control\_volume\_put | | |
| Synopsis | Set the PCM volume rate value in playback, capture, TDM playback, TDM capture cases | |
| Syntax | static int snd\_adsp\_control\_volume\_put(struct snd\_kcontrol \*kcontrol, struct snd\_ctl\_elem\_value \*ucontrol) | |
| Parameter | struct snd\_kcontrol \*kcontrol | Pointer to control instance |
| struct snd\_ctl\_elem\_value \*ucontrol | Pointer to volume setting value |
| Return value | 0 | Hardware parameter is still not changed. |
| 1 | Hardware parameter is changed. |
| -EINVAL | Can’t set parameter to ADSP.  Set volume TDM Capture/TDM Renderer at runtime |

[Covers: RD\_001]

### snd\_adsp\_control\_sample\_rate\_put

FD\_DRV\_ALSA\_020

|  |  |  |
| --- | --- | --- |
| snd\_adsp\_control\_sample\_rate\_put | | |
| Synopsis | Set sample rate for hardware output in playback and TDM playback case, input in the capture and TDM capture case | |
| Syntax | static int snd\_adsp\_control\_sample\_rate\_put(struct snd\_kcontrol \*kcontrol, struct snd\_ctl\_elem\_value \*ucontrol) | |
| Parameter | struct snd\_kcontrol \*kcontrol | Pointer to control instance |
| struct snd\_ctl\_elem\_value \*ucontrol | Pointer to sample rate value |
| Return value | 1 | Hardware parameter is changed. |
| -EINVAL | Can’t set parameter to ADSP. |

[Covers: RD\_001]

### snd\_adsp\_control\_eqz\_put

FD\_DRV\_ALSA\_021

|  |  |  |
| --- | --- | --- |
| snd\_adsp\_control\_eqz\_put | | |
| Synopsis | Set equalizer parameter in playback, capture cases | |
| Syntax | static int snd\_adsp\_control\_eqz\_put(struct snd\_kcontrol \*kcontrol, struct snd\_ctl\_elem\_value \*ucontrol) | |
| Parameter | struct snd\_kcontrol \*kcontrol | Pointer to control instance |
| struct snd\_ctl\_elem\_value \*ucontrol | Pointer to equalizer parameters |
| Return value | 1 | Hardware parameter is changed. |
| -EINVAL | Can’t set parameter to ADSP. |

[Covers: RD\_001]

### snd\_adsp\_control\_eqz\_switch\_put

FD\_DRV\_ALSA\_022

|  |  |  |
| --- | --- | --- |
| snd\_adsp\_control\_eqz\_switch\_put | | |
| Synopsis | Enable or disable Equalizer control in playback, capture cases | |
| Syntax | static int snd\_adsp\_control\_eqz\_switch\_put(struct snd\_kcontrol \*kcontrol, struct snd\_ctl\_elem\_value \*ucontrol) | |
| Parameter | struct snd\_kcontrol \*kcontrol | Pointer to control instance |
| struct snd\_ctl\_elem\_value \*ucontrol | Pointer to equalizer switch |
| Return value | 1 | Success |
| -EINVAL | Cannot enable EQZ when Renderer/Capture plugin is running. |

[Covers: RD\_001]

### snd\_adsp\_control\_rdr\_out\_channel\_put

FD\_DRV\_ALSA\_023

|  |  |  |
| --- | --- | --- |
| snd\_adsp\_control\_rdr\_out\_channel\_put | | |
| Synopsis | Set Renderer output channel in playback case | |
| Syntax | static int snd\_adsp\_control\_rdr\_out\_channel\_put(struct snd\_kcontrol \*kcontrol, struct snd\_ctl\_elem\_value \*ucontrol) | |
| Parameter | struct snd\_kcontrol \*kcontrol | Pointer to control instance |
| struct snd\_ctl\_elem\_value \*ucontrol | Pointer to Renderer output channel value |
| Return value | 1 | Parameter after setting is changed. |
| -EINVAL | Cannot set output channel for Renderer |

[Covers: RD\_001]

## Detail of APIs for ASoC Platform interface

### snd\_adsp\_pcm\_new

FD\_DRV\_ALSA\_024

|  |  |  |
| --- | --- | --- |
| snd\_adsp\_pcm\_new | | |
| Synopsis | This API registers necessary control interfaces for ADSP soundcard based on CPU DAI type (playback/capture type or TDM playback/TDM capture type). In playback/capture case, when registering control interfaces for the second/third/fourth playback/capture, MIX usage flag is raised to 1 for that stream to be played mixed with the first one. It also pre-allocates a memory region for ALSA buffer for transferring data. | |
| Syntax | static int snd\_adsp\_pcm\_new(struct snd\_soc\_pcm\_runtime \*runtime) | |
| Parameter | struct snd\_soc\_pcm\_runtime \*runtime | PCM runtime data |
| Return value | 0 | Success |
| -EINVAL | Cannot register a control into ALSA framework |

[Covers: RD\_001]

## Detail of APIs for ASoC Platform driver

### snd\_adsp\_probe

FD\_DRV\_ALSA\_025

|  |  |  |
| --- | --- | --- |
| snd\_adsp\_probe | | |
| Synopsis | This callback is called to register platform driver with 5 CPU DAIs (4 DAIs for playback/record and 1 DAI for TDM playback/TDM record) and add ADSP sound card component to ASoC framework. | |
| Syntax | static int snd\_adsp\_probe(struct platform\_device \*pdev) | |
| Parameter | struct platform\_device \*pdev | Pointer to platform device structure |
| Return value | 0 | Success |
| -ENOMEM | Cannot allocate ADSP card data structure |
| -EINVAL | Cannot register platform device  Cannot add ADSP sound card component to ASoC framework |

[Covers: RD\_001]

### snd\_adsp\_remove

FD\_DRV\_ALSA\_026

|  |  |  |
| --- | --- | --- |
| snd\_adsp\_remove | | |
| Synopsis | This callback is used to unregister platform driver and remove ADSP sound card component from ASoC framework. | |
| Syntax | static int snd\_adsp\_remove(struct platform\_device \*pdev) | |
| Parameter | struct platform\_device \*pdev | Pointer to platform device structure |
| Return value | 0 | Success |
| -ENODEV | ADSP sound card is invalid |

[Covers: RD\_001]

# Sequence diagram

## Playback/TDM playback streams flow

### Open a renderer/TDM renderer substream

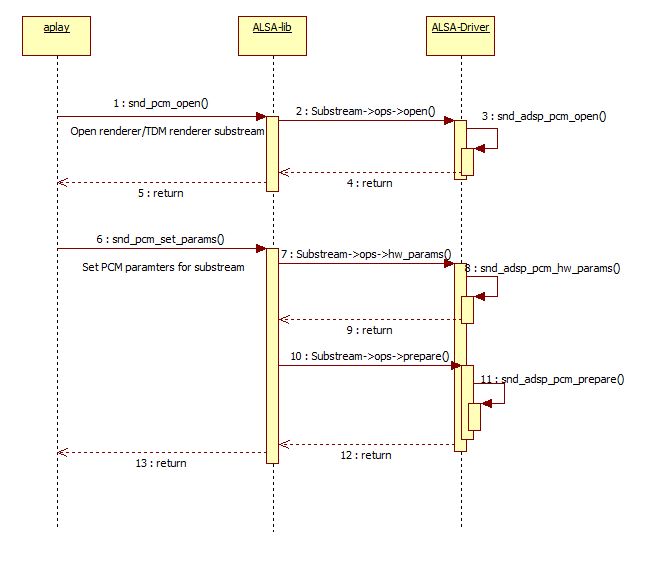


Figure 4‑1 Open flow for playback/TDM playback stream

### Write data flow

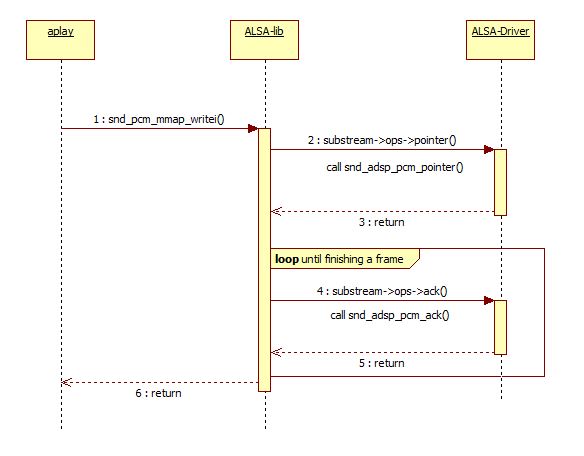


Figure 4‑2 Write data flow

### Close a playback/TDM playback substream

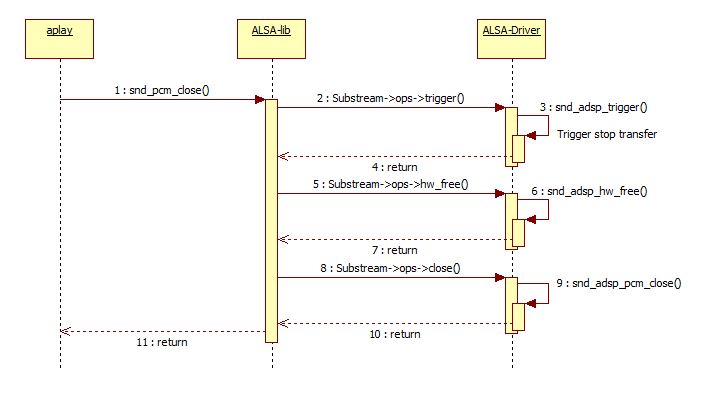


Figure 4‑3 Close flow for playback/TDM playback stream

## Capture/TDM capture streams flow

### Open a capture/TDM capture substream

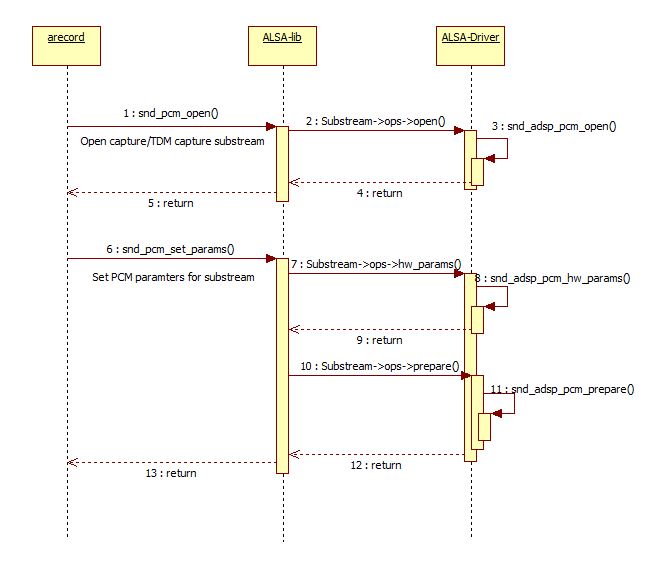


Figure 4‑4 Open flow for capture/TDM capture stream

### Read data flow

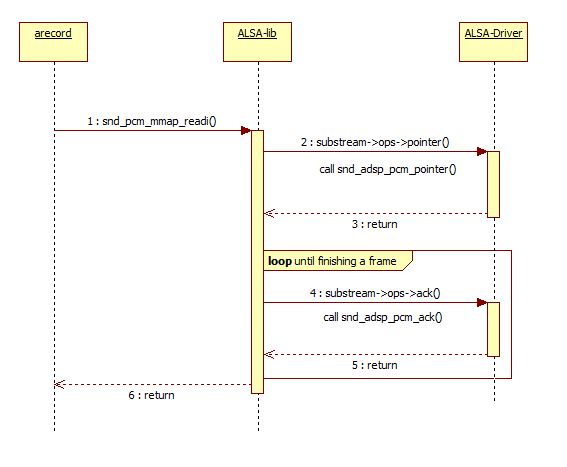


Figure 4‑5 Read data flow

### Close a capture/TDM capture substream

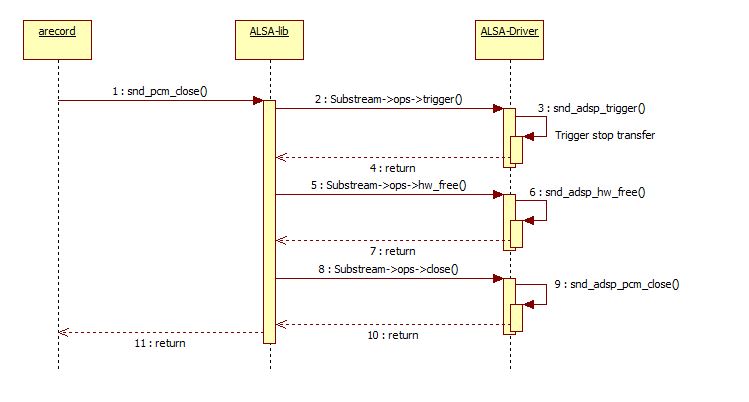


Figure 4‑6 Close flow for capture/TDM capture stream

## Volume Control Flow

### Get volume value from hardware

In playback case, amixer runs with argument: cget name=**”PlaybackVolume”**,index=n to get the volume value of n-th Renderer (n = 0, 1, 2, 3). This string maps to the control defined in ADSP ALSA Driver.

In capture case, amixer runs with argument: cget name=**”CaptureVolume”**,index=n to get the volume value of n-th Capture (n = 0, 1, 2, 3). This string maps to the control defined in ADSP ALSA Driver.

In TDM playback case, amixer runs with argument: cget name=**”TDMPlaybackVolume”**. This string maps to the control defined in ADSP ALSA Driver.

In TDM capture case, amixer runs with argument: cget name=**”TDMCaptureVolume”**. This string maps to the control defined in ADSP ALSA Driver.

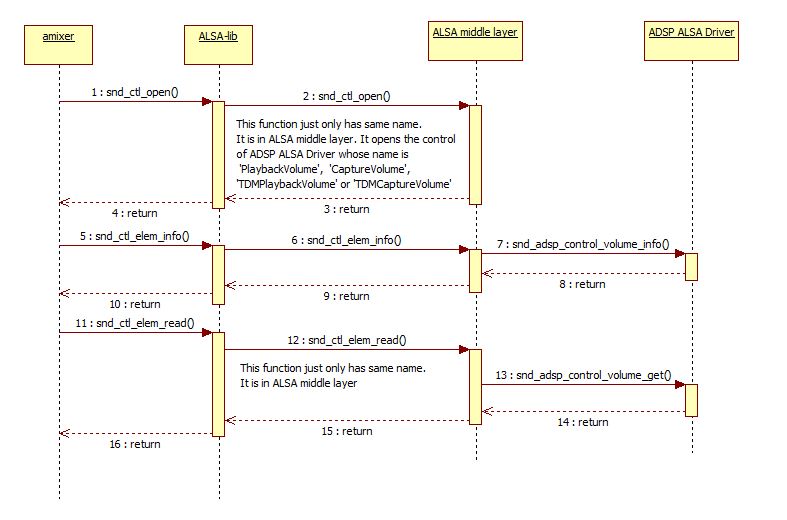


Figure 4‑7 Flow of getting volume information from hardware

### Set volume value to hardware

In playback case, amixer runs with argument: cset name=**”PlaybackVolume”**,index=n 200 to turn up the volume of the n-th playback stream to 200% (n = 0, 1, 2, 3). **“PlaybackVolume”** string maps the control defined in ADSP ALSA Driver.

In capture case, amixer runs with argument: cset name=”**CaptureVolume”**,index=n 50 to turn down volume of the n-th record stream to 50% (n = 0, 1, 2, 3). **“CaptureVolume”** string maps the control defined in ADSP ALSA Driver.

In TDM playback case, amixer runs with argument: cset name=**”TDMPlaybackVolume”** 50 to turn down the volume to 50%. **”TDMPlaybackVolume”** string maps the control defined in ADSP ALSA Driver.

In TDM capture case, amixer runs with argument: cset name=**”TDMCaptureVolume”** 200 to turn up the volume to 200%. **’TDMCaptureVolume’** string maps the control defined in ADSP ALSA Driver.

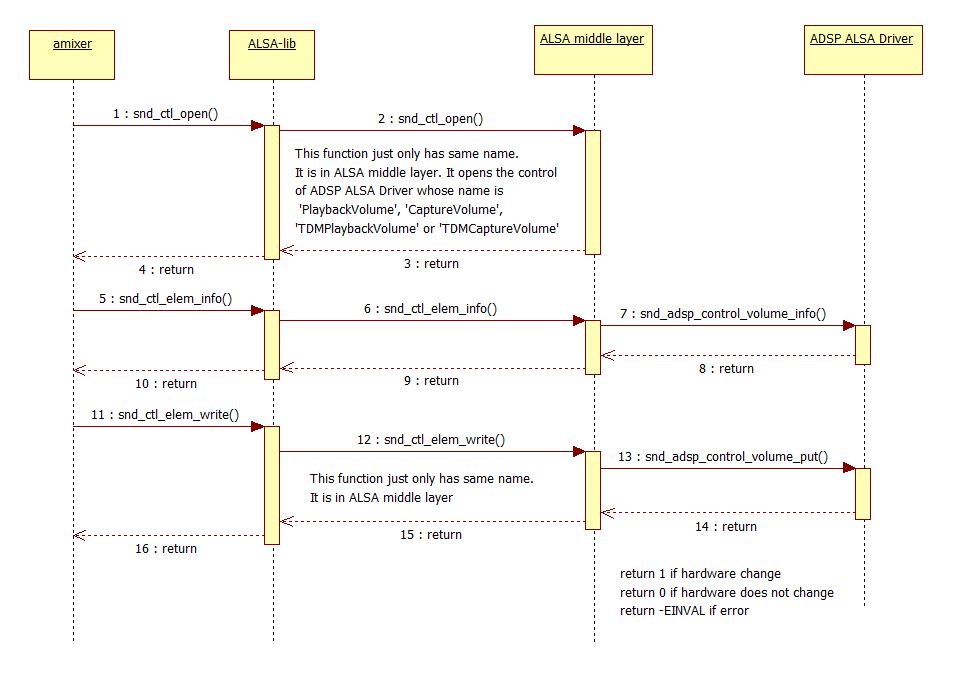


Figure 4‑8 Flow of setting volume of hardware

## Sample Rate Converter Control Flow

### Get sample rate of hardware

In playback case, amixer runs with argument: cget name=**”PlaybackOutRate”**,index=n to get n-th Renderer’s output sample rate (n = 0, 1, 2, 3). This string maps to the control defined in ADSP ALSA Driver.

In capture case, amixer runs with argument: cget name=**”CaptureInRate”**,index=n to get n-th Capture’s input sample rate (n = 0, 1, 2, 3). This string maps to the control defined in ADSP ALSA Driver.

In TDM playback case, amixer runs with argument: cget name=**”TDMPlaybackOutRate”**. This string maps to the control defined in ADSP ALSA Driver.

In TDM capture case, amixer runs with argument: cget name=”**TDMCaptureInRate”**. This string maps to the control defined in ADSP ALSA Driver.

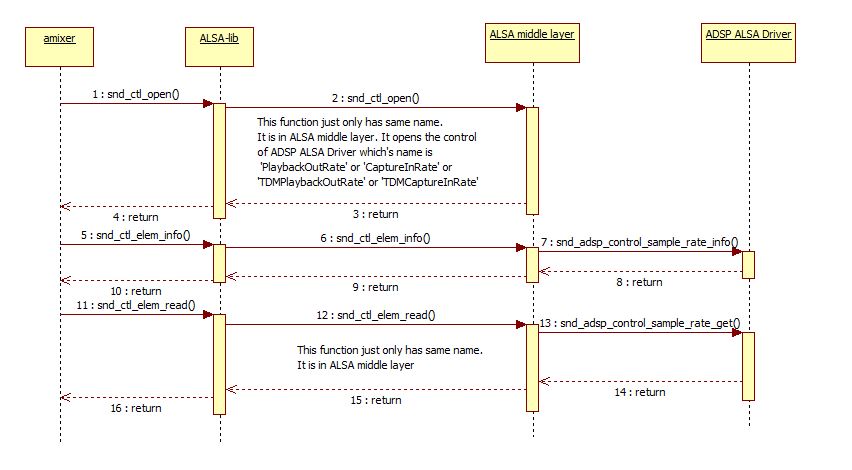


Figure 4‑9 Flow of getting sample rate information from hardware

### Set sample rate of hardware

In playback case, amixer runs with argument: cset name=”**PlaybackOutRate”**,index=n 44100 to set output sample rate of n-th Renderer to 44100 (n = 0, 1, 2, 3). **“PlaybackOutRate”** string maps to the control defined in ADSP ALSA Driver.

In capture case, amixer runs with argument: cset name=”**CaptureInRate”**,index=n 48000 to set input sample rate of n-th Capture input to 48000 (n = 0, 1, 2, 3). “**CaptureInRate”** string maps to the control defined in ADSP ALSA Driver.

In TDM playback case, amixer runs with argument: cset name=”**TDMPlaybackOutRate”** 44100 to set sample rate of hardware output to 44100. “**TDMPlaybackOutRate”** string maps to the control defined in ADSP ALSA Driver.

In TDM capture case, amixer runs with argument: cset name=**”TDMCaptureInRate”** 48000 to set sample rate of hardware input to 48000. “**TDMCaptureInRate”** string maps to the control defined in ADSP ALSA Driver.

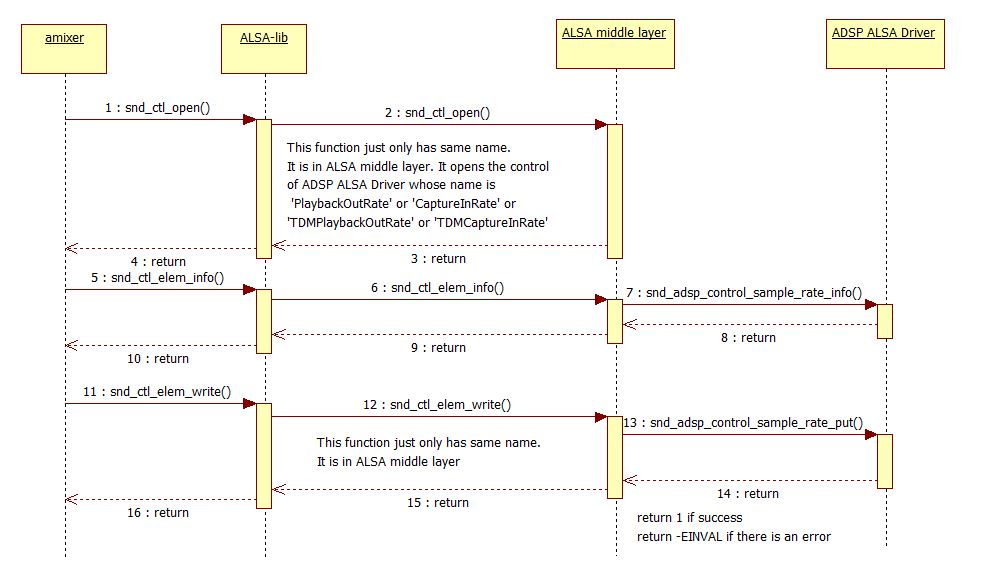


Figure 4‑10 Flow of setting sample rate of hardware

## Output Channel Control Flow

### Get output channel of Renderer

In playback case, amixer runs with argument: cget name=**”PlaybackOutChannel”**,index=n to get the output channel of n-th Renderer (n = 0, 1, 2, 3). This string maps to the control defined in ADSP ALSA Driver.

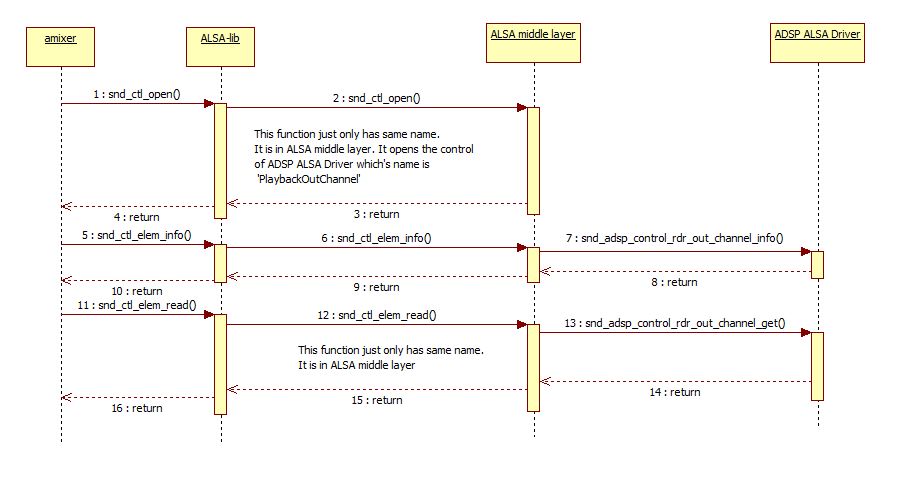


Figure 4‑11 Flow of getting Renderer output channel information

### Set output channel of Renderer

In playback case, amixer runs with argument: cset name=**”PlaybackOutChannel”**,index=n 2 to set n-th Renderer output channel to 2 (n = 0, 1, 2, 3). **“PlaybackOutChannel”** string maps to the control defined in ADSP ALSA Driver.

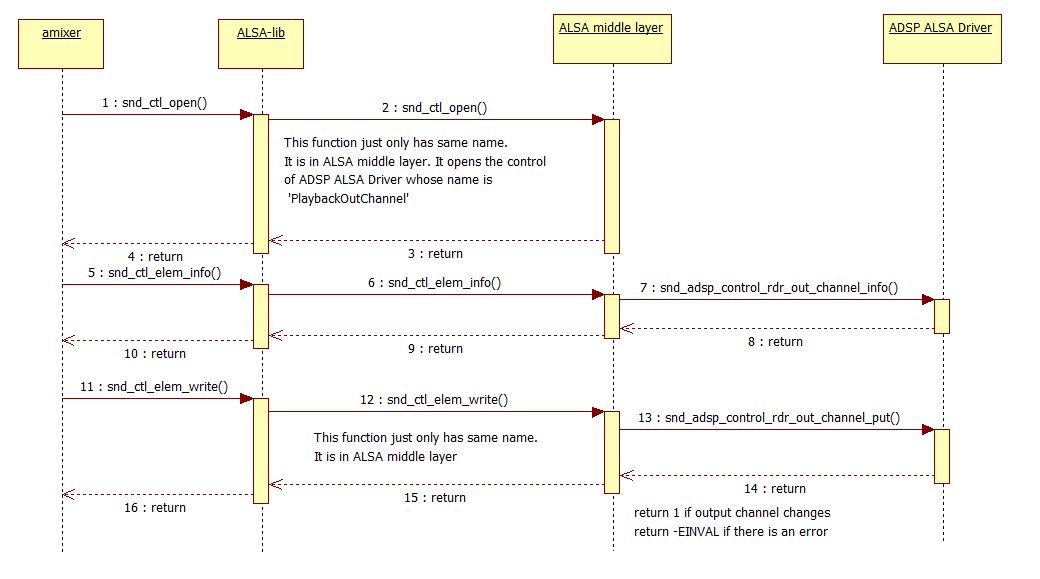


Figure 4‑12 Flow of setting Renderer output channel

## Equalizer Control Flow

### Get status of Equalizer control

In playback case, amixer runs with argument: cget name=**”PlaybackEQZSwitch”**,index=n to get Equalizer status of n-th Renderer (n = 0, 1, 2, 3). This string maps to the control defined in ADSP ALSA Driver.

In capture case, amixer runs with argument: cget name=**”CaptureEQZSwitch”**,index=n to get Equalizer status of n-th Capture (n = 0, 1, 2, 3). This string maps to the control defined in ADSP ALSA Driver.

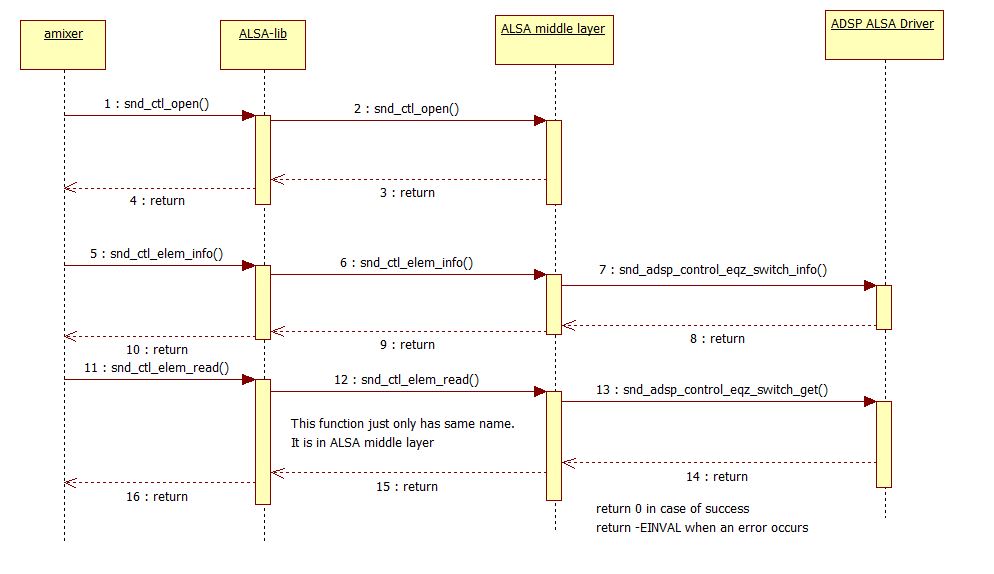


Figure 4‑13 Flow of Equalizer control’s status getting

### Enable Equalizer control

In playback case, amixer runs with argument: cset name=**”PlaybackEQZSwitch”**,index=n 1 to enable Equalizer status of n-th Renderer (n = 0, 1, 2, 3). This string maps to the control defined in ADSP ALSA Driver.

In capture case, amixer runs with argument: cset name=**”CaptureEQZSwitch”**,index=n 1 to enable Equalizer status of n-th Capture (n = 0, 1, 2, 3). This string maps to the control defined in ADSP ALSA Driver.

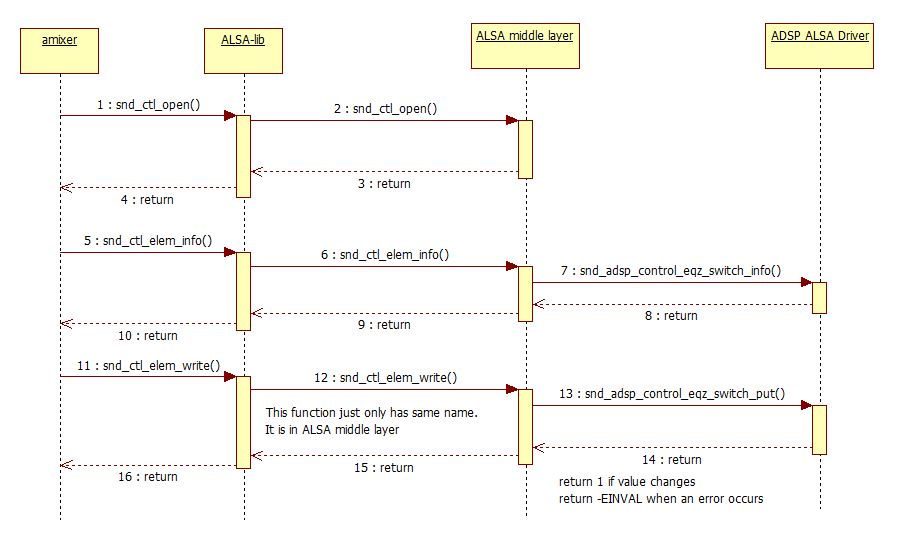


Figure 4‑14 Flow of Equalizer activation setting

### Get Equalizer parameters of hardware

In playback case, amixer runs with argument: cget name=**”PlaybackEQZControl”**,index=n to get parameters’ values of n-th Equalizer (n = 0, 1, 2, 3). This string maps to the control defined in ADSP ALSA Driver.

In capture case, amixer runs with argument: cget name=**”CaptureEQZControl”**,index=n to get parameters’ values of n-th Equalizer (n = 0, 1, 2, 3). This string maps to the control defined in ADSP ALSA Driver.

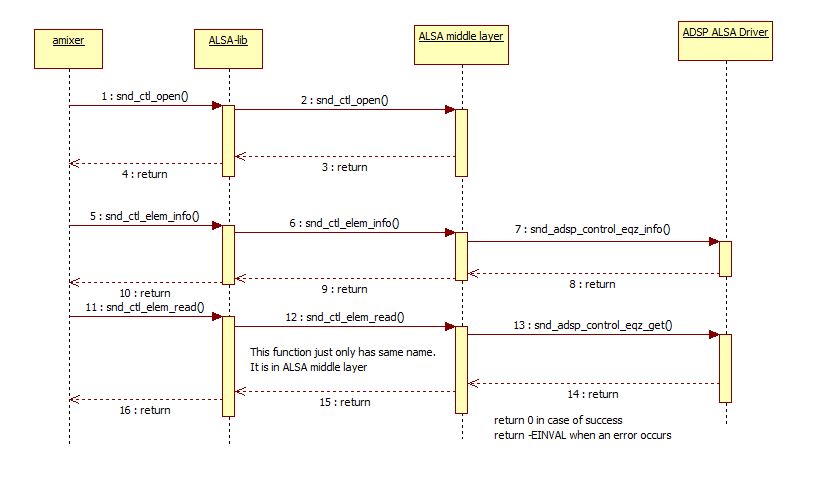


Figure 4‑15 Flow of getting Equalizer parameters of hardware

### Set Equalizer parameters of hardware

In playback case, amixer runs with argument: cset name=**”PlaybackEQZControl”**,index=n 1,1,…,1 (55 values) to set parameters’ values of n-th Equalizer (n = 0, 1, 2, 3). This string maps to the control defined in ADSP ALSA Driver.

In capture case, amixer runs with argument: cset name=**”CaptureEQZControl”**,index=n 1,1,…,1 (55 values) to set parameters’ values of n-th Equalizer (n = 0, 1, 2, 3). This string maps to the control defined in ADSP ALSA Driver.

In Parametric Equalizer, there are 9 filters. Each filter has its own parameters: frequency centre, bandwidth, filter type, gain base, gain. Therefore, there are 55 values to set as the table below:

|  |  |  |
| --- | --- | --- |
| **Parameters** | **Value range** | **Number** |
| Equalizer type | 0 (for Parametric) | 1 |
| Filter index | 1 - 9 | 9 |
| Frequency centre | 20 – 20000 | 9 |
| Bandwidth | 0.2 x 227 – 15 x 227 | 9 |
| Filter type | 0 – 2 | 9 |
| Gain base | 20 - 20000 | 9 |
| Gain | 10-10/20 x 228 – 1010/20 x 228 | 9 |
|  | | Total: 55 |

Order of parameters to set for Parametric Equalizer:

*Equalizer type, 9 x (filter index, frequency centre, bandwidth, filter type, gainbase, gain)*

In Graphic Equalizer, there are 5 filters, each of which has its own parameter: graphic gain. The settings are described below:

|  |  |  |
| --- | --- | --- |
| **Parameters** | **Value range** | **Number** |
| Equalizer type | 1 (for Graphic) | 1 |
| Filter index | 1 - 5 | 5 |
| Graphic gain | 10-10/20 x 228 – 1010/20 x 228 | 5 |
|  | | Total: 11 |

Order of parameters to set for Graphic Equalizer:

*Equalizer type, 5 x (filter index, graphic gain)*

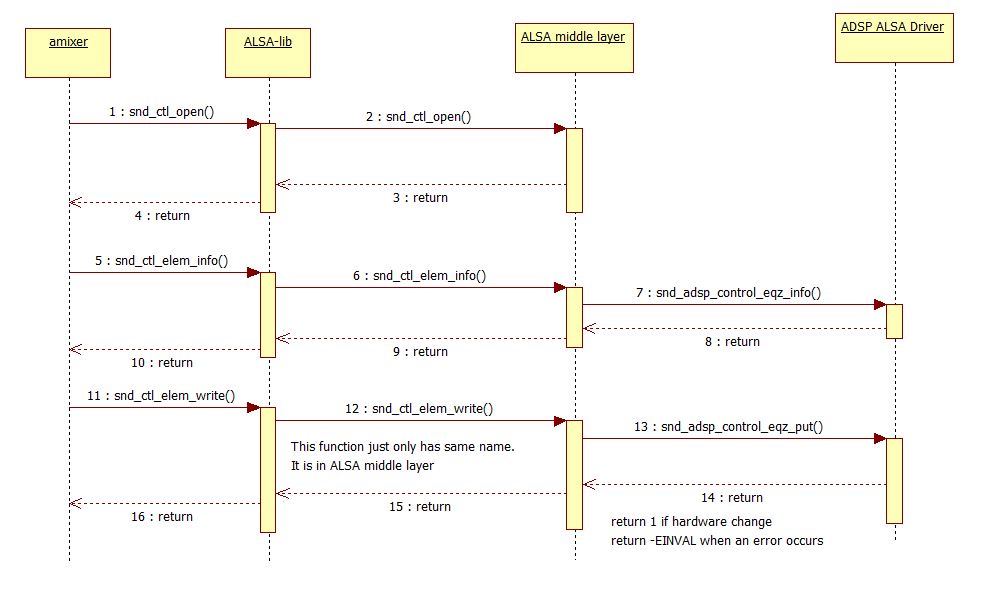


Figure 4‑16 Flow of setting Equalizer parameters of hardware

## MIX Control Flow

This flow describes how MIX control flag is raised. In playback/capture case, from the second capture/playback on, MIX control flag is raised when the CPU DAI for it is registered.

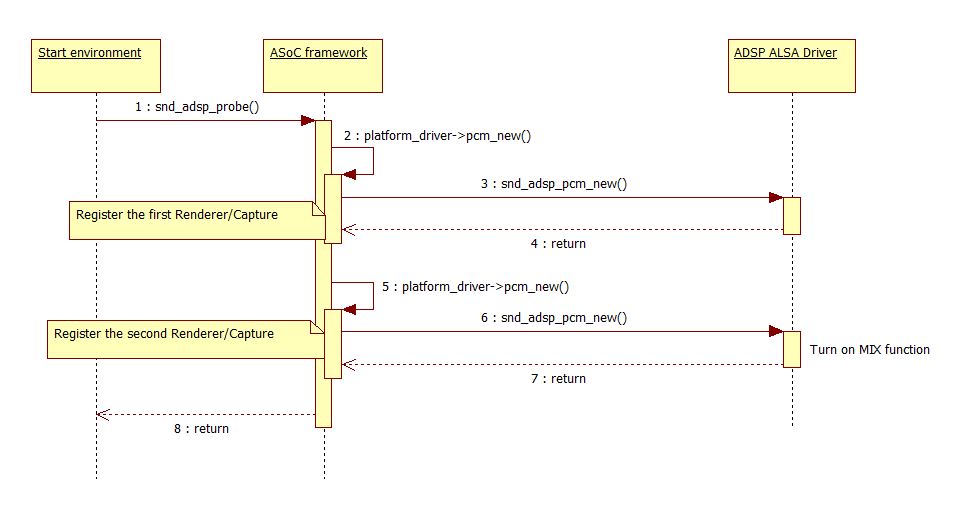


Figure 4‑17 Flow of activating MIX control usage

# List of Usage

This section is to help user understand the usage of the ADSP ALSA interface.

Below table show target platforms support for each use case.

|  |  |  |
| --- | --- | --- |
| Use case | Chip | Board |
| Playback/Capture | H3/M3/M3N/E3 | Salvator, Ebisu board |
| TDM Playback/ TDM Capture | H3/M3 | Starter KIT –Kingfisher board |

Table 5‑1 Target environment for each use case.

[Note] ADSP sound does not support to handle XRUN state (buffer underrun/buffer overrun).

(Refer to ALSA wiki *https://alsa.opensrc.org/Xruns* for more information)

## Playback

External Memory

Speaker

SRC

DVC

SCU

SSI

CTU

ADMAC

ADMACpp

PCM data

MIX

Figure 5‑1 Data path for playback

Below table shows information of ADSP ALSA Driver sound card.

|  |  |
| --- | --- |
| Sound card | Description |
| hw:0,0,0 | ADSP ALSA sound card (card 0) with DAI 0 |
| hw:0,1,0 | ADSP ALSA sound card (card 0) with DAI 1 |
| hw:0,2,0 | ADSP ALSA sound card (card 0) with DAI 2 |
| hw:0,3,0 | ADSP ALSA sound card (card 0) with DAI 3 |

Table 5‑2 Detailed information of ADSP ALSA sound card

The sound card is configured in device tree as default card (card 0). User can run multi playback stream with different DAIs. More information about mixing multi stream, please refer 5.1.5

[Note] Codec ak4613 only supports data with 24 bit. Therefore, when playback 16-bit stream user must use “**plughw**” card instead of “**hw**” card for the conversion. And the monaural format cannot be used due to this conversion (unsupported 24-bit/monaural).

### Playback stream

ADSP ALSA Driver supports playback stream (PCM width 16/24, sample rate 32/44.1/48 KHz, frame size 1024, 1/2 channels).

* User setting:

aplay -D plughw:0,0,0 -c2 -r32000 -fS24\_3LE thetest\_FULL\_s\_32000\_24.pcm

### Setting Playback Volume

ADSP ALSA Driver supports setting volume with range from 0 to 800. Value 800 means that increase volume to 8 time.

* User setting: perform below steps to set volume
* Run stream

aplay -D plughw:0,0,0 –c2 -r48000 -fS16\_LE thetest\_FULL\_s\_48000\_16.pcm

* Set volume

amixer -c 0 cset name="PlaybackVolume",index=0 100

* Get volume information

amixer -c 0 cget name="PlaybackVolume",index=0

### Setting Output Sample Rate

ADSP ALSA Driver supports converting data’s sample rate to other value. Sample rate supported with range (32/44.1/48 kHz).

* User setting
* Set output sample rate

amixer -c 0 cset name="PlaybackOutRate",index=0 48000

* Run stream

aplay -D plughw:0,0,0 -c2 -r32000 -fS24\_3LE thetest\_FULL\_s\_32000\_24.pcm

* Get information output sample rate

amixer -c 0 cget name="PlaybackOutRate",index=0

### Setting Output Channel

ADSP ALSA Driver supports convert data’s channel number to other value as below table.

|  |  |  |  |
| --- | --- | --- | --- |
| Number | Input data | Output data | Supported |
| 1 | 16 bit & 1 channel | 16 bit & 2 channel | O |
| 2 | 16 bit & 2 channel | 16 bit & 1 channel | O |
| 3 | 24 bit & 2 channel | 24 bit & 1 channel | X |

Table 5‑3 List of channel number conversation

O means supported

X means unsupported

[Note] Due to the limitation of AK4613 which noted in 5.1, case 1 and case 2 are also unsupported.

* User setting:
* Set output channel number

amixer -c 0 cset name="PlaybackOutChannel",index=0 2

* Run stream

aplay -D plughw:0,0,0 -c2 -r48000 -fS24\_3LE thetest\_FULL\_s\_48000\_24.pcm

* Get information about channel number

amixer -c 0 cget name="PlaybackOutChannel",index=0

### MIX function

ADSP ALSA Driver supports mixing multi (2/3/4) playback stream with same sample rate.

But due to hardware performance, H3 can support mixing 2, 3 or 4 stream but M3/E3 only supports mixing 2 stream.

#### Mix 2 playback stream

SSI

MIX

ADMAC

ADMAC

ADMACpp

SRC

CTU

External Memory

SRC

CTU

External Memory

Figure 5‑2 Data path when mixing 2 streams

* User setting
* Playback 1st stream:

aplay -D plughw:0,0,0 –c2 –r44100 -fS16\_LE thetest\_FULL1\_s\_44100\_16.pcm

* Playback 2nd stream:

amixer -c 0 cset name="PlaybackOutChannel",index=1 2

amixer -c 0 cset name="PlaybackOutRate",index=1 44100

aplay -D plughw:0,1,0 –c2 -r48000 -fS16\_LE thetest\_FULL2\_s\_48000\_16.pcm

#### Mix 3 playback stream

SSI

MIX

External Memory

SRC

CTU

ADMAC

ADMAC

ADMAC

ADMACpp

External Memory

SRC

CTU

SRC

CTU

External Memory

Figure 5‑3 Data path when mixing 3 streams

* User setting
* Playback 1st stream:

aplay -D plughw:0,0,0 –c2 -r44100 -fS16\_LE thetest\_FULL1\_s\_44100\_16.pcm

* Playback 2nd stream:

amixer -c 0 cset name="PlaybackOutRate",index=1 44100

aplay -D plughw:0,1,0 –c2 -r48000 -fS16\_LE thetest\_FULL2\_s\_48000\_16.pcm

* Playback 3rd stream:

amixer -c 0 cset name="PlaybackOutRate",index=2 44100

aplay -D plughw:0,2,0 –c2 -r32000 -fS16\_LE thetest\_FULL3\_s\_32000\_16.pcm

#### Mix 4 playback stream

SSI

MIX

External Memory

SRC

CTU

ADMAC

ADMAC

ADMAC

ADMAC

ADMACpp

SRC

CTU

External Memory

External Memory

SRC

CTU

SRC

CTU

External Memory

Figure 5‑4 Data path when mixing 4 streams

* User setting
* Playback 1st stream:

amixer -c 0 cset name="PlaybackOutRate",index=0 44100

aplay -D plughw:0,0,0 –c2 -r48000 -fS16\_LE thetest\_FULL1\_s\_48000\_16.pcm

* Playback 2nd stream:

amixer -c 0 cset name="PlaybackOutRate",index=1 44100

aplay -D plughw:0,1,0 –c2 -r48000 -fS16\_LE thetest\_FULL2\_s\_48000\_16.pcm

* Playback 3rd stream:

amixer -c 0 cset name="PlaybackOutRate",index=2 44100

aplay -D plughw:0,2,0 –c2 -r32000 -fS16\_LE thetest\_FULL3\_s\_32000\_16.pcm

* Playback 4th stream:

amixer -c 0 cset name="PlaybackOutRate",index=3 44100

aplay -D plughw:0,3,0 -c2 -r32000 -fS16\_LE thetest\_FULL4\_s\_32000\_16.pcm

## Capture



SSI

ADMACpp

External Memory

SRC

ADMACpp

DVC

Figure 5‑5 Data path for capture stream

Below table shows information of ADSP ALSA Driver sound card.

|  |  |
| --- | --- |
| Sound card | Description |
| hw:0,0,0 | ADSP ALSA sound card (card 0) with DAI 0 |
| hw:0,1,0 | ADSP ALSA sound card (card 0) with DAI 1 |
| hw:0,2,0 | ADSP ALSA sound card (card 0) with DAI 2 |
| hw:0,3,0 | ADSP ALSA sound card (card 0) with DAI 3 |

Table 5‑4 Detailed information of ADSP ALSA sound card

The sound card is configured in device tree as default card (card 0).

[Note] Codec ak4613 only supports data with 24 bit. Therefore, when playback 16-bit stream user must use “**plughw**” card instead of “**hw**” card for the conversion. And the monaural format cannot be used due to this conversion (unsupported 24-bit/monaural).

### Capture stream

ADSP ALSA Driver supports recording stream (PCM width 16/24, sample rate 32/44.1/48 KHz, channel 1/2).

* User setting:

arecord -D plughw:0,3,0 -c2 -r32000 -fS16\_LE -d 5 -t raw thetest\_FULL\_s\_32000\_16.pcm

### Setting Capture Volume

ADSP ALSA Driver supports setting volume for record stream. Value range from 0 to 800. Value 800 means that increase volume to 8 times.

* User setting:
* Record stream

arecord -D plughw:0,0,0 -c2 -r32000 -fS16\_LE -d 5 -t raw thetest\_FULL\_s\_32000\_16.pcm

* Set volume

amixer -c 0 cset name="CaptureVolume",index=0 50

* Get information about volume

amixer -c 0 cget name="CaptureVolume",index=0

### Setting Input Sample Rate

ADSP ALSA Driver supports converting data’s sample rate to value. Sample rate supported with range (32/44.1/48 KHz).

* User setting
* Set input sample rate

amixer -c 0 cset name="CaptureInRate",index=0 44100

* Record stream

arecord -D plughw:0,0,0 -c2 -r48000 -fS16\_LE -d 5 -t raw thetest\_FULL\_s\_48000\_16.pcm

* Get information about input sample rate

amixer -c 0 cget name="CaptureInRate",index=0

## TDM Playback

External Memory

Speaker

SRC

DVC

SCU

SSI

ADMAC

ADMACpp

PCM data ( 6/8 channel)

Figure 5‑6 Data path for multichannel

Command is used when running aplay to play TDM stream:

# aplay -D plughw:0,0,0 –c<value> -r<value> -f<name> <input>

Explanation:

-D plughw:0,0,0: Card selected is 0, DAI index is 0, sub-device is 0

-c<value>: Channel number (6/8)

-r<value>: Sampling rate (32000/44100/48000)

-f<name>: Format of PCM width

<input>: input file (.pcm, .wav)

### TDM Playback stream

ADSP ALSA Driver supports run multichannel stream (PCM width 16/24, 6/8 channels, sample rate 44.1/48 kHz). If user run a stream 32 kHz, must convert sample rate to 44.1/48 kHz, please refer 5.3.3.

[Note] Codec pcm3168 only supports sample rate 48 kHz. So if working on codec pcm3168, user must convert data sample rate to 48kHz before through codec.

* User setting
* Set output sample rate if it is different from 48kHz

amixer -c 0 cset name="TDMPlaybackOutRate" 48000

* Playback multichannel stream

aplay -D plughw:0,0,0 -c6 -r32000 -fS24\_3LE thetest\_FULL\_6ch\_32000\_24.pcm

* Get information about sample rate

amixer -c 0 cget name="TDMPlaybackOutRate"

### Setting TDM Playback Volume

ADSP ALSA Driver supports setting volume for TDM Playback stream. Value range from 0 to 800. But updating volume runtime is unsupported. So user needs to set volume value before running multichannel stream.

* User setting
* Set volume

amixer -c 0 cset name="TDMPlaybackVolume" 100

* Playback multichannel stream

aplay -D plughw:0,0,0 –c6 -r48000 -fS16\_LE thetest\_FULL\_6ch\_48000\_16.pcm

* Get information about volume

amixer -c 0 cget name="TDMPlaybackVolume"

### Setting TDM Output Sample Rate

ADSP ALSA Driver supports convert data’s sample rate to other value. Range of output sample rate supported (44.1/48 kHz).

* User setting
* Set output sample rate

amixer -c 0 cset name="TDMPlaybackOutRate" 48000

* Run stream

aplay -D plughw:0,0,0 –c6 -r44100 -fS16\_LE thetest\_FULL\_6ch\_44100\_16.pcm

* Get information output sample rate

amixer -c 0 cget name="TDMPlaybackOutRate"

## TDM Capture

External Memory

Microphone

SRC

DVC

SCU

SSI

ADMACpp

ADMACpp

PCM data (6/8 channel)



Figure 5‑7 Data path for recording multichannel stream

Command are used when running arecord to record multichannel stream:

arecord -D plughw:0,0,0 –c<value> -r<value> -f<name> -d <value> -t raw <output>

Explanation:

-D plughw:0,0,0: Card selected is 0, DAI index is 0, sub-device is 0

-c<value>: Channel number (6/8)

-r<value>: Sampling rate (32000/44100/48000)

-f<name>: Format of PCM width

-d <value>: Recording duration (second)

-t raw <output>: Output file is raw type (.pcm)

### TDM Capture stream

ADSP ALSA Driver supports recording multichannel stream (PCM width 16/24, sample rate 32/44.1/48 KHz, channel 6/8). If user record a stream 32 kHz, must convert input sample rate to 44.1/48 kHz, please refer 5.4.3.

[Note] Codec pcm3168 only supports sample rate 48 kHz. So if working on codec pcm3168, user must convert input sample rate to 48kHz.

* User setting

arecord -D plughw:0,0,0 -c8 -r48000 -fS16\_LE -d 15 -t raw output.pcm

### Setting TDM Capture Volume

ADSP ALSA Driver supports setting volume for record multichannel stream. Value range from 0 to 800. However, updating volume runtime is unsupported. Value 800 means that increase volume to 8 times.

* User setting
* Set volume

amixer -c 0 cset name="TDMCaptureVolume" 100

* Record multichannel stream

arecord -D plughw:0,0,0 -c8 -r48000 -fS16\_LE -d 15 -t raw out.pcm

* Get information about volume

amixer -c 0 cget name="TDMCaptureVolume"

### Setting TDM Input Sample Rate

ADSP ALSA Driver supports convert input sample rate to other value. Range of input sample rate supported (44.1/48 kHz).

* User setting
* Set input sample rate

amixer -c 0 cset name="TDMCaptureInRate" 48000

* Record stream

arecord -D plughw:0,0,0 -c8 -r44100 -fS16\_LE -d 15 -t raw output.pcm

* Get information input sample rate

amixer -c 0 cget name="TDMCaptureInRate"

## Equalizer

* User setting parameters for Parametric Equalizer:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Parameter | Data format | Range | Step | Description |
| Type | 32-bit integer | 0 to 3  T: Through  P: Peaking  B: Bass  R: Treble | 1 | Specify filter type of one filter. |
| Fc[Hz] | 32-bit integer | It is specified with respect to each filter type. | 1Hz | Specify center frequency of a peaking filter. |
| Gain | Fixed point decimal (Q4.28) | -15dB to 15dB | 0.125dB | Specify gain at a center frequency of a peaking filter. |
| Base Gain | Fixed point decimal (Q4.28) | -10dB to 10dB | 0.125dB | Specify a base gain.  It is used for Bass/Treble filter and it is ignored for Peaking filter.  Summed gain of Gain and Base Gain do not have to exceed -15～15dB. |
| Q | Fixed point decimal (Q5.27) | 0.2 to 15 | 0.1 | Specify band width of a peaking/notch filter |

Table 5‑5 User setting parameters for Parametric Equalizer

* User setting parameters for Graphic Equalizer:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Parameter | Data format | Range | Step | Description |
| Fs[Hz] | 32-bit integer | 48kHz  (44.1kHz)  (32kHz) | 1Hz | Specify sampling frequency of input signal. |
| Gain | Fixed point decimal (Q4.28) | -10dB to 10dB | 0.125dB | Specify gain at a center frequency of a peaking/notch filter. |
| Channel | 32-bit integer | 0 to 1 | 1 | Specify which channel to set. |
| Band | 32-bit integer | 0 to 4 | 1 | Specify which band to set |

Table 5‑6 User setting parameters for Graphic Equalizer

### Equalizer for Playback

ADSP ALSA Driver supports setting Parametric Equalizer and Graphic Equalizer for playback stream.

Equalizer plugin does not support setting in runtime.

#### Setting Parametric Equalizer

* Setting flow
* Enable Equalizer control

amixer -c 0 cset name="PlaybackEQZSwitch",index=3 1

index=3: DAI index 3

Value 1: enable Equalizer control; 0: disable Equalizer control

* Set Parametric Equalizer

./aeqz -c 0 cset name="PlaybackEQZControl",index=3 parametric\_config.txt

Index=3: DAI index 3

Content of parametric\_config.txt

Parametric

T 15000 0.707 0 0

T 15000 0.707 0 0

T 15000 0.707 0 0

T 15000 0.707 0 0

T 15000 0.707 0 0

T 15000 0.707 0 0

T 15000 0.707 0 0

T 15000 0.707 0 0

T 15000 0.707 0 0

[Note]

1st column selects filter type. User can select filter types (T: Through, P: Peak, B: Bass, R: Treble)

2nd column selects frequency center (Fc)

3rd column selects bandwidth (Q). Use unit decimal.

4th column selects gain, 5th column selects base gain. Use number format decimal.

Each line corresponds with parameters of a filter

* Playback stream:

aplay -D plughw:0,3,0 -c2 -r48000 -fS24\_3LE thetest\_FULL\_s\_48000\_24.pcm

* Get information of Equalizer parameter:

./aeqz -c 0 cget name="PlaybackEQZControl",index=3 out\_config.txt

Information of Equalizer parameter will be recorded in out\_config.txt similar with below file example.

Parametric

T 15000 94891933 268435456 268435456

T 15000 94891933 268435456 268435456

T 15000 94891933 268435456 268435456

T 15000 94891933 268435456 268435456

T 15000 94891933 268435456 268435456

T 15000 94891933 268435456 268435456

T 15000 94891933 268435456 268435456

T 15000 94891933 268435456 268435456

T 15000 94891933 268435456 268435456

1st, 2nd column same with content in parametric\_config.txt

3rd column presents bandwidth (Q) same with parametric\_config.txt but number format is fixed point Q5.27.

4th, 5th column presents gain and base gain same with parametric\_config.txt but number format is fixed point Q4.28.

* Get information of Equalizer status:

amixer -c 0 cget name="PlaybackEQZSwitch",index=3

#### Setting Graphic Equalizer

* Setting flow
* Enable Equalizer control

amixer -c 0 cset name="PlaybackEQZSwitch",index=3 1

index=3: DAI index 3

Value 1: enable Equalizer control; 0: disable Equalizer control

* Set Graphic Equalizer parameter:

./aeqz -c 0 cset name="PlaybackEQZControl",index=3 graphic\_config.txt

index=3: DAI index 3

Content of graphic\_config.txt

Graphic

-10.0

-10.0

-10.0

-10.0

-10.0

[Note]

-10.0 means gain value (decimal)

Each line corresponds with gain of a band

* Playback stream:

aplay -D plughw:0,3,0 -c2 -r48000 -fS24\_3LE thetest\_FULL\_s\_48000\_24.pcm

* Get information of Equalizer parameter:

./aeqz -c 0 cget name="PlaybackEQZControl",index=3 out\_config.txt

Information of Equalizer parameter will be recorded in out\_config.txt similar with below file example.

Graphic

84886744

84886744

84886744

84886744

84886744

Value 84886744 presents gain same with graphic\_config.txt but number format is fixed point Q4.28.

* Get information of Equalizer status:

amixer -c 0 cget name="PlaybackEQZSwitch",index=3

### Equalizer for Capture

ADSP ALSA Driver supports setting Parametric Equalizer and Graphic Equalizer for record stream.

Equalizer plugin does not support setting in runtime.

#### Setting Parametric Equalizer

* Setting flow
* Enable Equalizer control

amixer -c 0 cset name="CaptureEQZSwitch",index=3 1

index=3: DAI index 3

Value 1: enable Equalizer control; 0: disable Equalizer control

* Set Parametric Equalizer

./aeqz -c 0 cset name="CaptureEQZControl",index=3 parametric\_config.txt

index=3: DAI index 3

Content of parametric\_config.txt

Parametric

T 15000 0.707 0 0

T 15000 0.707 0 0

T 15000 0.707 0 0

T 15000 0.707 0 0

T 15000 0.707 0 0

T 15000 0.707 0 0

T 15000 0.707 0 0

T 15000 0.707 0 0

T 15000 0.707 0 0

[Note]

1st column selects filter type. User can select filter types (T: Through, P: Peak, B: Bass, R: Treble)

2nd column selects frequency center (Fc)

3rd column selects bandwidth (Q). Use unit decimal.

4th column selects gain, 5th column selects base gain. Use number format decimal.

Each line corresponds with parameters of a filter

* Record stream:

arecord -D plughw:0,3,0 -c2 -r32000 -fS16\_LE -d 5 -t raw thetest\_FULL\_s\_32000\_16.pcm

* Get information of Equalizer parameter:

./aeqz -c 0 cget name="CaptureEQZControl",index=3 out\_config.txt

Information of Equalizer parameter will be recorded in out\_config.txt similar with below file example.

Parametric

T 15000 94891933 268435456 268435456

T 15000 94891933 268435456 268435456

T 15000 94891933 268435456 268435456

T 15000 94891933 268435456 268435456

T 15000 94891933 268435456 268435456

T 15000 94891933 268435456 268435456

T 15000 94891933 268435456 268435456

T 15000 94891933 268435456 268435456

T 15000 94891933 268435456 268435456

1st, 2nd column same with content in parametric\_config.txt

3rd column presents bandwidth (Q) same with parametric\_config.txt but number format is fixed point Q5.27.

4th, 5th column presents gain and base gain same with parametric\_config.txt but number format is fixed point Q4.28.

* Get information of Equalizer status:

amixer -c 0 cget name="CaptureEQZSwitch",index=3

#### Setting Graphic Equalizer

* Setting flow
* Enable Equalizer control

amixer -c 0 cset name="PlaybackEQZSwitch",index=3 1

index=3: DAI index 3

Value 1 to enable, 0 to disable Equalizer control

* Set Graphic Equalizer parameter:

./aeqz -c 0 cset name="PlaybackEQZControl",index=3 graphic\_config.txt

index=3: DAI index 3

Content of graphic\_config.txt

Graphic

-10.0

-10.0

-10.0

-10.0

-10.0

[Note]

-10.0 means gain value (decimal)

Each line corresponds with a band

* Record stream:

arecord -D plughw:0,3,0 -c2 -r32000 -fS16\_LE -d 5 -t raw thetest\_FULL\_s\_32000\_16.pcm

* Get information of Equalizer parameter:

./aeqz -c 0 cget name="PlaybackEQZControl",index=3 out\_config.txt

Information of Equalizer parameter will be recorded in out\_config.txt similar with below file example.

Graphic

84886744

84886744

84886744

84886744

84886744

Value 84886744 presents gain of band same with graphic\_config.txt but number format is fixed point Q4.28.

* Get information of Equalizer status:

amixer -c 0 cget name="CaptureEQZSwitch",index=3

# Appendix

## Error code

Below table shows error types and corresponding value of callback functions in ALSA Interface

Table 6‑1 Error code for ALSA callback functions

|  |  |  |
| --- | --- | --- |
| Error code | Description | Reference |
| EINVAL | Invalid argument or some functions get failed | https://elixir.free-electrons.com/linux/v4.0/source/include/uapi/asm-generic/errno-base.h |
| ENOMEM | Cannot allocate memory |
| ENODEV | Invalid driver data in platform device |

## Structure and type definitions

Below tables shows structures and type definitions used in this material.

Table 6‑2 Structures or type definition are defined in ALSA middle layer.

|  |  |  |
| --- | --- | --- |
| Structure | Description | Reference |
| snd\_pcm\_substreams | It contains elements of PCM substream’s information, only one of which is used for above callback functions – runtime object. This object contains PCM parameters (pcm width, channel, sample rate, buffer size). Other elements are used by PCM middle layer. | *https://elixir.free-electrons.com/linux/latest/source/include/sound/pcm.h* |
| snd\_kcontrol | It is used to communicate a control of user space with the control interface on kernel. It contains control index, elements point the get, set, info callback inside the control. | *https://elixir.free-electrons.com/linux/latest/source/include/sound/control.h* |
| snd\_ctl\_elem\_value | It contains value to set parameter to ADSP or get parameter from ADSP. | *http://elixir.free-electrons.com/linux/v4.2/source/include/uapi/sound/asound.h* |
| snd\_ctl\_elem\_info | It contains detail information on the control. | *http://elixir.free-electrons.com/linux/v4.3/source/include/uapi/sound/asound.h* |
| hrtimer | It contains detail information of high-resolution timer. | *https://elixir.bootlin.com/linux/latest/source/include/linux/hrtimer.h* |
| spinlock\_t | It contains detail information of spinlock. | *https://elixir.bootlin.com/linux/latest/source/include/linux/spinlock\_types.h* |
| snd\_pcm\_indirect | It contains detail information for indirect PCM data transfer | *https://elixir.bootlin.com/linux/latest/source/include/sound/pcm-indirect.h* |
| ktime\_t | This is the data type for kernel time | *https://elixir.bootlin.com/linux/v4.17.1/source/include/linux/ktime.h#L28* |

Table 6-3 shows list of structures are defined in ADSP ALSA Driver.

Table 6‑3 Structures defined in ADSP ALSA Driver

|  |  |  |
| --- | --- | --- |
| Structures | Size (bytes) | Description |
| snd\_adsp\_control | 1860 | It is used to store parameters from use |
| snd\_adsp\_card | 1944 | It is used to store data for ALSA sound card |
| snd\_adsp\_base\_info | 200 | It is used to store base data for ADSP sound card |
| snd\_adsp\_playback | 224 | It is used to store necessary information for Renderer |
| snd\_adsp\_record | 224 | It is used to store necessary information for Capture |
| snd\_adsp\_tdm\_playback | 216 | It is used to store necessary information for TDM Renderer |
| snd\_adsp\_tdm\_record | 216 | It is used to store necessary information for TDM Capture |

### snd\_adsp\_control structure

Table 6‑4 snd\_adsp\_control structure information

|  |  |  |
| --- | --- | --- |
| Member name | | Outline |
| int | vol\_rate[DIRECT\_NUM] [MAX\_DAI\_IDX – 1] | Volume rate value for Capture/Renderer |
| int | tdm\_vol\_rate[DIRECT\_NUM] | Volume rate for TDM Capture/TDM Renderer |
| int | sample\_rate[DIRECT\_NUM] [MAX\_DAI\_IDX – 1] | Out sample rate with Renderer, in sample rate with Capture |
| int | tdm\_sample\_rate[DIRECT\_NUM] | Out sample rate with TDM Renderer, in sample rate with TDM Capture |
| int | rdr\_out\_ch[MAX\_DAI\_IDX – 1] | Output channel of Renderer |
| struct xf\_adsp\_equalizer\_params | eqz\_params[DIRECT\_NUM] [MAX\_DAI\_IDX – 1] | Equalizer parameters |
| int | eqz\_switch[DIRECT\_NUM] [MAX\_DAI\_IDX – 1] | Equalizer switch |
| int | mix\_usage | MIX control |

[Note] DIRECT\_NUM is a macro representing the number of streams that can execute concurrently. In this material, DIRECT\_NUM is 2, which means it supports running a playback stream and a capture stream concurrently.

MAX\_DAI\_IDX is a macro representing the number of stream types. In this material, MAX\_DAI\_IDX is 5, which means it supports 4 playback/record streams and 1 TDM playback/TDM record stream.

mix\_usage is controlled by ADSP ALSA driver, not by control interface.

### snd\_adsp\_base\_info structure

Table 6‑5 snd\_adsp\_base\_info structure information

|  |  |  |
| --- | --- | --- |
| Member name | | Outline |
| struct hrtimer | hrtimer | High resolution timer data |
| ktime\_t | ktime | Kernel time value in nanosecond |
| int | hrt\_state | High resolution timer state |
| int | handle\_id | Target handle ID of ALSA driver |
| char | \*buffer[XF\_BUF\_POOL\_SIZE] | Data buffer |
| int | buf\_bytes | Size of each allocated data buffer |
| int | buf\_idx | Index of data buffer |
| int | buf\_queue | Number of data buffer in the queue |
| int | hw\_idx | Hardware index in bytes |
| int | period\_bytes | Number of bytes in one period |
| struct snd\_pcm\_substream | \*substream | Substream runtime object |
| spinlock\_t | lock | Spinlock data |
| struct snd\_pcm\_indirect | pcm\_indirect | Indirect PCM data transfer |
| int | runtime\_err | Runtime error indicator |

[Note] XF\_BUF\_POOL\_SIZE is a macro representing number of buffer in a data pool. In this material XF\_BUF\_POOL\_SIZE is 4.

### snd\_adsp\_playback structure

Table 6‑6 snd\_adsp\_playback structure information

|  |  |  |
| --- | --- | --- |
| Member name | | Outline |
| struct snd\_adsp\_base\_info | base | Base information of stream |
| struct xf\_adsp\_renderer | \*renderer | Renderer component’s data |
| struct xf\_adsp\_equalizer | \*equalizer | Equalizer component’s data |
| int | rdr\_state | Renderer component’s state |
| int | eqz\_state | Equalizer component’s state |

### snd\_adsp\_record structure

Table 6‑7 snd\_adsp\_record structure information

|  |  |  |
| --- | --- | --- |
| Member name | | Outline |
| struct snd\_adsp\_base\_info | base | Base information of stream |
| struct xf\_adsp\_capture | \*capture | Capture component’s data |
| struct xf\_adsp\_equalizer | \*equalizer | Equalizer component’s data |
| int | cap\_state | Capture component’s state |
| int | eqz\_state | Equalizer component’s state |

### snd\_adsp\_card structure

Table 6‑8 snd\_adsp\_card structure information

|  |  |  |
| --- | --- | --- |
| Member name | | Outline |
| struct snd\_adsp\_playback | \*playback[MAX\_DAI\_IDX – 1] | Playback data |
| struct snd\_adsp\_record | \*record[MAX\_DAI\_IDX – 1] | Record data |
| struct snd\_adsp\_tdm\_playback | \*tdm\_playback | TDM playback data |
| struct snd\_adsp\_tdm\_record | \*tdm\_record | TDM record data |
| struct snd\_adsp\_control | ctr\_if | Structure containing parameter information for control |

### snd\_adsp\_tdm\_playback structure

Table 6‑9 snd\_adsp\_tdm\_playback structure information

|  |  |  |
| --- | --- | --- |
| Member name | | Outline |
| struct snd\_adsp\_base\_info | base | Base information of stream |
| struct xf\_adsp\_tdm\_renderer | \*tdm\_renderer | TDM Renderer component’s data |
| int | state | TDM Renderer component’s state |

### snd\_adsp\_tdm\_record structure

Table 6‑10 snd\_adsp\_tdm\_record structure information

|  |  |  |
| --- | --- | --- |
| Member name | | Outline |
| struct snd\_adsp\_base\_info | base | Base information of stream |
| struct xf\_adsp\_tdm\_capture | \*tdm\_capture | TDM Capture component’s data |
| int | state | TDM Capture component’s state |