- Table of Contents -

[1. Overview 4](#_Toc530586527)

[1.1. Overview of this document. 4](#_Toc530586528)

[1.2. The architecture of the Software and scope of this document 4](#_Toc530586529)

[1.3. Software necessary to be prepared in advance 4](#_Toc530586530)

[1.4. Related documents 5](#_Toc530586531)

[1.5. The definition of common types 5](#_Toc530586532)

[2. Software specification 6](#_Toc530586533)

[2.1. The list of functions 6](#_Toc530586534)

[2.2. The list of structures 7](#_Toc530586535)

[2.3. Function specification 8](#_Toc530586536)

[2.3.1. xf\_proxy\_init 8](#_Toc530586537)

[2.3.2. xf\_proxy\_close 8](#_Toc530586538)

[2.3.3. xf\_proxy\_pool 8](#_Toc530586539)

[2.3.4. xf\_pool\_alloc 9](#_Toc530586540)

[2.3.5. xf\_pool\_free 9](#_Toc530586541)

[2.3.6. xf\_buffer\_get 9](#_Toc530586542)

[2.3.7. xf\_buffer\_put 10](#_Toc530586543)

[2.3.8. xf\_buffer\_data 10](#_Toc530586544)

[2.3.9. xf\_buffer\_length 10](#_Toc530586545)

[2.3.10. xf\_open 10](#_Toc530586546)

[2.3.11. xf\_close 11](#_Toc530586547)

[2.3.12. xf\_handle\_aux 11](#_Toc530586548)

[2.3.13. xf\_handle\_auxlen 11](#_Toc530586549)

[2.3.14. xf\_route 12](#_Toc530586550)

[2.3.15. xf\_unroute 12](#_Toc530586551)

[2.3.16. xf\_command 13](#_Toc530586552)

[2.3.17. xf\_response\_put 13](#_Toc530586553)

[2.3.18. xf\_response\_get 13](#_Toc530586554)

[2.4. Callback function specification 14](#_Toc530586555)

[2.4.1. xf\_response\_cb 14](#_Toc530586556)

[2.5. Structures specification 15](#_Toc530586557)

[2.5.1. xf\_proxy\_t 15](#_Toc530586558)

[2.5.2. xf\_handle\_t 15](#_Toc530586559)

[2.5.3. xf\_pool\_t 15](#_Toc530586560)

[2.5.4. xf\_buffer\_t 15](#_Toc530586561)

[2.5.5. xf\_user\_msg\_t 16](#_Toc530586562)

[3. Process sequence 17](#_Toc530586563)

[3.1. Initialization flow 17](#_Toc530586564)

[3.2. Flow of sending a command 18](#_Toc530586565)

[3.3. Flow to allocate input / output buffer 19](#_Toc530586566)

[3.4. Flow of send / receive input buffer 20](#_Toc530586567)

[3.5. Flow to send / receive output buffer 21](#_Toc530586568)

[3.6. Flow to release input / output buffer 22](#_Toc530586569)

[3.7. Termination flow 23](#_Toc530586570)

[4. Notes 24](#_Toc530586571)

[4.1. Function Call 24](#_Toc530586572)

[4.2. Other notes 24](#_Toc530586573)

[4.2.1. Allocation of memory 24](#_Toc530586574)

[4.2.2. Out of range memory access 24](#_Toc530586575)

[4.2.3. Combination with other applications 24](#_Toc530586576)

[4.2.4. Monitoring on Performance 24](#_Toc530586577)

[5. Appendix 25](#_Toc530586578)

[5.1. Standard Use Case: Playback 25](#_Toc530586579)

[5.1.1. Playback flow 1: Normal 27](#_Toc530586580)

[5.1.2. Playback flow 2: Sampling Rate Conversion 27](#_Toc530586581)

[5.1.3. Playback flow 3: Channel Transfer 29](#_Toc530586582)

[5.1.4. Playback flow 4: Volume Control 30](#_Toc530586583)

[5.1.5. Playback flow 5: MIX function 31](#_Toc530586584)

[5.2. Standard Use Case: Record 35](#_Toc530586585)

[5.2.1. Capture flow 1: Normal 37](#_Toc530586586)

[5.2.2. Capture flow 2: Sampling Rate Conversion 37](#_Toc530586587)

[5.2.3. Capture flow 3: Volume Control 38](#_Toc530586588)

[5.3. Standard Use Case: Equalizer 39](#_Toc530586589)

[5.4. Standard Use Case: Route 40](#_Toc530586590)

[5.4.1. Route flow 1: Capture – Equalizer 40](#_Toc530586591)

[5.4.2. Route flow 2: Equalizer – Renderer 41](#_Toc530586592)

[5.4.3. Route flow 3: Capture – Equalizer – Renderer 42](#_Toc530586593)

[5.4.4. Route flow 4: Capture – Renderer 43](#_Toc530586594)

[5.4.5. Route flow 5: Equalizer – Renderer use MIX function 44](#_Toc530586595)

[5.5. Standard Use Case: TDM 45](#_Toc530586596)

[5.5.1. TDM Renderer flow 1: Normal 49](#_Toc530586597)

[5.5.2. TDM Renderer flow 2: Sampling Rate Conversion 49](#_Toc530586598)

[5.5.3. TDM Renderer flow 3: Volume Control 50](#_Toc530586599)

[5.5.4. TDM Capture flow 1: Normal 50](#_Toc530586600)

[5.5.5. TDM Capture flow 2: Sampling Rate Conversion 51](#_Toc530586601)

[5.5.6. TDM Capture flow 3: Volume Control 52](#_Toc530586602)

- List of Figures -

[Figure 1‑1 The software architecture 4](#_Toc530586603)

[Figure 3‑1 Initialization flow 17](#_Toc530586604)

[Figure 3‑2 The standard flow to send / receive a command 18](#_Toc530586605)

[Figure 3‑3 The standard flow to allocate input / output buffer 19](#_Toc530586606)

[Figure 3‑4 The standard flow to send / receive data 20](#_Toc530586607)

[Figure 3‑5 The flow to send / receive output buffer 21](#_Toc530586608)

[Figure 3‑6 The standard flow to release input / output buffer 22](#_Toc530586609)

[Figure 3‑7 Termination flow 23](#_Toc530586610)

[Figure 5‑1 Audio data path for renderer function 25](#_Toc530586611)

[Figure 5‑2 Data path for payback normal 27](#_Toc530586612)

[Figure 5‑3 Data path for playback with sampling rate conversion in R-Car H3/M3/M3N 27](#_Toc530586613)

[Figure 5‑4 Data path for playback with sampling rate conversion in R-Car E3 28](#_Toc530586614)

[Figure 5‑5 Data path for playback with channel transfer in R-Car H3/M3/M3N 29](#_Toc530586615)

[Figure 5‑6 Data path for playback with channel transfer in R-Car E3 29](#_Toc530586616)

[Figure 5‑7 Data path for playback with volume control in R-Car H3/M3/M3N 30](#_Toc530586617)

[Figure 5‑8 Data path for playback with volume control in R-Car E3 30](#_Toc530586618)

[Figure 5‑9 Data path for playback MIX 2 streams into 1 stream in R-Car H3/M3/M3N 32](#_Toc530586619)

[Figure 5‑10 Data path for playback MIX 2 streams into 1 stream in R-Car E3 32](#_Toc530586620)

[Figure 5‑11 Data path for playback MIX 3 streams into 1 stream in R-Car H3/M3/M3N 33](#_Toc530586621)

[Figure 5‑12 Data path for playback MIX 3 streams into 1 stream in R-Car E3 33](#_Toc530586622)

[Figure 5‑13 Data path for playback MIX 4 streams into 1 stream in R-Car H3/M3/M3N 34](#_Toc530586623)

[Figure 5‑14 Data path for playback MIX 4 streams into 1 stream in R-Car E3 34](#_Toc530586624)

[Figure 5‑15 Audio data path for capture function 35](#_Toc530586625)

[Figure 5‑16 Data path for case record normal 37](#_Toc530586626)

[Figure 5‑17 Data path for record sampling rate conversion in R-Car H3/M3/M3N 37](#_Toc530586627)

[Figure 5‑18 Data path for record sampling rate conversion in R-Car E3 37](#_Toc530586628)

[Figure 5‑19 Data path for record volume control in R-Car H3/M3/M3N 38](#_Toc530586629)

[Figure 5‑20 Data path for record volume control in R-Car E3 38](#_Toc530586630)

[Figure 5‑21 Data path for equalizer 39](#_Toc530586631)

[Figure 5‑22 Data path for route Capture – Equalizer 40](#_Toc530586632)

[Figure 5‑23 Data path for route Equalizer – Renderer 41](#_Toc530586633)

[Figure 5‑24 Data path for route Capture – Equalizer – Renderer 42](#_Toc530586634)

[Figure 5‑25 Data path for route Equalizer – Renderer 43](#_Toc530586635)

[Figure 5‑26 Data path for route Equalizer – Renderer use MIX function 44](#_Toc530586636)

[Figure 5‑27 Audio data path for TDM renderer function 45](#_Toc530586637)

[Figure 5‑28 Audio data path for TDM capture function 45](#_Toc530586638)

[Figure 5‑29 Data path for TDM renderer 49](#_Toc530586639)

[Figure 5‑30 Data path for TDM renderer use sample rate conversation 49](#_Toc530586640)

[Figure 5‑31 Data path for TDM renderer use volume control 50](#_Toc530586641)

[Figure 5‑32 Data path for TDM Capture nomal 50](#_Toc530586642)

[Figure 5‑33 Data path for TDM capture use sampling rate conversion 51](#_Toc530586643)

[Figure 5‑34 Data path for TDM capture use volume control 52](#_Toc530586644)

- List of Tables -

[Table 1‑1 The list of related documents 5](#_Toc530586645)

[Table 1‑2 The list of common types 5](#_Toc530586646)

[Table 2‑1 The list of functions 6](#_Toc530586647)

[Table 2‑2 The list of structures 7](#_Toc530586648)

[Table 5‑1 Target environment for each use case. 25](#_Toc530586849)

[Table 5‑2 Supported features for OMX Renderer Interface 25](#_Toc530586850)

[Table 5‑3 Supported data for playback 26](#_Toc530586851)

[Table 5‑4 Number of stream can play in MIX function 31](#_Toc530586852)

[Table 5‑5 Supported features for OMX Capture Interface 35](#_Toc530586853)

[Table 5‑6 Supported data for record 36](#_Toc530586854)

[Table 5‑7 Supported features for OMX TDM Interface 45](#_Toc530586855)

[Table 5‑8 Supported data for TDM Renderer 46](#_Toc530586856)

[Table 5‑9 Supported data for TDM Capture 46](#_Toc530586857)

RCG3AHIFL4001ZDPE

Rev. 1.00

Jul. 05, 2017

ADSP Interface for Linux User's Manual

# Overview

## Overview of this document.

In this chapter, overview of ADSP Interface for Linux is explained.

## The architecture of the Software and scope of this document

The architecture of ADSP Interface for Linux is shown in Figure 1‑1. ADSP Interface for Linux is a user space library which provides the interface to control ADSP Framework via ADSP Driver for Linux.

ARM

User Application

ADSP

Codec

DAC/ADC

ARM

Audio HW

SRC/DVC SSI/

ADMA

ADSP Interface for Linux

NC/EC

ADSP Driver for Linux

ADSP Framework

ADSP Reference Renderer Plugin

ADSP Reference Equalizer Plugin

User Space

Kernel Space

This document’s target is in side of red square.

Figure 1‑1 The software architecture

## Software necessary to be prepared in advance

ADSP Driver for Linux should be loaded in advance to use ADSP Interface for Linux.

## Related documents

Table 1‑1 shows related documents.

Table 1‑1 The list of related documents

|  |  |  |
| --- | --- | --- |
| No. | Name | Published by |
| [1] | R-Car Series, 3rd Generation User’s Manual: Hardware | Renesas Electronics Corporation |
| [2] | ADSP Framework User’s Manual | Renesas Electronics Corporation |

## The definition of common types

Table 1‑2 shows the list of type definitions used in ADSP Interface for Linux.

Table 1‑2 The list of common types

|  |  |  |  |
| --- | --- | --- | --- |
| type | size [byte] |  |  |
| s8 | 1 | signed 8 bit integer | -128 to 127 |
| s16 | 2 | signed 16 bit integer | -32768 to 32767 |
| s32 | 4 | signed 32 bit integer | -2147483648 to 2147483647 |
| u8 | 1 | unsigned 8 bit integer | 0 to 255 |
| u16 | 2 | unsigned 16 bit integer | 0 to 65535 |
| u32 | 4 | unsigned 32 bit integer | 0 to 4294967295 |

[notice] the size of a pointer depends on architecture.

# Software specification

## The list of functions

Table 2‑1 shows the functions provided by this software. See 2.3 for more detailed specification of the functions.

Table 2‑1 The list of functions

|  |  |
| --- | --- |
| name | outline |
| xf\_proxy\_init | ADSP Interface for Linux initialization |
| xf\_proxy\_close | ADSP Interface for Linux close |
| xf\_proxy\_pool | Get pointer to auxiliary shared buffer pool from ADSP Interface for Linux struct |
| xf\_pool\_alloc | Allocate buffer pool |
| xf\_pool\_free | Release buffer pool |
| xf\_buffer\_get | Get buffer from buffer pool |
| xf\_buffer\_put | Return buffer to buffer pool |
| xf\_buffer\_data | Get buffer address |
| xf\_buffer\_length | Get buffer size |
| xf\_open | Get handle of ADSP Plugin |
| xf\_close | Release handle of ADSP Plugin |
| xf\_handle\_aux | Get the address of shared buffer from the handle of ADSP Plugin |
| xf\_handle\_auxlen | Get the size of shared buffer from the handle of ADSP Plugin. |
| xf\_route | Set routing of ADSP Plugin |
| xf\_unroute | Release routing of ADSP Plugin |
| xf\_command | Send a command to ADSP Plugin |
| xf\_response\_put | Send a message to pipe |
| xf\_response\_get | Get a message from pipe |
| xf\_response\_cb | Callback function |

## The list of structures

Table 2‑2 shows the list of structures which user should allocate memory in using the software.

See 2.5 for more detailed specification of the structures.

Table 2‑2 The list of structures

|  |  |  |
| --- | --- | --- |
| name | outline | remark |
| xf\_proxy\_t | The structure of ADSP Interface for Linux | Direct access to a member of the structure is prohibited. |
| xf\_handle\_t | The handle structure of ADSP Plugin | Direct access to a member of the structure is prohibited. |
| xf\_pool\_t | The structure of buffer pool | Direct access to a member of the structure is prohibited. |
| xf\_buffer\_t | The structure of buffer | Direct access to a member of the structure is prohibited. |
| xf\_set\_param\_msg\_t | The structure of XF\_SET\_PARAM | See the manual of ADSP Framework for details |
| xf\_get\_param\_msg\_t | The structure of XF\_GET\_PARAM | See the manual of ADSP Framework for details |
| xf\_start\_msg\_t | The structure of Runtime Initialization Stage | See the manual of ADSP Framework for details |
| xf\_user\_msg\_t | The structure of user message | See the manual of ADSP Framework for details |

## Function specification

### xf\_proxy\_init

|  |  |  |
| --- | --- | --- |
| xf\_proxy\_init | | |
| Synopsis | This function initializes ADSP Interface for Linux.  It should be called once in advance to call other functions. | |
| Syntax | int xf\_proxy\_init(  xf\_proxy\_t \*proxy,  u32 core); | |
| Parameter | proxy | The pointer to ADSP Interface for Linux structure |
| core | Specify 0. |
| Return values | success | 0 |
| fail | non-zero |

### xf\_proxy\_close

|  |  |  |
| --- | --- | --- |
| xf\_proxy\_close | | |
| Synopsis | This function terminates ADSP Interface for Linux.  After calling this function, other function can’t be called until executing xf\_proxy\_init. | |
| Syntax | void xf\_proxy\_close(  xf\_proxy\_t \*proxy); | |
| Parameter | proxy | The pointer to ADSP Interface for Linux structure |
| Return value | None | |

### xf\_proxy\_pool

|  |  |  |
| --- | --- | --- |
| xf\_proxy\_pool | | |
| Synopsis | This function gets the member of auxiliary shared buffer pool from ADSP Interface for Linux structure. This function is a function-like macro. | |
| Syntax | #define xf\_proxy\_pool(proxy) ((proxy)->aux) | |
| Parameter | proxy | The pointer to ADSP Interface for Linux structure |
| Return value | The pointer to AUX shared buffer pool | |

### xf\_pool\_alloc

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| xf\_pool\_alloc | | | | |
| Synopsis | This function allocates buffer pool. | | | |
| Syntax | int xf\_pool\_alloc(  xf\_proxy\_t \*proxy,  u32 number,  u32 length,  xf\_pool\_type\_t type,  xf\_pool\_t \*\*pool); | | | |
| Parameter | proxy | The pointer to ADSP Interface for Linux structure | | |
| number | The number of buffers to allocate | | |
| length | The length of each buffer | | |
| type | The kind of buffer pool to allocate | | |
|  |  | XF\_POOL\_AUX | AUX buffer |
|  |  | XF\_POOL\_INPUT | Input buffer |
|  |  | XF\_POOL\_OUTPUT | Output buffer |
| pool | The double pointer to buffer pool to allocate. When allocating AUX buffer, specify the pointer to xf\_proxy\_pool macro. | | |
| Return value | success | 0 | | |
| fail | non-zero | | |

### xf\_pool\_free

|  |  |  |
| --- | --- | --- |
| xf\_pool\_free | | |
| Synopsis | This functions releases buffer pool | |
| Syntax | void xf\_pool\_free(  xf\_pool\_t \*pool); | |
| Parameter | pool | The pointer to buffer pool to be released. |
| Return value | None | |

### xf\_buffer\_get

|  |  |  |
| --- | --- | --- |
| xf\_buffer\_get | | |
| Synopsis | This function gets buffer from buffer pool. | |
| Syntax | xf\_buffer\_t \* xf\_buffer\_get(  xf\_pool\_t \*pool); | |
| Parameter | pool | The pointer to buffer pool from which buffer is obtained. |
| Return value | success | The address of buffer |
| fail | 0 |

### xf\_buffer\_put

|  |  |  |
| --- | --- | --- |
| xf\_buffer\_put | | |
| Synopsis | This functions returns buffer to buffer pool. | |
| Syntax | void xf\_buffer\_put(  xf\_buffer\_t \*buffer); | |
| Parameter | buffer | The pointer to the buffer to be returned |
| Return value | None | |

### xf\_buffer\_data

|  |  |  |
| --- | --- | --- |
| xf\_buffer\_data | | |
| Synopsis | This function gets buffer address. | |
| Syntax | static inline void\* xf\_buffer\_data(  xf\_buffer\_t \*buffer); | |
| Parameter | buffer | The pointer to the buffer |
| Return value | The address of the buffer | |

### xf\_buffer\_length

|  |  |  |
| --- | --- | --- |
| xf\_buffer\_length | | |
| Synopsis | This function gets buffer size. | |
| Syntax | static inline size\_t xf\_buffer\_length(  xf\_buffer\_t \*buffer); | |
| Parameter | buffer | The pointer to the buffer |
| Return value | The size of the buffer | |

### xf\_open

|  |  |  |
| --- | --- | --- |
| xf\_open | | |
| Synopsis | This function gets the handle of ADSP Plugin. | |
| Syntax | int xf\_open(xf\_proxy\_t \*proxy,  xf\_handle\_t \*handle,  xf\_id\_t id,  u32 core,  xf\_response\_cb response); | |
| Parameter | proxy | The pointer to ADSP Interface for Linux structure |
| handle | The pointer to ADSP Plugin handle structure |
| id | The component ID of the plugin. |
| core | Specify 0 |
| response | The pointer to the callback function |
| Return value | success | 0 |
| fail | non-zero |

### xf\_close

|  |  |  |
| --- | --- | --- |
| xf\_close | | |
| Synopsis | This function releases the handle of ADSP Plugin. | |
| Syntax | void xf\_close(  xf\_handle\_t \*handle); | |
| Parameter | handle | The pointer to ADSP Plugin handle structure |
| Return value | None | |

### xf\_handle\_aux

|  |  |  |
| --- | --- | --- |
| xf\_handle\_aux | | |
| Synopsis | This function gets the address of shared buffer from ADSP Plugin handle. | |
| Syntax | static inline void\* xf\_handle\_aux(  xf\_handle\_t \*handle); | |
| Parameter | handle | The pointer to ADSP Plugin handle structure |
| Return value | The address of shared buffer | |

### xf\_handle\_auxlen

|  |  |  |
| --- | --- | --- |
| xf\_handle\_auxlen | | |
| Synopsis | This function gets the size of shared buffer from ADSP Plugin handle. | |
| Syntax | static inline size\_t xf\_handle\_auxlen(  xf\_handle\_t \*handle); | |
| Parameter | handle | The pointer to ADSP Plugin handle structure |
| Return value | The size of shared buffer | |

### xf\_route

|  |  |  |
| --- | --- | --- |
| xf\_route | | |
| Synopsis | This function connects the ports between 2 ADSP Plugins.  It enables to use the data of source ADSP Plugin as the input of target ADSP Plugin.  It is impossible to connect one port to multiple ports. | |
| Syntax | int xf\_route(xf\_handle\_t \*src,  u32 src\_port,  xf\_handle\_t \*dst,  u32 dst\_port,  u32 num,  u32 size,  u32 align); | |
| Parameter | src | The pointer of source ADSP Plugin handle structure |
| src\_port | The source port |
| dst | The pointer of target ADSP Plugin handle structure |
| dst\_port | The target port |
| num | The number of buffers allocated between the ports |
| size | The size of buffers allocated between the ports |
| align | The align of buffers allocated between the ports |
| Return value | success | 0 |
| fail | non-zero |

### xf\_unroute

|  |  |  |
| --- | --- | --- |
| xf\_unroute | | |
| Synopsis | This function disconnects the connection between the ports of 2 ADSP Plugins. | |
| Syntax | int xf\_unroute(xf\_handle\_t \*src,  u32 src\_port); | |
| Parameter | src | The pointer of source ADSP Plugin handle structure |
| src\_port | The source port |
| Return value | success | 0 |
| fail | non-zero |

### xf\_command

|  |  |  |
| --- | --- | --- |
| xf\_command | | |
| Synopsis | This function sends a command to ADSP Plugin. | |
| Syntax | int xf\_command(xf\_handle\_t \*handle,  u32 port,  u32 opcode,  void \*buffer,  u32 length) | |
| Parameter | handle | The pointer of target ADSP Plugin handle structure |
| port | The target port of ADSP Plugin |
| opcode | Command code  (see user manuals of ADSP Framework and ADSP Plugin) |
| buffer | The pointer to the buffer in which command is stored. |
| length | The size of the buffer in which command is stored. |
| Return value | success | 0 |
| fail | non-zero |

### xf\_response\_put

|  |  |  |
| --- | --- | --- |
| xf\_response\_put | | |
| Synopsis | This function sends a message to pipe.  The main usage is to send a message from callback function to main process. | |
| Syntax | static inline int xf\_response\_put(xf\_handle\_t \*handle,  xf\_user\_msg\_t \*msg); | |
| Parameter | handle | The pointer to ADSP Plugin handle structure which sends a command. |
| msg | The pointer to user message structure |
| Return value | success | 0 |
| fail | non-zero |

### xf\_response\_get

|  |  |  |
| --- | --- | --- |
| xf\_response\_get | | |
| Synopsis | This function gets a message from pipe.  It waits for reception of a message if no message exists.  The main usage is to wait for the finish of callback process after transmission of a command. | |
| Syntax | static inline int xf\_response\_get(xf\_handle\_t \*handle,  xf\_user\_msg\_t \*msg); | |
| Parameter | handle | The pointer to ADSP Plugin handle structure which sends a command. |
| msg | The pointer to user message structure |
| Return value | success | 0 |
| fail | non-zero |

## Callback function specification

### xf\_response\_cb

|  |  |  |
| --- | --- | --- |
| xf\_response\_cb | | |
| Synopsis | This function performs ADSP Plugin callback feature.  It is necessary for the user to make the callback function with the format of the syntax. Register the pointer to the callback function as the response parameter when executing xf\_open. | |
| Syntax | void (\*xf\_response\_cb)(xf\_handle\_t \*h,  xf\_user\_msg\_t \*msg); | |
| Parameter | h | The pointer to ADSP Plugin handle structure |
| msg | The pointer to user message structure |
| Return value | None | |

## Structures specification

### xf\_proxy\_t

|  |  |  |
| --- | --- | --- |
| xf\_proxy\_t | | |
| Synopsis | This is ADSP Interface for Linux structure.  Direct access to a member of the structure is prohibited. | |
| Access function | xf\_proxy\_pool | This function gets the pointer to AUX shared buffer pool in this structure. |

### xf\_handle\_t

|  |  |  |
| --- | --- | --- |
| xf\_handle\_t | | |
| Synopsis | This is ADSP Plugin handle structure.  Direct access to a member of the structure is prohibited. | |
| Access function | xf\_handle\_aux | This function gets the address of shared buffer in this structure. |
| xf\_handle\_auxlen | This function gets the size of shared buffer in this structure. |

### xf\_pool\_t

|  |  |  |
| --- | --- | --- |
| xf\_pool\_t | | |
| Synopsis | This is buffer pool structure.  Direct access to a member of the structure is prohibited.  Allocate this structure as many as necessary in / out buffer pools. | |
| Access function | xf\_buffer\_get | This function gets buffer from buffer pool allocated by this structure. |
| xf\_buffer\_put | This function returns buffer to buffer pool. |

### xf\_buffer\_t

|  |  |  |
| --- | --- | --- |
| xf\_buffer\_t | | |
| Synopsis | This is buffer structure.  Direct access to a member of the structure is prohibited.  Allocate this structure as many as necessary in / out buffers. | |
| Access function | xf\_buffer\_data | This function gets the address of buffer in this structure. |
| xf\_buffer\_length | This function gets the size of buffer in this structure. |

### xf\_user\_msg\_t

|  |  |  |  |
| --- | --- | --- | --- |
| xf\_user\_msg\_t | | | |
| Synopsis | This is user message structure.  Specify the pointer of this structure as a parameter of callback function. | | |
| Member | u32 | id | Component ID (only used for administration and not used by user) |
| u32 | opcode | Command code  (see user’s manuals of ADSP Framework and ADSP Plugin for details) |
| u32 | length | The length of the data stored in the buffer |
| void \* | buffer | The address of the buffer |

# Process sequence

## Initialization flow

Figure 3‑1 shows the initialization flow.

User Application

ADSP Interface for Linux

ADSP Driver for Linux

ADSP Framework

ADSP Plugin

xf\_proxy\_init

xf\_pool\_alloc

xf\_open

(1)

(2)

(3)

Figure 3‑1 Initialization flow

1. The xf\_proxy\_init performs initialization of ADSP Interface for Linux and ADSP Driver for Linux.
2. The xf\_pool\_alloc performs allocation of the shared buffer pool whose type is XF\_POOL\_AUX. Specify the pointer returned from xf\_proxy\_pool in advance.
3. xf\_open performs initialization of ADSP Plugin. A shared buffer whose type is XF\_POOL\_AUX is allocated to ADSP Plugin from shared buffer pool.

## Flow of sending a command

Figure 3‑2 shows the reference flow to send / receive a command.

User Application

ADSP Interface for Linux

ADSP Driver for Linux

ADSP Framework

ADSP Plugin

xf\_handle\_aux

xf\_command

xf\_handle\_auxlen

(1)

(2)

(3)

(4)

(6)

xf\_response\_cb

pipe

xf\_response\_get

xf\_response\_put

(7)

(8)

(9)

(5)

(10)

Figure 3‑2 The standard flow to send / receive a command

1. xf\_handle\_aux gets the address of buffer in which command is stored.
2. xf\_handle\_auxlen gets the size of the buffer.
3. Store a command to the buffer. Take care that the command does not exceed the size of the buffer.
4. xf\_command sends the command. The function ends after the finish of sending.
5. Some commands are processed by ADSP Framework, others by ADSP Plugin.
6. xf\_response\_get waits for a message from pipe.
7. The callback function registered by xf\_open is called after finish of the process of command.
8. xf\_response\_put sends the received message to pipe.
9. When xf\_response\_put is executed, xf\_response\_get stops. If the order of (6) and (8) is inverted, xf\_response\_get stops immediately.
10. Continue process according to received message.

## Flow to allocate input / output buffer

Figure 3‑1 shows the reference flow to allocate input / output buffer.

User Application

ADSP Interface for Linux

ADSP Driver for Linux

ADSP Framework

ADSP Plugin

xf\_pool\_alloc

(1)

(2)

xf\_buffer\_get

Figure 3‑3 The standard flow to allocate input / output buffer

1. xf\_pool\_alloc allocates buffer pool whose type is XF\_POOL\_INPUT or XF\_POOL\_OUTPUT.
2. xf\_buffer\_get gets buffer from the buffer pool.

## Flow of send / receive input buffer

Figure 3‑2 shows the reference flow to send / receive input buffer.

User Application

ADSP Interface for Linux

ADSP Driver for Linux

ADSP Framework

ADSP Plugin

xf\_buffer\_data

xf\_command

xf\_buffer\_length

(1)

(2)

(3)

(4)

xf\_response\_cb

(5)⑤

(6)⑥

xf\_command

Figure 3‑4 The standard flow to send / receive data

1. xf\_buffer\_data gets the address of buffer to send data.
2. xf\_buffer\_length gets the size of the buffer.
3. Store data to the buffer. Take care that the data does not exceed the size of the buffer.
4. xf\_command sends the data in the buffer. The function ends after the finish of sending. Specify XF\_EMPTY\_THIS\_BUFFER for the command code. See ADSP Framework user’s manual for details.
5. The callback function registered by xf\_open is called after finish of the process.
6. If necessary, continue to store next data to the buffer and send again. It is also possible to use another thread, not by callback function.

## Flow to send / receive output buffer

Figure 3‑2 shows the reference flow to send / receive output buffer.

User Application

ADSP Interface for Linux

ADSP Driver for Linux

ADSP Framework

ADSP Plugin

xf\_buffer\_data

xf\_command

xf\_buffer\_length

(1)

(2)

(3)

xf\_response\_cb

(4)

(5)

xf\_command

Figure 3‑5 The flow to send / receive output buffer

1. xf\_buffer\_data gets the address of buffer to send data.
2. xf\_buffer\_length gets the size of the buffer.
3. xf\_command sends the data in the buffer. The function ends after the finish of sending. Specify XF\_FILL\_THIS\_BUFFER for the command code. See ADSP Framework user’s manual for details.
4. The callback function registered by xf\_open is called after finish of the process.
5. Get the output data from the buffer after the process is finished. If necessary, continue to store next data to the buffer and send again. It is also possible to use another thread, not by callback function.

## Flow to release input / output buffer

Figure 3‑1 shows the reference flow to release input / output buffer.

User Application

ADSP Interface for Linux

ADSP Driver for Linux

ADSP Framework

ADSP Plugin

xf\_pool\_free

(2)

(1)

xf\_buffer\_put

Figure 3‑6 The standard flow to release input / output buffer

1. xf\_buffer\_put returns all grabbed buffers to the buffer pool.
2. xf\_pool\_free releases the buffer pool.

## Termination flow

Figure 3‑1 shows termination flow.

User Application

ADSP Interface for Linux

ADSP Driver for Linux

ADSP Framework

ADSP Plugin

xf\_pool\_free

xf\_close

(2)

(1)

xf\_proxy\_close

(3)

Figure 3‑7 Termination flow

1. xf\_close performs ADSP Plugin termination.  
     
   (The following process should be performed only when ADSP Interface for Linux is also terminated.)
2. xf\_pool\_free releases the shared buffer pool whose type is XF\_POOL\_AUX by using the return value of xf\_proxy\_pool.
3. xf\_proxy\_close terminates ADSP Interface for Linux and ADSP Driver for Linux.

# Notes

This section describes the notice of developing user programs.

## Function Call

User programs which calls the functions in this specification should obey the calling rules of compiler.

## Other notes

### Allocation of memory

Before calling the functions in this specification, allocate necessary memory area and each structure used for the parameters of each function.

### Out of range memory access

The functions in this specification never access out of allocated memory or related I/O.

### Combination with other applications

Take care not to duplicate symbol names when other applications are combined with other programs.

### Monitoring on Performance

The products embedding this Software shall observe performance of the Software periodically with Watch Dog timer or such functions in order not to damage system performance.

# Appendix

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

Below table show target platforms support for each use case.

Table 5‑1 Target environment for each use case.

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

## Standard Use Case: Playback

Figure 5-1, shows an example of the audio data path are transferred which use renderer function.

External Memory

Speaker

SRC

DVC

SCU

SSI

CTU

ADMAC/

ADMACpp

ADMAC/

ADMACpp

PCM data

MIX

Figure 5‑1 Audio data path for renderer function

Below table shows supported features and its usage for OMX renderer.

Table 5‑2 Supported features for OMX Renderer Interface

|  |  |
| --- | --- |
| Name | Usage |
| Audio-DMAC/ Audio-DMACpp | Transfer data between two audio modules; memory and audio module.  **[Note]**:  About Rcar E3:   * ADMAC supports from channels 0 to 15. * ADMAC cannot be used to transfer data between SCU modules and SSI0. |
| SSI | Send audio data to speaker. |
| SRC | Convert sampling rate between 32 kHz, 44.1 kHz, and 48 kHz.  **[Note]**:  In Rcar E3 only two SRC modules supporting multi-channels are SRC1, SRC3.  In Rcar H3/M3/M3N four SRC modules supporting multi-channels are SRC0, SRC1, SRC3, and SRC4. |
| DVC | Volume control which can modify volume value. |
| CTU | Channel transfer (from 1, 2, 4, 6, 8 channel to 1 or 2 channel). |
| MIX | It is used for mixing (adding) streams from two to four audio stream sources into a single stream. It also support the volume control (gain level) for each input stream. |

Below table show parameter of data is supported.

Table 5‑3 Supported data for playback

| Item | Description |
| --- | --- |
| Input data format | |  |  |  |  | | --- | --- | --- | --- | | Channel number | PCM bit-width (fix-point) | | Sample rate  (Hz) | | 16-bit | 24-bit | 48000/44100/32000 | | 1ch | O | X | O | | 2ch | O | O | O | | 4ch | O | O | O | | 6ch | O | O | O | | 8ch | O | O | O | |
| Output data format | |  |  |  |  | | --- | --- | --- | --- | | Channel number | PCM bit-width (fix-point) | | Sample rate  (Hz) | | 16-bit | 24-bit | 48000/44100/32000 | | 1ch | O | X | O | | 2ch | O | O | O | |
| Frame size | 1024 bytes |

Implementation:

O: Supported.

X: Not supported.

User can use ADSP Smoketest application to play the stream.

ADSP Smoketest program usage guideline:

#./adsp-omx-launch -card 0,4 -i <name> –w <value> -o <name> -rdr <name>]

Explain:

-i <name>: Input file (.pcm or .wav).

-card 0,4: Select rcar-sound card is to open codec.

–w <value>: PCM Bit per sample (16/24).

–o <name>: Output device (renderer).

–rdr <name>: Renderer configuration file.

Renderer configuration file example, parameters will be set in a row follow the order:

# In\_fs Out\_fs Dmach1 Output1 Dmach2 Output2 Volume Framesize In\_ch Out\_ch Mix\_ctrl

Parameters meaning:

In\_fs: Renderer input sampling frequency (32000/44100/48000).

Out\_fs: Renderer output sampling frequency (32000/44100/48000).

Dmach1: Renderer DMA channel 1 (ADMAC\_CH00 to

ADMAC\_CH31/ADMACPP\_CH00 to ADMACPP\_CH28).

Output1: Renderer output source 1 (SSI00/SRC0 to SRC9).

Dmach2: Renderer DMA channel2 (ADMAC\_CH00 to

ADMAC\_CH31/ADMACPP\_CH00 to ADMACPP\_CH28).

Output2: Renderer output source 2 (SSI00/NONCONFIG).

Volume: Renderer volume gain (gain from 0 to 8. If DVC module is not

used, this value must be -1)

Framesize: Renderer frame size (1024).

In\_ch: Number of Renderer input channels (1/2/4/6/8).

Out\_ch: Number of Renderer output channels (1/2).

Mix\_ctrl: Renderer Mix enable control (0/1) (0 is disable MIX function,

1 is enable MIX function).

### Playback flow 1: Normal

In this flow, data without data conversion is transfer to speaker.

External Memory

SSI

ADMAC/

ADMACpp

Figure 5‑2 Data path for payback normal

* Example use smoke test:

#./adsp-omx-launch –card 0,4 -i thetest\_FULL\_s\_32000\_16.pcm -w 16 -o renderer

-rdr rdrconfig\_ADMACPP.txt

Content of rdrconfig\_ADMACPP.txt:

# In\_fs Out\_fs Dmach1 Output1 Dmach2 Output2 Volume Framesize In\_ch Out\_ch Mix\_ctrl

32000 32000 ADMACPP\_CH15 SSI00 ADMACPP\_CH00 NONCONFIG -1 1024 2 2 0

### Playback flow 2: Sampling Rate Conversion

In this flow, data is converted from input sample rate to output sample rate, then transferred to speaker.

External Memory

ADMAC/

ADMACpp

ADMAC/

ADMACpp

SSI

SRC

Figure 5‑3 Data path for playback with sampling rate conversion in R-Car H3/M3/M3N

External Memory

ADMAC/

ADMACpp

ADMACpp

SSI

SRC

Figure 5‑4 Data path for playback with sampling rate conversion in R-Car E3

* Range value:

In\_fs: 32000/44100/48000

Out\_fs: 32000/44100/48000

Output1: SRC0-SRC9

**[Note]:**

In R-Car E3 two SRC modules supporting multi-channels are SRC1, SRC3.

In R-Car H3/M3/M3N four SRC modules supporting multi-channels are SRC0, SRC1, SRC3, SRC4.

* Example use smoke test:

#./adsp-omx-launch –card 0,4 -i thetest\_FULL\_s\_32000\_16.pcm -w 16 -o renderer

-rdr rdrconfig\_ADMACPP.txt

Content of rdrconfig\_ADMACPP.txt:

# In\_fs Out\_fs Dmach1 Output1 Dmach2 Output2 Volume Framesize In\_ch Out\_ch Mix\_ctrl

32000 48000 ADMACPP\_CH15 SRC0 ADMACPP\_CH00 SSI00 -1 1024 2 2 0

### Playback flow 3: Channel Transfer

In this flow, data is converted from input channel to output channel, then transferred to speaker.

External Memory

ADMAC/

ADMACpp

ADMAC/

ADMACpp

CTU

SSI

SRC

Figure 5‑5 Data path for playback with channel transfer in R-Car H3/M3/M3N

External Memory

ADMACpp

ADMAC/

ADMACpp

CTU

SSI

SRC

Figure 5‑6 Data path for playback with channel transfer in R-Car E3

* Range value:

In\_ch: 1, 2, 4, 6, 8

Out\_ch: 1, 2

Ouput1: SRC0-SRC9

**[Note]:**

In R-Car E3 two SRC modules supporting multi-channels are SRC1, SRC3.

In R-Car H3/M3/M3N four SRC modules supporting multi-channels are SRC0, SRC1, SRC3, SRC4.

* Example use smoke test:

#./adsp-omx-launch –card 0,4 -i thetest\_FULL\_8ch\_32000\_16.pcm -w 16 -o renderer

-rdr rdrconfig\_ADMACPP.txt

Content of rdrconfig\_ADMACPP.txt:

# In\_fs Out\_fs Dmach1 Output1 Dmach2 Output2 Volume Framesize In\_ch Out\_ch Mix\_ctrl

32000 32000 ADMACPP\_CH15 SRC1 ADMACPP\_CH00 SSI00 -1 1024 8 2 0

### Playback flow 4: Volume Control

In this flow, data is changed volume, then transferred to speaker.

External Memory

ADMAC/

ADMACpp

ADMAC/

ADMACpp

DVC

SSI

SRC

Figure 5‑7 Data path for playback with volume control in R-Car H3/M3/M3N

External Memory

ADMAC/

ADMACpp

ADMACpp

DVC

SSI

SRC

Figure 5‑8 Data path for playback with volume control in R-Car E3

* Range value:

Volume: 0 – 8 (step 0.5).

**[Note]:** If DVC module is not used, this value must be -1.

* Example use smoke test:

#./adsp-omx-launch -card 0,4 -i thetest\_FULL\_s\_32000\_16.pcm -w 16 -o renderer

-rdr rdrconfig\_ADMACPP.txt

Content of rdrconfig\_ADMACPP.txt:

# In\_fs Out\_fs Dmach1 Output1 Dmach2 Output2 Volume Framesize In\_ch Out\_ch Mix\_ctrl

32000 32000 ADMACPP\_CH15 SRC0 ADMACPP\_CH00 SSI00 1.5 1024 2 2 0

Beside that, stream can update volume during playback (DVC module must be enable – value of Volume is not -1) by pressing “u” to set volume up and “d” to set volume down.

### Playback flow 5: MIX function

When mixing is used, the second stream and later must be consistent output sample rate, output channel and PCM width with the first one. We can convert sample rate, number of channel by SRC, CTU module.

Below table shows the maximum stream can play in MIX mode for R-Car E3/M3/M3N/H3

Table 5‑4 Number of stream can play in MIX function

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Number of stream | E3 | M3 | M3N | H3 |
| 2 streams | O | O | O | O |
| 3 streams | O | O | O | O |
| 4 streams | O | O | O | O |

The order to playback stream when use Smoketest application is from the longest to the shortest duration stream.

* Range value:

Dmach1, Dmach2: ADMAC\_CH00 to ADMAC\_CH31/ADMACPP\_CH00 to ADMACPP\_CH28

Mix\_ctrl: 0 or 1(0 is disable MIX function,

1 is enable MIX function)

**[Note]:**

If Dmach1 is ADMACPP:

* Input sample rate (In\_fs) of multi-channel stream must be same.
* 16-bit stream: the total number of input channel (In\_ch) is maximum 10:
* 24-bit stream: the total number of input channel (In\_ch) is maximum 6.

If Dmach1 is ADMAC:

* 16-bit stream: the total number of input channel (In\_ch) is maximum 16:
* 24-bit stream: the total number of input channel (In\_ch) is maximum 8:

[Note]

For the case of 24-bit streams when Equalizers are involved, the maximum of streams to be mixed is 2 due to memory limitation.

#### MIX function flow 1: MIX 2 streams into 1 stream

In this flow, 2 stream is mixed into 1 stream. If the second stream is not consistent with the first one (sample rate or channel), stream must setup sample rate or channel to be the same as the first, then transferred to speaker.

CTU

SRC

ADMAC/

ADMACpp

External Memory

SSI

MIX

ADMAC/

ADMACpp

External Memory

CTU

SRC

ADMAC/

ADMACpp

Figure 5‑9 Data path for playback MIX 2 streams into 1 stream in R-Car H3/M3/M3N

CTU

SRC

ADMAC/

ADMACpp

External Memory

SSI

MIX

ADMACpp

External Memory

CTU

SRC

ADMAC/

ADMACpp

Figure 5‑10 Data path for playback MIX 2 streams into 1 stream in R-Car E3

* Example use smoke test:

#./adsp-omx-launch -card 0,4 –i thetest\_FULL\_4ch\_32000\_16.pcm -w 16 -o renderer

-rdr rdrconfig\_ADMAC\_1.txt

#./adsp-omx-launch -card 0,4 –i thetest\_FULL\_6ch\_32000\_16.pcm -w 16 -o renderer

-rdr rdrconfig\_ADMAC\_2.txt

Content of rdrconfig\_ADMAC\_1.txt:

# In\_fs Out\_fs Dmach1 Output1 Dmach2 Output2 Volume Framesize In\_ch Out\_ch Mix\_ctrl

32000 32000 ADMAC\_CH05 SRC3 ADMACPP\_CH00 SSI00 -1 1024 4 2 1

Content of rdrconfig\_ADMAC\_2.txt:

# In\_fs Out\_fs Dmach1 Output1 Dmach2 Output2 Volume Framesize In\_ch Out\_ch Mix\_ctrl

32000 32000 ADMAC\_CH13 SRC1 ADMACPP\_CH10 SSI00 -1 1024 6 2 1

#### MIX function flow 2: MIX 3 streams into 1 stream

In this flow, 3 stream is mixed into 1 stream. If the later stream is not consistent with the first one (sample rate or channel) stream must setup sample rate or channel to be the same as the first, then transferred to speaker.

CTU

SRC

ADMAC/

ADMACpp

External Memory

SSI

MIX

External Memory

CTU

SRC

ADMAC/

ADMACpp

ADMAC/

ADMACpp

ADMAC/

ADMACpp

SRC

External Memory

CTU

Figure 5‑11 Data path for playback MIX 3 streams into 1 stream in R-Car H3/M3/M3N

CTU

SRC

ADMAC/

ADMACpp

External Memory

SSI

MIX

External Memory

CTU

SRC

ADMACpp

ADMAC/

ADMACpp

ADMAC/

ADMACpp

SRC

External Memory

CTU

Figure 5‑12 Data path for playback MIX 3 streams into 1 stream in R-Car E3

* Example use smoke test:

#./adsp-omx-launch -card 0,4 –i thetest\_FULL\_4ch\_32000\_16.pcm -w 16 -o renderer -rdr rdrconfig\_ADMAC\_1.txt

#./adsp-omx-launch -card 0,4 –i thetest\_FULL\_4ch\_32000\_16.pcm -w 16 -o renderer -rdr rdrconfig\_ADMAC\_2.txt

#./adsp-omx-launch -card 0,4 –i thetest\_FULL\_s\_32000\_16.pcm -w 16 -o renderer -rdr rdrconfig\_ADMAC\_3.txt

Content of rdrconfig\_ADMAC\_1.txt:

# In\_fs Out\_fs Dmach1 Output1 Dmach2 Output2 Volume Framesize In\_ch Out\_ch Mix\_ctrl

32000 32000 ADMAC\_CH05 SRC3 ADMACPP\_CH00 SSI00 -1 1024 4 2 1

Content of rdrconfig\_ADMAC\_2.txt:

# In\_fs Out\_fs Dmach1 Output1 Dmach2 Output2 Volume Framesize In\_ch Out\_ch Mix\_ctrl

32000 32000 ADMAC\_CH13 SRC1 ADMACPP\_CH10 SSI00 0.5 1024 4 2 1

Content of rdrconfig\_ADMAC\_3.txt:

# In\_fs Out\_fs Dmach1 Output1 Dmach2 Output2 Volume Framesize In\_ch Out\_ch Mix\_ctrl

32000 32000 ADMAC\_CH03 SRC0 ADMACPP\_CH04 SSI00 2 1024 2 2 1

#### MIX function flow 3: MIX 4 streams into 1 stream

In this flow, 4 stream is mixed into 1 stream, if the later stream is not consistent with the first one (sample rate or channel) stream must setup sample rate or channel to be the same as the first, then transferred to speaker.

CTU

SRC

ADMAC/

ADMACpp

External Memory

External Memory

CTU

SRC

ADMAC/

ADMACpp

MIX

SSI

ADMAC/

ADMACpp

ADMAC/

ADMACpp

SRC

External Memory

CTU

ADMAC/

ADMACpp

CTU

SRC

External Memory

Figure 5‑13 Data path for playback MIX 4 streams into 1 stream in R-Car H3/M3/M3N

CTU

SRC

ADMAC/

ADMACpp

External Memory

External Memory

CTU

SRC

ADMAC/

ADMACpp

MIX

SSI

ADMACpp

ADMAC/

ADMACpp

SRC

External Memory

CTU

ADMAC/

ADMACpp

CTU

SRC

External Memory

Figure 5‑14 Data path for playback MIX 4 streams into 1 stream in R-Car E3

* Example use smoke test:

#./adsp-omx-launch -card 0,4 –i thetest\_FULL\_s\_32000\_16.pcm -w 16 -o renderer

-rdr rdrconfig\_ADMAC\_1.txt

#./adsp-omx-launch -card 0,4 –i thetest\_FULL\_s\_32000\_16.pcm -w 16 -o renderer

-rdr rdrconfig\_ADMAC\_2.txt

#./adsp-omx-launch -card 0,4 –i thetest\_FULL\_4ch\_32000\_16.pcm -w 16 -o renderer

-rdr rdrconfig\_ADMAC\_3.txt

#./adsp-omx-launch -card 0,4 –i thetest\_FULL\_6ch\_32000\_16.pcm -w 16 -o renderer

-rdr rdrconfig\_ADMAC\_4.txt

Content of rdrconfig\_ADMAC\_1.txt:

# In\_fs Out\_fs Dmach1 Output1 Dmach2 Output2 Volume Framesize In\_ch Out\_ch Mix\_ctrl

32000 32000 ADMAC\_CH05 SRC0 ADMACPP\_CH00 SSI00 -1 1024 2 2 1

Content of rdrconfig\_ADMAC\_2.txt:

# In\_fs Out\_fs Dmach1 Output1 Dmach2 Output2 Volume Framesize In\_ch Out\_ch Mix\_ctrl

32000 32000 ADMAC\_CH13 SRC5 ADMACPP\_CH10 SSI00 -1 1024 2 2 1

Content of rdrconfig\_ADMAC\_3.txt:

# In\_fs Out\_fs Dmach1 Output1 Dmach2 Output2 Volume Framesize In\_ch Out\_ch Mix\_ctrl

32000 32000 ADMAC\_CH03 SRC1 ADMACPP\_CH04 SSI00 -1 1024 4 2 1

Content of rdrconfig\_ADMAC\_4.txt:

# In\_fs Out\_fs Dmach1 Output1 Dmach2 Output2 Volume Framesize In\_ch Out\_ch Mix\_ctrl

32000 32000 ADMAC\_CH06 SRC3 ADMACPP\_CH6 SSI00 -1 1024 6 2 1

## Standard Use Case: Record

Figure 5-15, shows an example of the audio data path are transferred which use capture function.

External Memory

Microphone

SRC

DVC

SCU

SSI

ADMAC/

ADMACpp

ADMAC/

ADMACpp

PCM data



Figure 5‑15 Audio data path for capture function

Below table shows supported features and its usage for OMX capture.

Table 5‑5 Supported features for OMX Capture Interface

|  |  |
| --- | --- |
| Name | Usage |
| Audio-DMAC/ Audio-DMACpp | Transfer data between two audio modules; memory and audio module.  **[note]**:  About R-Car E3:   * ADMAC supports from channels 0 to 15. * ADMAC cannot be used to transfer data between SSI10 modules and SCU. |
| SSI | Receive audio data from microphone. |
| SRC | Convert sampling rate between 32 kHz, 44.1 kHz, and 48 kHz. |
| DVC | Volume control which can modify volume value. |

Below table Show parameter of data is supported.

Table 5‑6 Supported data for record

| Item | Description |
| --- | --- |
| Input data format | |  |  |  |  | | --- | --- | --- | --- | | Channel number | PCM bit-width (fix-point) | | Sample rate  (Hz) | | 16-bit | 24-bit | 48000/44100/32000 | | 1ch | O | X | O | | 2ch | O | O | O | |
| Output data format | |  |  |  |  | | --- | --- | --- | --- | | Channel number | PCM bit-width (fix-point) | | Sample rate  (Hz) | | 16-bit | 24-bit | 48000/44100/32000 | | 1ch | O | X | O | | 2ch | O | O | O | |
| Frame size | 1024 bytes |

Implementation:

O: Supported.

X: Not supported.

User can use ADSP Smoketest application to record the stream.

ADSP Smoketest program usage guideline:

#./adsp-omx-launch -card 0,4 -i <name> –c <value> –w <value> -l <value>

-o <name> -cap <name>]

Explain:

-i <name>: Input device (capture)

-card 0,4: Select rcar-sound card is to open codec

-c <value>: PCM channel number for Capture (1/2)

–w <value>: PCM Bit per sample (16/24)

–l <value>: Recording time (second)

–o <name>: Output file (.pcm)

–cap <name>: Capture configuration file

Capture configuration file example, parameters will be set in a row follow the order:

# In\_fs Out\_fs Dmach1 Input1 Dmach2 Input2 Volume Framesize

Parameters meaning:

In\_fs: Capture input sampling frequency (32000/44100/48000)

Out\_fs: Capture output sampling frequency (32000/44100/48000)

Dmach1: Capture DMA channel 1 (ADMAC\_CH00 to

ADMAC\_CH31/ADMACPP\_CH00 to ADMACPP\_CH28)

Input1: Capture input source 1 (SSI10/SRC0 to SRC9)

Dmach2: Capture DMA channel2 (ADMAC\_CH00 to

ADMAC\_CH31/ADMACPP\_CH00 to ADMACPP\_CH28)

Input2: Capture input source 2 (SSI10/NONCONFIG)

Volume: Capture volume gain (gain from 0 to 8. If DVC module is not

used, this value must be -1)

Framesize: Capture frame size (1024)

### Capture flow 1: Normal

In this flow, data without data conversion is transferred from microphone to external memory



External Memory

ADMAC/

ADMACpp

SSI

Figure 5‑16 Data path for case record normal

* Example use smoke test:

#./adsp-omx-launch -card 0,4 -i capture -c 2 -w 16 –l 15 -o out.pcm -cap capconfig\_ADMACPP.txt

Content of capconfig\_ADMACPP.txt:

#In\_fs Out\_fs Dmach1 Input1 Dmach2 Input2 Volume Framesize

48000 48000 ADMACPP\_CH15 SSI10 ADMACPP\_CH00 NONCONFIG -1 1024

### Capture flow 2: Sampling Rate Conversion

In this flow, data, which is converted from input sample rate to output sample rate, is transferred from microphone to external memory



External Memory

ADMAC/

ADMACpp

ADMAC/

ADMACpp

SSI

SRC

Figure 5‑17 Data path for record sampling rate conversion in R-Car H3/M3/M3N



External Memory

ADMACpp

ADMAC/

ADMACpp

SSI

SRC

Figure 5‑18 Data path for record sampling rate conversion in R-Car E3

* Range value.

In\_fs: 32000/44100/48000

Out\_fs: 32000/44100/48000

Input1: SRC0-SRC9

* Example use smoke test:

#./adsp-omx-launch -card 0,4 -i capture -c 2 -w 16 –l 15 -o out.pcm -cap capconfig\_ADMACPP.txt

Content of capconfig\_ADMACPP.txt:

#In\_fs Out\_fs Dmach1 Input1 Dmach2 Input2 Volume Framesize

32000 48000 ADMACPP\_CH15 SRC0 ADMACPP\_CH00 SSI10 -1 1024

### Capture flow 3: Volume Control

In this flow, data, which is update volume, is transferred from microphone to external memory



External Memory

ADMAC/

ADMACpp

ADMAC/

ADMACpp

DVC

SSI

SRC

Figure 5‑19 Data path for record volume control in R-Car H3/M3/M3N



External Memory

ADMACpp

ADMAC/

ADMACpp

DVC

SSI

SRC

Figure 5‑20 Data path for record volume control in R-Car E3

* Range value.

Volume: 0 – 8 (step 0.5).

**[Note]:** If DVC module is not used, this value must be -1.

* Example use smoke test:

#./adsp-omx-launch -card 0,4 -i capture -c 2 -w 16 –l 15 -o out.pcm -cap capconfig\_ADMACPP.txt

Content of capconfig\_ADMACPP.txt:

#In\_fs Out\_fs Dmach1 Input1 Dmach2 Input2 Volume Framesize

32000 48000 ADMACPP\_CH15 SRC0 ADMACPP\_CH00 SSI10 2 1024

Record processing can stop by pressing “s”.

Besides that, stream can update volume during record (DVC module must be enable – value

of Volume is not -1) by pressing “u” to set volume up and “d” to set volume down.

## Standard Use Case: Equalizer

Equalizer plugin does not support setting in runtime.

User can use ADSP Smoketest application to equalize stream:

ADSP Smoketest program usage guideline:

#./adsp-omx-launch -i <name> –w <value> –c <value> -o <name>

-eq <value> -eqzfs <value> -eqz <name>]

Explain:

-i <name>: Input file (.pcm or .wav).

–o <name>: Output file (.pcm).

–w <value>: PCM Bit per sample (16/24).

–c <value>: PCM channel number for Equalizer (1/2).

-eq <value>: Enable/Disable equalizer (on/off) (default: off).

-eqzfs <value>: Equalizer PCM sampling frequency (32000/44100/48000).

-eqz <name>: Equalizer configuration file.

Equalizer configuration file example:

* Parametric configuration file
* Start 1st line with "Parametric" identity.
* 2nd line to 10th line will contain information structure like below:

Type Fc Bandwidth Gain BaseGain

* Type: Filter type (P: Peak, R: Treble, B: Bass, T: Through)
* Fc: Frequency center (Peak|Through: 20-20000,

Treble: 5000-11000, Bass: 50-500)

* Bandwidth: 0.5 - 15
* Gain: -15.0 - 15.0 (dB)
* BaseGain: -10.0 - 10.0 (dB)
* Graphic configuration file
* Start 1st line with "Graphic" identity.
* 2nd line to 6th line with Gain (-10.0 - 10.0) dB value.

Equalizer plugin

External Memory

External Memory

Figure 5‑21 Data path for equalizer

* Example use smoke test:

#./adsp-omx-launch -i thetest\_FULL\_s\_32000\_16.pcm -w 16 –c 2

-o out.pcm –eq on –eqzfs 32000 -eqz parametric\_config.txt

Content of parametric\_config.txt:

Parametric

T 15000 0.707 1.0 1.0

T 15000 0.707 1.0 1.0

T 15000 0.707 1.0 1.0

T 15000 0.707 1.0 1.0

T 15000 0.707 1.0 1.0

T 15000 0.707 1.0 1.0

T 15000 0.707 1.0 1.0

T 15000 0.707 1.0 1.0

T 15000 0.707 1.0 1.0

## Standard Use Case: Route

User can use ADSP Smoketest application to route between Capture, Renderer and Equalizer

### Route flow 1: Capture – Equalizer

In this flow, data is transferred from microphone to Capture and Equalizer plugin and stored in external memory.



External Memory

Capture plugin

Equalizer plugin

Figure 5‑22 Data path for route Capture – Equalizer

* Example use smoke test:

#./adsp-omx-launch -o out.pcm –card 0,4 –i capture -w 24 -c 2 -eqzfs 48000 -eqz parametric\_config.txt -eq on -cap capconfig\_ADMACPP.txt

Content of capconfig\_ADMACPP.txt:

#In\_fs Out\_fs Dmach1 Input1 Dmach2 Input2 Volume Framesize

32000 48000 ADMACPP\_CH15 SRC6 ADMACPP\_CH04 SSI10 0.5 1024

Content of parametric\_config.txt:

Parametric

T 15000 0.707 1.0 1.0

T 15000 0.707 1.0 1.0

T 15000 0.707 1.0 1.0

T 15000 0.707 1.0 1.0

T 15000 0.707 1.0 1.0

T 15000 0.707 1.0 1.0

T 15000 0.707 1.0 1.0

T 15000 0.707 1.0 1.0

T 15000 0.707 1.0 1.0

### Route flow 2: Equalizer – Renderer

In this flow, data is transferred from external memory to Equalizer and Renderer plugin and to speaker.

External Memory

Renderer plugin

Equalizer plugin

Figure 5‑23 Data path for route Equalizer – Renderer

* Example use smoke test:

#./adsp-omx-launch –card 0,4 -o renderer -i thetest\_FULL\_s\_48000\_24.pcm -w 24 -c 2 -eqzfs 48000 -eqz graphic\_config.txt -eq on -rdr rdrconfig\_ADMACPP.txt

Content of rdrconfig\_ADMACPP.txt:

# In\_fs Out\_fs Dmach1 Output1 Dmach2 Output2 Volume Framesize In\_ch Out\_ch Mix\_ctrl

48000 32000 ADMACPP\_CH25 SRC3 ADMACPP\_CH00 SSI00 0.5 1024 2 2 0

Content of graphic\_config.txt:

Graphic

-10.0

-10.0

-10.0

-10.0

-10.0

### Route flow 3: Capture – Equalizer – Renderer

In this flow, data is transferred from microphone to Capture and Equalizer plugin and then to Renderer plugin and to speaker.



Renderer plugin

Equalizer plugin

Capture plugin

Figure 5‑24 Data path for route Capture – Equalizer – Renderer

[Note] The input sample rate of Capture and output sample rate of Renderer should be specified to the same value.

* Example use smoke test:

#./adsp-omx-launch –card 0,4 -o renderer -i capture -w 24 -c 2 -eqzfs 48000 -eqz parametric\_config.txt -eq on -rdr rdrconfig\_ADMACPP.txt -cap capconfig\_ADMACPP.txt

Content of capconfig\_ADMACPP.txt:

#In\_fs Out\_fs Dmach1 Input1 Dmach2 Input2 Volume Framesize

32000 48000 ADMACPP\_CH15 SRC6 ADMACPP\_CH04 SSI10 0.5 1024

Content of rdrconfig\_ADMACPP.txt:

# In\_fs Out\_fs Dmach1 Output1 Dmach2 Output2 Volume Framesize In\_ch Out\_ch Mix\_ctrl

48000 32000 ADMACPP\_CH25 SRC3 ADMACPP\_CH00 SSI00 0.5 1024 2 2 0

Content of parametric\_config.txt:

Parametric

T 15000 0.707 1.0 1.0

T 15000 0.707 1.0 1.0

T 15000 0.707 1.0 1.0

T 15000 0.707 1.0 1.0

T 15000 0.707 1.0 1.0

T 15000 0.707 1.0 1.0

### Route flow 4: Capture – Renderer

In this flow, data is transferred from microphone to Capture, Renderer plugin and to speaker.



Capture plugin

Renderer plugin

Figure 5‑25 Data path for route Equalizer – Renderer

[Note] The input sample rate of Capture and output sample rate of Renderer should be specified to the same value.

* Example use smoke test:

#./adsp-omx-launch –card 0,4 -o renderer -i capture -w 24 -c 2 –rdr rdrconfig\_ADMACPP.txt -cap capconfig\_ADMACPP.txt

Content of capconfig\_ADMACPP.txt:

#In\_fs Out\_fs Dmach1 Input1 Dmach2 Input2 Volume Framesize

32000 48000 ADMACPP\_CH15 SRC6 ADMACPP\_CH04 SSI10 0.5 1024

Content of rdrconfig\_ADMACPP.txt:

# In\_fs Out\_fs Dmach1 Output1 Dmach2 Output2 Volume Framesize In\_ch Out\_ch Mix\_ctrl

48000 32000 ADMACPP\_CH25 SRC3 ADMACPP\_CH00 SSI00 0.5 1024 2 2 0

Content of parametric\_config.txt:

Parametric

T 15000 0.707 1.0 1.0

T 15000 0.707 1.0 1.0

T 15000 0.707 1.0 1.0

T 15000 0.707 1.0 1.0

T 15000 0.707 1.0 1.0

T 15000 0.707 1.0 1.0

### Route flow 5: Equalizer – Renderer use MIX function

In this flow, streams are transferred from external memory to Equalizer and Renderer plugin and to speaker. Below is a simple image when using mix function with Equalizer routing. For detail configuration, please refer to 5.1.5.3

Renderer plugin

Equalizer plugin

External Memory

Renderer plugin

Equalizer plugin

External Memory

SSI

MIX

External Memory

Equalizer plugin

Renderer plugin

External Memory

Equalizer plugin

Renderer plugin

Figure 5‑26 Data path for route Equalizer – Renderer use MIX function

* Range:

If using 24 bit/stereo, maximum 3 mixing streams are supported.

* Example use smoke test:

#./adsp-omx-launch –card 0,4 -o renderer -i thetest\_FULL\_s\_32000\_16.pcm -w 16 -c 2 -eqzfs 32000 -eqz graphic\_config.txt -eq on -rdr rdrconfig\_ADMACPP1.txt

#./adsp-omx-launch –card 0,4 -o renderer -i thetest\_FULL\_s\_48000\_16.pcm -w 16 -c 2 -eqzfs 48000 -eqz graphic\_config.txt -eq on -rdr rdrconfig\_ADMACPP2.txt

#./adsp-omx-launch –card 0,4 -o renderer -i thetest\_FULL\_s\_44100\_16.pcm -w 16 -c 2 -eqzfs 44100 -eqz graphic\_config.txt -eq on -rdr rdrconfig\_ADMACPP3.txt

#./adsp-omx-launch –card 0,4 -o renderer -i thetest\_FULL\_s\_48000\_16.pcm -w 16 -c 2 -eqzfs 48000 -eqz graphic\_config.txt -eq on -rdr rdrconfig\_ADMACPP4.txt

Content of rdrconfig\_ADMACPP1.txt:

# In\_fs Out\_fs Dmach1 Output1 Dmach2 Output2 Volume Framesize In\_ch Out\_ch Mix\_ctrl

32000 32000 ADMACPP\_CH00 SRC0 ADMACPP\_CH02 SSI00 0.5 1024 2 2 1

Content of rdrconfig\_ADMACPP2.txt:

# In\_fs Out\_fs Dmach1 Output1 Dmach2 Output2 Volume Framesize In\_ch Out\_ch Mix\_ctrl

48000 32000 ADMACPP\_CH05 SRC1 ADMACPP\_CH04 SSI00 1 1024 2 2 1

Content of rdrconfig\_ADMACPP3.txt:

# In\_fs Out\_fs Dmach1 Output1 Dmach2 Output2 Volume Framesize In\_ch Out\_ch Mix\_ctrl

44100 32000 ADMACPP\_CH10 SRC3 ADMACPP\_CH06 SSI00 -1 1024 2 2 1

Content of rdrconfig\_ADMACPP4.txt:

# In\_fs Out\_fs Dmach1 Output1 Dmach2 Output2 Volume Framesize In\_ch Out\_ch Mix\_ctrl

48000 32000 ADMACPP\_CH13 SRC5 ADMACPP\_CH08 SSI00 2 1024 2 2 1

Content of graphic\_config.txt:

Graphic

-10.0

-10.0

-10.0

-10.0

-10.0

## Standard Use Case: TDM

Figure 5-27, shows an example of the audio data path are transferred which use TDM renderer function.

External Memory

Speaker

SRC

DVC

SCU

SSI

ADMAC/

ADMACpp

ADMAC/

ADMACpp

TDM PCM data

Figure 5‑27 Audio data path for TDM renderer function

Figure 5-28, shows an example of the audio data path are transferred which use TDM capture function.

External Memory

Microphone

SRC

DVC

SCU

SSI

ADMAC/

ADMACpp

ADMAC/

ADMACpp

TDM PCM data



Figure 5‑28 Audio data path for TDM capture function

Table 5‑7 Supported features for OMX TDM Interface

|  |  |
| --- | --- |
| Name | Usage |
| Audio-DMAC/ Audio-DMACpp | Transfer data between two audio modules; memory and audio module. |
| SSI | Send audio data to speaker (SSI30)  Receive audio data from microphone (SSI40) |
| SRC | Convert sampling rate between 32 kHz, 44.1 kHz and 48 kHz. |
| DVC | Volume control  Non-supported setting in runtime |

Two below table show parameter of data is supported.

Table 5‑8 Supported data for TDM Renderer

| Item | Description |
| --- | --- |
| Input data format | |  |  |  |  |  | | --- | --- | --- | --- | --- | | Channel number | | PCM bit-width (fix-point) | | Sample rate  (Hz) | | 16-bit | 24-bit | 48000/44100/32000 | | 6ch | 3 x 2ch | O | O | O | | 1 x 6ch | O | O | O | | 8ch | 4 x 2ch | O | O | O | | 1 x 8ch | O | O | O | |
| Output data format | |  |  |  |  |  | | --- | --- | --- | --- | --- | | Channel number | | PCM bit-width (fix-point) | | Sample rate  (Hz) | | 16-bit | 24-bit | 44100/48000 | | 6ch | 3 x 2ch | O | O | O | | 1 x 6ch | O | O | O | | 8ch | 4 x 2ch | O | O | O | | 1 x 8ch | O | O | O | |
| Frame size | 1024 bytes |

Table 5‑9 Supported data for TDM Capture

| Item | Description |
| --- | --- |
| Input data format | |  |  |  |  |  | | --- | --- | --- | --- | --- | | Channel number | | PCM bit-width (fix-point) | | Sample rate  (Hz) | | 16-bit | 24-bit | 44100/48000 | | 6ch | 3 x 2ch | O | O | O | | 1 x 6ch | O | O | O | | 8ch | 4 x 2ch | O | O | O | | 1 x 8ch | O | O | O | |
| Output data format | |  |  |  |  |  | | --- | --- | --- | --- | --- | | Channel number | | PCM bit-width (fix-point) | | Sample rate  (Hz) | | 16-bit | 24-bit | 48000/44100/48000 | | 6ch | 3 x 2ch | O | O | O | | 1 x 6ch | O | O | O | | 8ch | 4 x 2ch | O | O | O | | 1 x 8ch | O | O | O | |
| Frame size | 1024 bytes |

Implementation:

O: Supported.

X: Not supported.

User can use ADSP Smoketest application to run TDM Capture.

ADSP Smoketest program usage guideline:

#./adsp-omx-tdm-launch -card 0,1 -i ssi1 –w <value> -chmod <value> -o1 <name> -o2 <name> -o3 <name> -o4 <name> -l <value> [-<command> <value>]

Explain:

-i ssi1: Use TDM Capture.

-card 0,1: Select rcar-sound card is to open codec.

–w <value>: PCM Bit per sample (16/24).

–chmod <value>: Channel mode.

|  |  |
| --- | --- |
| Value | Description |
| 0 | 4 streams stereo |
| 1 | 1 stream 8 channels |
| 3 | 3 streams stereo |
| 4 | 1 stream 6 channels |

–o1 <name>: 1st output stream (.pcm).

–o2 <name>: 2nd output stream (.pcm).

–o3 <name>: 3rd output stream (.pcm).

–o4 <name>: 4th output stream (.pcm).

–l <value>: Recording time (second).

–capfs <value>: Input sampling frequency.

(44100/48000, set '0' to disable SRC module).

-capoutfs <value>: Output sampling frequency (32000/44100/48000).

–capdmachannel1 <value>: 1st DMA channel.

(ADMAC\_CH00 to ADMAC\_CH31)/(ADMACPP\_CH00 to ADMACPP\_CH28).

–capinsource1 <value>: 1st input device.

(SSI40)/(SCU\_SRCI0, SCU\_SRCI1, SCU\_SRCI3, SCU\_SRCI4).

–capdmachannel2 <value>: 2nd DMA channel.

(ADMAC\_CH00 to ADMAC\_CH31)/(ADMACPP\_CH00 to ADMACPP\_CH28)

-capinsource2 <value>: 2nd input device (SSI40/NONCONFIG).

If capinsource1 is 'SSI40', this value must be 'NONCONFIG'.

-capvol <value>: Volume gain (0-8, set 'FFFFFFFF' to disable DVC module).

-capframe <value>: Frame size (1024).

User can use ADSP Smoketest application to run TDM Renderer.

ADSP Smoketest program usage guideline:

#./adsp-omx-tdm-launch -card 0,1 -o ssi0 –w <value> -chmod <value> -o1 <name> -o2 <name> -o3 <name> -o4 <name> -l <value> [-<command> <value>]

Explain:

-o ssi0: Use TDM Renderer.

-card 0,1: Select rcar-sound card is to open codec.

–w <value>: PCM Bit per sample (16/24).

–chmod <value>: Channel mode.

|  |  |
| --- | --- |
| Value | Description |
| 0 | 4 streams stereo |
| 1 | 1 stream 8 channels |
| 3 | 3 streams stereo |
| 4 | 1 stream 6 channels |

–i1 <name>: 1st input stream (.pcm).

–i2 <name>: 2nd input stream (.pcm).

–i3 <name>: 3rd input stream (.pcm).

–i4 <name>: 4th input stream (.pcm).

–rdrfs <value>: Input sampling frequency (32000/44100/48000).

-rdroutfs <value>: Output sampling frequency.

(44100/48000, set '0' to disable SRC module).

–rdrdmachannel1 <value>: 1st DMA channel.

(ADMAC\_CH00 to ADMAC\_CH31)/(ADMACPP\_CH00 to ADMACPP\_CH28).

–rdroutsource1 <value>: 1st output device.

(SSI30)/(SCU\_SRCI0, SCU\_SRCI1, SCU\_SRCI3, SCU\_SRCI4).

–rdrdmachannel2 <value>: 2nd DMA channel.

(ADMAC\_CH00 to ADMAC\_CH31)/(ADMACPP\_CH00 to ADMACPP\_CH28)

-rdroutsource2 <value>: 2nd output device (SSI30/NONCONFIG).

If rdroutsource1 is 'SSI30', this value must be 'NONCONFIG'.

-rdrvol <value>: Volume gain (0-8, set 'FFFFFFFF' to disable DVC module).

-rdrframe <value>: Frame size (1024).

[Note] Do not support to convert sampling rate between 32000 and 44100 Hz, and between 48000 and 44100 Hz.

### TDM Renderer flow 1: Normal

In this flow, TDM without data conversion data is transfer to speaker.

External Memory

SSI

ADMAC/

ADMACpp

Figure 5‑29 Data path for TDM renderer

* Example use smoke test:

#./adsp-omx-tdm-launch -card 0,1 -w 16 -chmod 3 -o ssi0 -i1 input1.pcm -i2 input2.pcm -i3 input3.pcm -rdrfs 32000 -rdrdmachannel1 ADMACPP\_CH01 -rdroutsource1 SSI30

### TDM Renderer flow 2: Sampling Rate Conversion

In this flow, TDM data, which is converted from input sample rate to output sample rate, is transfer to speaker.

External Memory

SRC

SSI

ADMAC/

ADMACpp

ADMAC/

ADMACpp

Figure 5‑30 Data path for TDM renderer use sample rate conversation

* Example use smoke test:

#./adsp-omx-tdm-launch -card 0,1 -w 16 -chmod 0 -o ssi0 -i1 input1.pcm -i2 input2.pcm -i3 input3.pcm -i4 input4.pcm -rdrfs 32000 -rdroutfs 48000 -rdrdmachannel1 ADMACPP\_CH01 -rdroutsource1 SSI30 -rdrdmachannel2 ADMACPP\_CH00 -rdroutsource2 NONCONFIG -rdrvol FFFFFFFF

### TDM Renderer flow 3: Volume Control

In this flow, TDM data is changed volume, then transferred to speaker.

External Memory

SRC

DVC

SSI

ADMAC/

ADMACpp

ADMAC/

ADMACpp

Figure 5‑31 Data path for TDM renderer use volume control

* Example use smoke test:

#./adsp-omx-tdm-launch -card 0,1 -w 16 -chmod 0 -o ssi0 -i1 input1.pcm -i2 input2.pcm -i3 input3.pcm -i4 input4.pcm -rdrfs 32000 -rdroutfs 48000 -rdrdmachannel1 ADMACPP\_CH01 -rdroutsource1 SCU\_SRCI0 -rdrdmachannel2 ADMACPP\_CH00 -rdroutsource2 SSI30 -rdrvol 1

### TDM Capture flow 1: Normal

In this flow, data without data conversion is transferred from microphone to external memory

External Memory

SSI

ADMAC/

ADMACpp



Figure 5‑32 Data path for TDM Capture nomal

* Example use smoke test:

#./adsp-omx-tdm-launch -card 0,1 -w 16 -chmod 3 -i ssi1 -o1 out1.pcm -o2 out2.pcm -o3 out3.pcm -capoutfs 44100 -capdmachannel1 ADMACPP\_CH01 -capinsource1 SSI40

### TDM Capture flow 2: Sampling Rate Conversion

In this flow, data, which is converted from input sample rate to output sample rate, is transferred from microphone to external memory

External Memory

SRC

SSI

ADMAC/

ADMACpp

ADMAC/

ADMACpp



Figure 5‑33 Data path for TDM capture use sampling rate conversion

* Example use smoke test:

#./adsp-omx-tdm-launch -card 0,1 -w 16 -chmod 0 -i ssi1 -o1 out1.pcm -o2 out2.pcm -o3 out3.pcm -o4 out4.pcm -capfs 48000 -capoutfs 44100 -capdmachannel1 ADMACPP\_CH01 -capinsource1 SCU\_SRCI0 -capdmachannel2 ADMACPP\_CH00 -capinsource2 SSI40 -capvol FFFFFFFF

### TDM Capture flow 3: Volume Control

In this flow, data, which is changed volume, is transferred from microphone to external memory

External Memory

SRC

DVC

SSI

ADMAC/

ADMACpp

ADMAC/

ADMACpp



Figure 5‑34 Data path for TDM capture use volume control

* Example use smoke test:

#./adsp-omx-tdm-launch -card 0,1 -w 16 -chmod 0 -i ssi1 -o1 out1.pcm -o2 out2.pcm -o3 out3.pcm -o4 out4.pcm -capfs 48000 -capoutfs 44100 -capdmachannel1 ADMACPP\_CH01 -capinsource1 SCU\_SRCI0 -capdmachannel2 ADMACPP\_CH00 -capinsource2 SSI40 -capvol 1