I. Investigate terminology use in Audio: channel, PCM, Frame, Gain, Latency, Mono, Stereo, Sample, Volume, ADC, Codec, DSP, ALSA.

ADSP: DSPs are processors or microcomputers whose hardware, software, and instruction sets are optimized for high-speed numeric processing applications, an essential for processing digital data representing analog signals in real time.**The ADSP** takes over by capturing the digitized information and processing it. It then feeds the digitized information back for use in the real world.

Analog-to-digital converter (ADC): Module that converts an analog signal (continuous in time and amplitude) to a digital signal (discrete in time and amplitude)

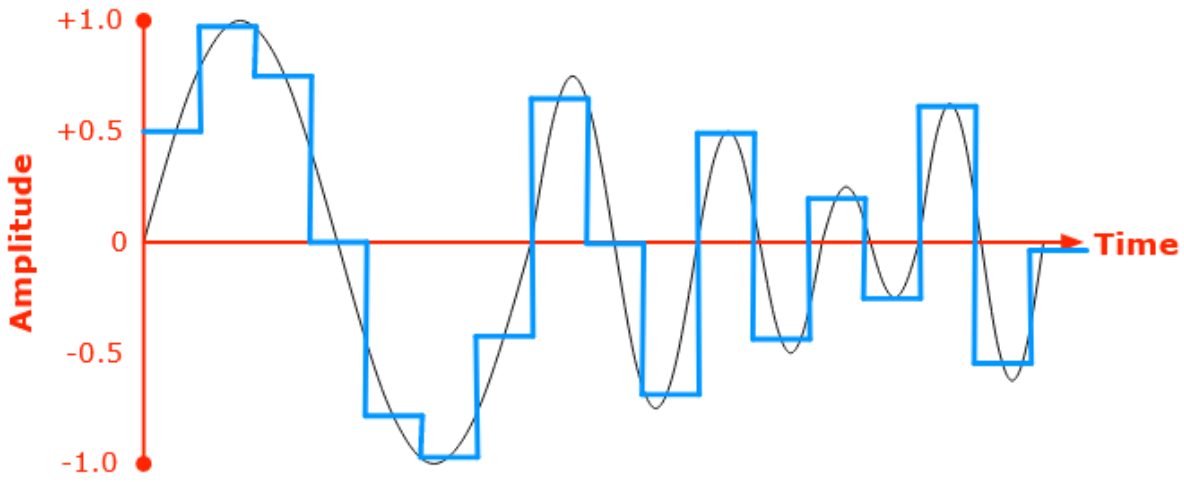
Pulse-Code Modulation (PCM) is the conventional method for converting analog audio into digital audio.

Audio bandwidth is the range of audio frequencies that the A/D converter is capable of capturing and converting into digital form.

Codec (**Co**der-**dec**oder). Module that encodes and/or decodes an audio signal from one representation to another (typically analog to PCM or PCM to analog).

Sample is the number representing the audio value for a single channel at a point in time. Analog signal is converted into digital form by a circuit that captures the incoming wave's amplitude at regular intervals, converting that data into a number in a form that is understood by the audio recording system. Each of these captured moments is a **sample**.

Sample rate: The number of samples taken per second is called the **sample rate**.



Frame is a set of samples, one per channel, at a point in time.

Channel: The position of each audio source within the audio signal is called a **channel**. Each channel contains a sample indicating the amplitude of the audio being produced by that source at a given moment in time. (Single stream of audio information, usually corresponding to one location of recording or playback.)

Mono: In monaural sound one single channel is used. It can be reproduced through several speakers, but all speakers are still reproducing the same copy of the signal.

Stereo: In stereophonic sound more channels are used (typically two). You can use two different channels and make one feed one speaker and the second channel feed a second speaker (which is the most common stereo setup). This is used to create directionality, perspective, space.

For 16-bit stereo audio, each sample taken from the analog signal is recorded as two 16-bit integers, one for the left channel and one for the right. That means each sample requires 32 bits of memory. At the common sample rate of 48 kHz (48,000 samples per second), this means each second of audio occupies 192 kB of memory. Therefore, a typical three-minute song requires about 34.5 MB of memory.

ALSA stands for **A**dvanced **L**inux **S**ound **A**rchitecture. It is a suite of hardware drivers, libraries and utilities which provide audio and MIDI functionality for the Linux operating system.

Gain: multiplicative factor greater than or equal to 1.0, applied to an audio signal to increase the signal level. Compare to *attenuation*.

Attenuation: multiplicative factor less than or equal to 1.0, applied to an audio signal to decrease the signal level. Compare to *gain*.

Audio latency is the time delay as an audio signal passes through a system.

Volume loudness, the subjective strength of an audio signal.

II. Read ADSP_know-how.pptx to understand scope of ADSP. Get overview about ADSP ALSA, ADSP Driver Extension, ADSP Framework, ADSP Plugin.

- ADSP ALSA.

Read focus on User Manual: overview, terminologies, list of usage.

- ADSP Driver Extension.

Read focus on User Manual: overview. Processing flow will need to read when investigate source codes.

- ADSP Framework.

Read focus on User Manual: overview, 2.1 state transition.

- ADSP Plugin.

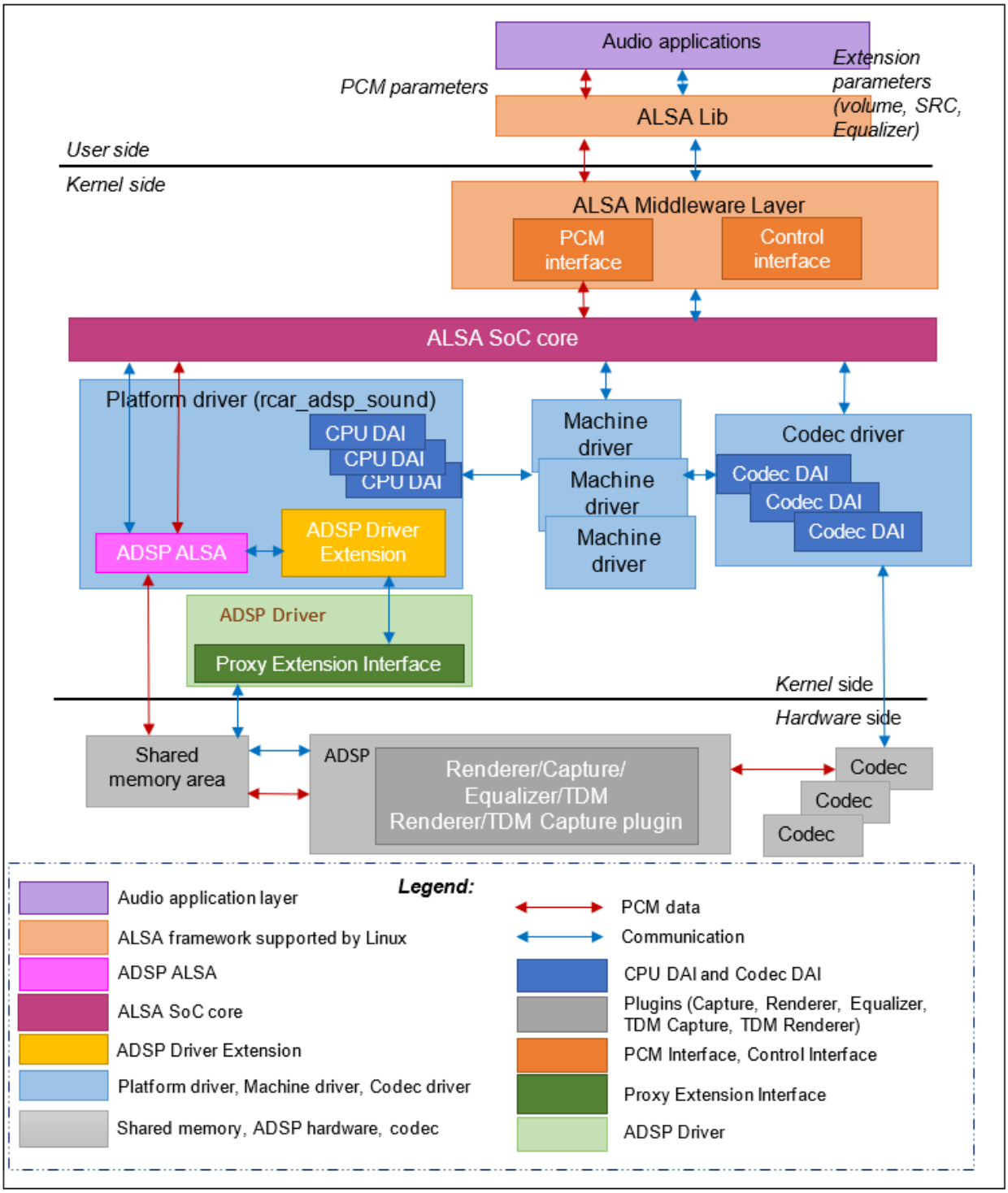
Run focus on User Manual: overview, Appendix.

+ To have basic knowledge about ADSP, you need to read as above guideline.

About Detail Design, It will provide detail of function, API and flow execute of function. Currently, you don't need care about it.

+ You also can read Related materials to more understand. Just read overview, flow, block diagram.

Overview of ADSP



Overview architecture of ADSP driver

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.

ADSP Driver Extension

It provides APIs for ADSP ALSA to connect to ADSP Driver (Proxy Extension Interface)

ADSP Driver (Proxy Extension Interface)

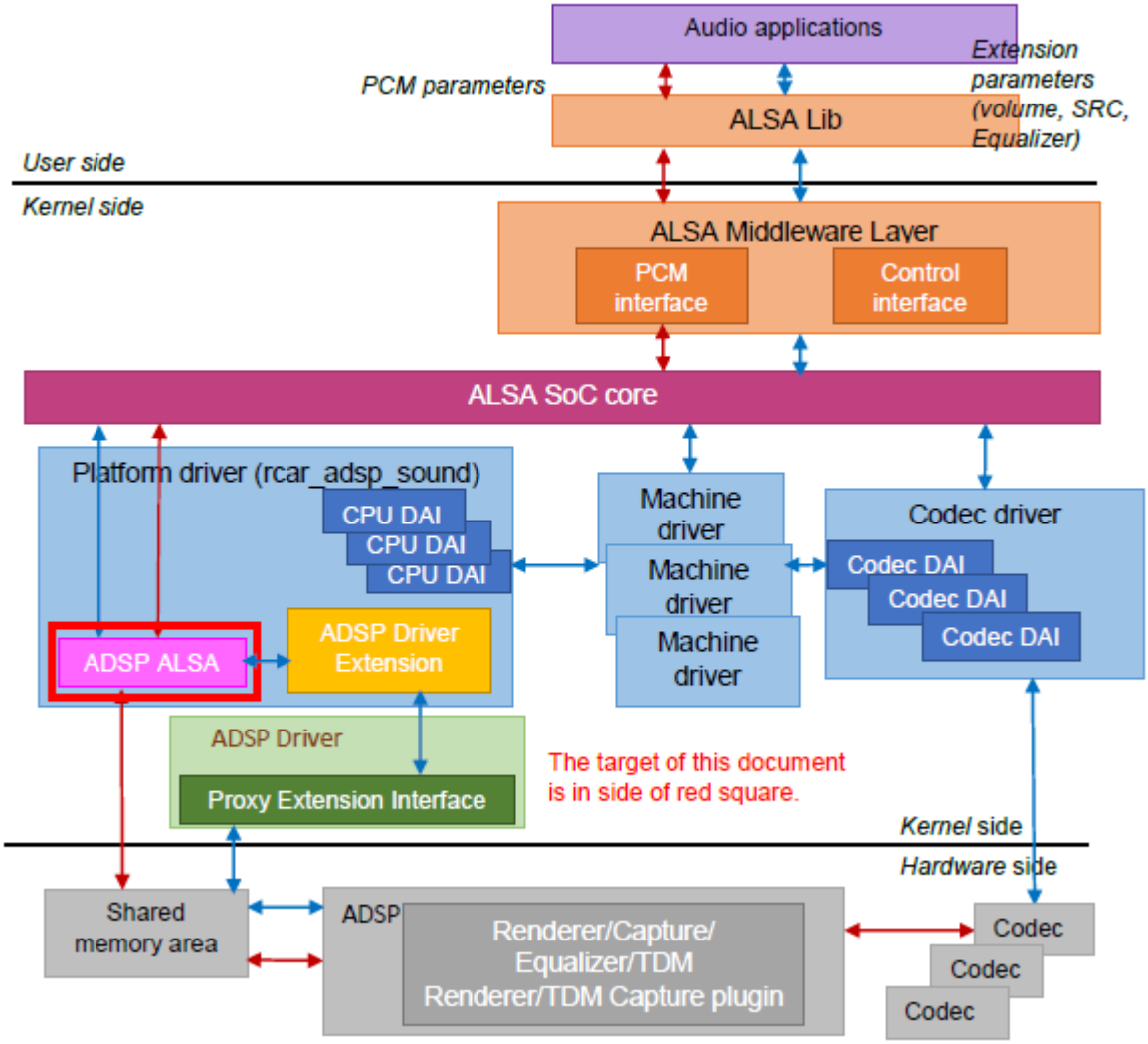
APIs of methods through which ADSP Driver Extension communicates with shared memory area in Hardware side.

CPU DAI

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

A. ADSP ALSA

1. Overview of ADSP ALSA

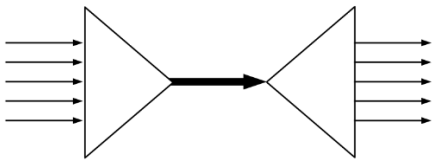


2. Some terminologies

- **Audio applications (tinycap, tinyplay, tinymix, 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

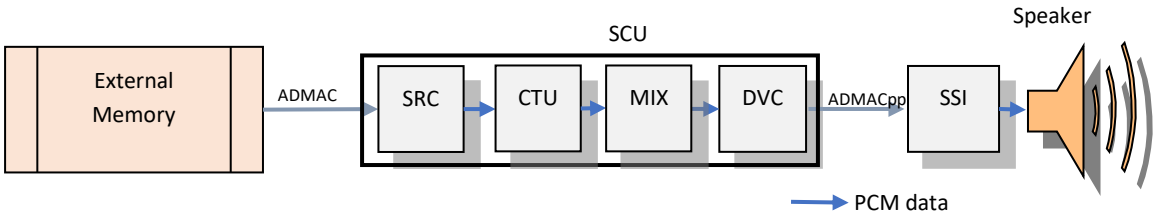
playback/TDM playback, it receives PCM data from user app and transfers to ADSP Renderer plugin/ADSP TDM Renderer plugin. For record, it receives PCM data from ADSP Capture plugin/ADSP TDM Capture plugin and transfers to user app. The equalization function can be integrated into playback and record by 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.
- **TDM**
  Time-division multiplexing (**TDM**) is a method of transmitting and receiving independent signals over a common signal path by means of synchronized switches at each end of the transmission line so that each signal appears on the line only a fraction of time in an alternating pattern.



3. List of usage

3.1. Playback



Data path for playback

3.1.1. Playback stream

ADSP ALSA Driver supports playback stream (PCM width 16/24, sample rate 32/44.1/48 kHz, frame size 4/8/16/32/64/128/256/512/1024, 1/2 channels, period count 2/4).

[Note]: In 16 bit mono case, ADSP ALSA Driver only supports frame size 1024. Other frame sizes are not guarantee.

- User setting:

  tinyplay thetest_FULL_s_32000_16.wav -D 0 -d 0 -p 64 -n 2 (optimized audio output latency)

  tinyplay thetest_FULL_s_32000_16.wav -D 0 -d 0 -p <framesize> -n <period count>

3.1.2. Setting

console: tinymix -D 0
Mixer name: 'audio-card'
Number of controls: 54

ctl	*type*	*num*	***name***
6	*INT*	*1*	*PlaybackVolume*
7	*INT*	*1*	*CaptureVolume*
8	*INT*	*1*	*PlaybackOutRate*
9	*INT*	*1*	*CaptureInRate*
10	*INT*	*55*	*PlaybackEQZControl*
11	*INT*	*55*	*CaptureEQZControl*
12	*INT*	*1*	*PlaybackEQZSwitch*
13	*INT*	*1*	*CaptureEQZSwitch*
14	*INT*	*1*	*PlaybackOutChannel*
15	*INT*	*1*	*PlaybackVolume*
16	*INT*	*1*	*CaptureVolume*
17	*INT*	*1*	*PlaybackOutRate*
18	*INT*	*1*	*CaptureInRate*
19	*INT*	*55*	*PlaybackEQZControl*
20	*INT*	*55*	*CaptureEQZControl*
21	*INT*	*1*	*PlaybackEQZSwitch*
22	*INT*	*1*	*CaptureEQZSwitch*

23	*INT*	*1*	*PlaybackOutChannel*
24	*INT*	*1*	*PlaybackVolume*
25	*INT*	*1*	*CaptureVolume*
26	*INT*	*1*	*PlaybackOutRate*
27	*INT*	*1*	*CaptureInRate*
28	*INT*	*55*	*PlaybackEQZControl*
29	*INT*	*55*	*CaptureEQZControl*
30	*INT*	*1*	*PlaybackEQZSwitch*
31	*INT*	*1*	*CaptureEQZSwitch*
32	*INT*	*1*	*PlaybackOutChannel*
33	*INT*	*1*	*PlaybackVolume*
34	*INT*	*1*	*CaptureVolume*
35	*INT*	*1*	*PlaybackOutRate*
36	*INT*	*1*	*CaptureInRate*
37	*INT*	*55*	*PlaybackEQZControl*
38	*INT*	*55*	*CaptureEQZControl*
39	*INT*	*1*	*PlaybackEQZSwitch*
40	*INT*	*1*	*CaptureEQZSwitch*
41	*INT*	*1*	*PlaybackOutChannel*

Example:

- Run stream

tinyplay thetest_FULL_s_48000_16.wav -D 0 -d 0

- **Set volume**

tinymix -D 0 PlaybackVolume 100 <0-800>
or
tinymix -D 0 6 100

- **Set output sample rate**

tinymix -D 0 PlaybackOutRate 48000 <32/44.1/48 kHz>
or
tinymix -D 0 8 48000

- **Set output channel number**

tinymix -D 0 PlaybackOutChannel 2
or
tinymix -D 0 14 2

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 0-1 List of channel number conversation

O means supported
X means unsupported

- Get information:

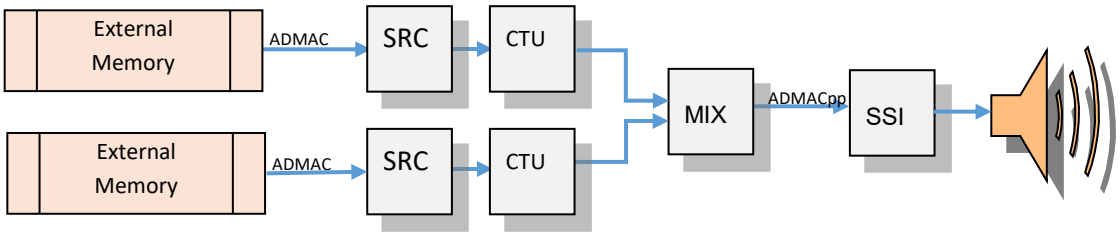
tinymix -D 0 <name>
or
tinymix -D 0 <ctl>

3.1.3. Mix function

ADSP ALSA Driver supports mixing multi (2/3/4) playback stream with same sample rate.
But due to hardware performance and memory, some limitation showed as below:

- In case routing between Equalizer and Renderer playback 24 bit stream only mixing 2 stream is supported.
- H3 can support mixing 2, 3 or 4 stream but M3 only supports mixing 2 stream.

Example: mix 2 playback streams



Data path when mixing 2 streams

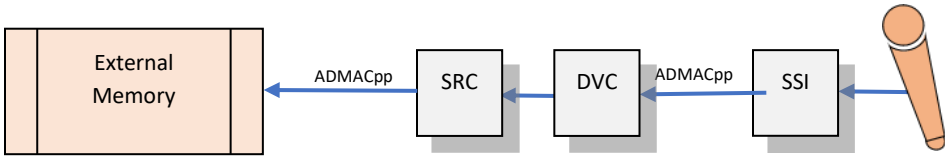
- User setting
  - Playback 1st stream:

    tinyplay thetest_FULL_s_44100_16.wav -D 0 -d 0

  - Playback 2nd stream:

    tinymix -D 0 17 44100
    tinyplay thetest_FULL_s_48000_16.wav -D 0 -d 1

3.2. Capture



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 0-2 Detailed information of ADSP ALSA sound card

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

3.2.1. Capture stream

ADSP ALSA Driver supports recording stream (PCM width 16/24, sample rate 32/44.1/48 kHz, frame size 4/8/16/32/64/128/256/512/1024, channel 1/2, period count 2/4).

[Note]: In 16 bit mono case, ADSP ALSA Driver only supports frame size 1024. Other frame sizes are not guarantee.

- User setting:

tinycap cap_s_32000_16.wav -D 0 -d 3 -c2 -r32000 -b 16 -T 5 -p <frame size> -n <period count>

tinycap cap_s_32000_16.wav -D 0 -d 3 -c2 -r32000 -b 16 -T 5 -p 64 -n 2 (Optimized audio input latency)

3.2.2. Setting

- **Record stream**

tinycap thetest_FULL_s_32000_16.wav -D 0 -d 0 -c2 -r32000 -b 16 -T 5

- **Set volume**

tinymix -D 0 CaptureVolume 50 <0-800>
or
tinymix -D 0 7 50

- **Set input sample rate**

tinymix -D 0 CaptureInRate 44100 <32/44.1/48 kHz>
or
tinymix -D 0 9 44100

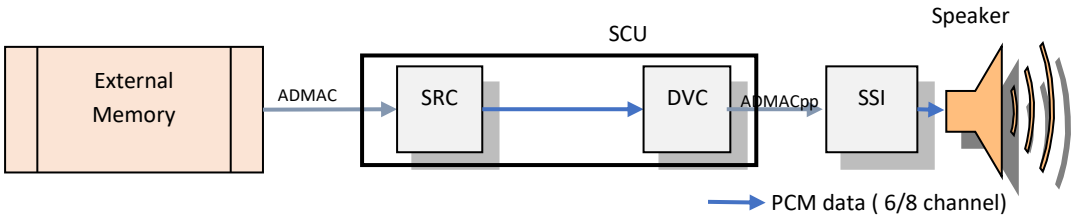
- **Get information about volume**

tinymix -D 0 CaptureVolume

or

tinymix -D 0 7

3.3. TDM Playback



Data path for multichannel

Command is used when running aplay to play TDM stream:

tinyplay <input> -D 0 -d 0

Explanation:

-D 0 -d 0:	Card selected is 0, DAI index is 0, sub-device is 0
<input>:	input file (.wav)

3.3.1. TDM Playback stream

ADSP ALSA Driver supports run multichannel stream (PCM width 16/24, 6/8 channels, frame size 1024, sample rate 44.1/48 kHz). If user run a stream 32 kHz, must convert sample rate to 44.1/48 kHz.

[Note] Do not support sample rate conversion between 32 and 44.1 kHz, and between 48 and 44.1 kHz.

- User setting
  - Set output sample rate if it is different from 48kHz
    tinymix -D 0 TDMPlaybackOutRate 48000

  - Playback multichannel stream
    tinyplay thetest_FULL_6ch_32000_16.wav -D 0 -d 0

  - Get information about sample rate
    tinymix -D 0 TDMPlaybackOutRate

3.3.2. Setting

Setting TDM Playback Volume

ADSP ALSA Driver supports setting volume for TDM Playback stream. Value range from 0 to 799. But updating volume runtime is unsupported. So user needs to set volume value before running multichannel stream.

- User setting
  - **Set volume**
    tinymix -D 0 TDMPlaybackVolume 100

  - **Playback multichannel stream**
    tinyplay thetest_FULL_6ch_48000_16.wav -D 0 -d 0

- **Get information about volume**

  tinymix -D 0 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**

  tinymix -D 0 TDMPlaybackOutRate 48000

- **Run stream**

  tinyplay thetest_FULL_6ch_44100_16.wav -D 0 -d 0

- **Get information output sample rate**

  tinymix -D 0 TDMPlaybackOutRate

3.4. TDM Capture

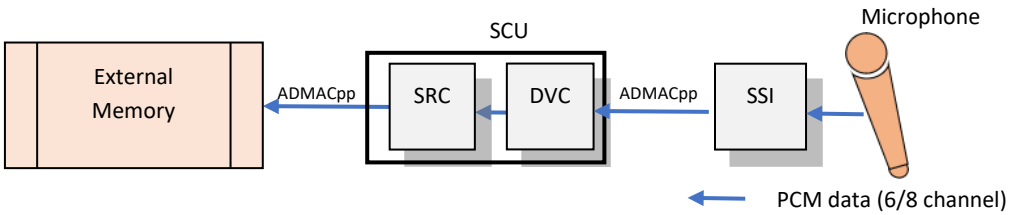


Figure 0-1 Data path for recording multichannel stream

Command are used when running tinycap to record multichannel stream:

tinycap <output> -D 0 -d 0 –c<value> -r<value> -b<name> -T <value>

Explanation:

-D 0 -d 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)
-b<name>:	Format of PCM width
-T <value>:	Recording duration (second)
<output>:	Output file is raw type (.wav)

3.4.1. TDM Capture stream

ADSP ALSA Driver supports recording multichannel stream (PCM width 16/24, sample rate 32/44.1/48 KHz, frame size 1024, channel 6/8). If user record a stream 32 kHz, must convert input sample rate to 44.1/48 kHz.

[Note] Do not support sample rate conversion between 32 and 44.1 kHz, and between 48 and 44.1 kHz.

- User setting

  tinycap output.wav -D 0 -d 0 -c8 -r48000 -b 16 -T 15

3.4.2. Setting

Setting TDM Capture Volume

ADSP ALSA Driver supports setting volume for record multichannel stream. Value range from 0 to 799. However, updating volume runtime is unsupported. Value 799 means that increase volume to 8 times.

- User setting
  - **Set volume**

    tinymix -D 0 TDMCaptureVolume 100

  - **Record multichannel stream**

    tinycap out.wav -D 0 -d 0 -c8 -r48000 -b 16 -T 15

  - **Get information about volume**

    tinymix -D 0 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**

    tinymix -D 0 TDMCaptureInRate 48000

  - **Record stream**

    tinycap output.wav -D 0 -d 0 -c8 -r44100 -b 16 -T 15

  - **Get information input sample rate**

    tinymix -D 0 TDMCaptureInRate

3.5. 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	0.125 dB	Specify gain at a center frequency of a peaking filter.
Base Gain	Fixed point decimal (Q4.28)	-10dB to 10dB	0.125 dB	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

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

User setting parameters for Graphic Equalizer

3.5.1. Equalizer for Playback

Setting Parametric Equalizer

- ➢ Setting flow
  - Enable Equalizer control

    tinymix -D 0 PlaybackEQZSwitch 1

    or

    tinymix -D 0 12 1

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

- Set Parametric Equalizer

  tinymix -D 0 PlaybackEQZControl 0 1 0 15000 94891933 268435456 268435456 2 0 15000 94891933 268435456 268435456 3 0 15000 94891933 268435456 268435456 4 0 15000 94891933 268435456 268435456 5 0 15000 94891933 268435456 268435456 6 0 15000 94891933 268435456 268435456 7 0 15000 94891933 268435456 268435456 8 0 15000 94891933 268435456 268435456 9 0 15000 94891933 268435456 268435456

  or

  tinymix -D 0 10 0 1 0 15000 94891933 268435456 268435456 2 0 15000 94891933 268435456 268435456 3 0 15000 94891933 268435456 268435456 4 0 15000 94891933 268435456 268435456 5 0 15000 94891933 268435456 268435456 6 0 15000 94891933 268435456 268435456 7 0 15000 94891933 268435456 268435456 8 0 15000 94891933 268435456 268435456 9 0 15000 94891933 268435456 268435456

  Please refer to *Set Equalizer parameters of hardware* detail information about the order and range for each parameters.

- Playback stream:
  tinyplay thetest_FULL_s_48000_16.wav -D 0 -d 0 -p 1024

- Get information of Equalizer parameter:
  tinymix -D 0 PlaybackEQZControl

- Get information of Equalizer status:

  tinymix -D 0 PlaybackEQZSwitch

Setting Graphic Equalizer

- Setting flow
  - Enable Equalizer control
    tinymix -D 0 PlaybackEQZSwitch 1
    or
    tinymix -D 0 12 1

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

  - Set Graphic Equalizer parameter:
    tinymix -D 0 PlaybackEQZControl 1 1 84886744 2 84886744 3 84886744 4 84886744 5 84886744
    or
    tinymix -D 0 10 1 1 84886744 2 84886744 3 84886744 4 84886744 5 84886744

    Please refer to *Set Equalizer parameters of hardware* detail information about the order and range for each parameters.

  - Playback stream:
    tinyplay thetest_FULL_s_48000_16.wav -D 0 -d 0 -p 1024

- Get information of Equalizer parameter:
  tinymix -D 0 PlaybackEQZControl

- Get information of Equalizer status:
  tinymix -D 0 PlaybackEQZSwitch

3.5.2. Equalizer for Capture

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

Equalizer plugin does not support setting in runtime. It only runs with frame size 1024.

Setting Parametric Equalizer

- Setting flow
  - Enable Equalizer control
    tinymix -D 0 CaptureEQZSwitch 1
    or
    tinymix -D 0 13 1

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

  - Set Parametric Equalizer
    tinymix -D 0 CaptureEQZControl 0 1 0 15000 94891933 268435456 268435456 2 0 15000 94891933 268435456 268435456 3 0 15000 94891933 268435456 268435456 4 0 15000 94891933 268435456 268435456 5 0 15000 94891933 268435456 268435456 6 0 15000 94891933 268435456 268435456 7 0 15000 94891933 268435456 268435456 8 0 15000 94891933 268435456 268435456 9 0 15000 94891933 268435456 268435456
    or
    tinymix -D 0 11 0 1 0 15000 94891933 268435456 268435456 2 0 15000 94891933 268435456 268435456 3 0 15000 94891933 268435456 268435456 4 0 15000 94891933 268435456 268435456 5 0 15000 94891933 268435456 268435456 6 0 15000 94891933 268435456 268435456 7 0 15000 94891933 268435456 268435456 8 0 15000 94891933 268435456 268435456 9 0 15000 94891933 268435456 268435456

    Please refer to *Set Equalizer parameters of hardware* detail information about the order and range for each parameters.

  - Record stream:
    tinycap cap_s_32000_16.wav -D 0 -d 0 -c2 -r32000 -b 16 -T 5 -p 1024

  - Get information of Equalizer parameter:
    tinymix -D 0 CaptureEQZControl

  - Get information of Equalizer status:
    tinymix -D 0 CaptureEQZSwitch

Setting Graphic Equalizer

- Setting flow
  - Enable Equalizer control

    tinymix -D 0 CaptureEQZSwitch 1

    or

    tinymix -D 0 13 1

    Value 1 to enable, 0 to disable Equalizer control

  - Set Graphic Equalizer parameter:

    tinymix -D 0 CaptureEQZControl 1 1 84886744 2 84886744 3 84886744 4 84886744 5 84886744

    or

    tinymix -D 0 11 1 1 84886744 2 84886744 3 84886744 4 84886744 5 84886744

    Please refer to *Set Equalizer parameters of hardware* detail information about the order and range for each parameters.

  - Record stream:

    tinycap cap_s_32000_16.wav -D 0 -d 0 -c2 -r32000 -b 16 -T 5 -p 1024

  - Get information of Equalizer parameter:

    tinymix -D 0 CaptureEQZControl

    or

    tinymix -D 0 11

  - Get information of Equalizer status:

    tinymix -D 0 CaptureEQZSwitch

    or

    tinymix -D 0 13

3.5.3. Apendix

Set Equalizer parameters of hardware

In playback case, the control name is **"PlaybackEQZControl"**. This string maps to the control defined in ADSP ALSA Driver.

In capture case, the control name is **"CaptureEQZControl"**. 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 center, 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 center	20 – 20000	9
Bandwidth	$0.2 \times 2^{27} – 15 \times 2^{27}$	9
Filter type	0 – 2	9
Gain base	20 - 20000	9
Gain	$10^{-10/20} \times 2^{28} – 10^{10/20} \times 2^{28}$	9
		Total: 55

Order of parameters to set for Parametric Equalizer:

Equalizer type, 9 x (filter index, frequency center, 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:

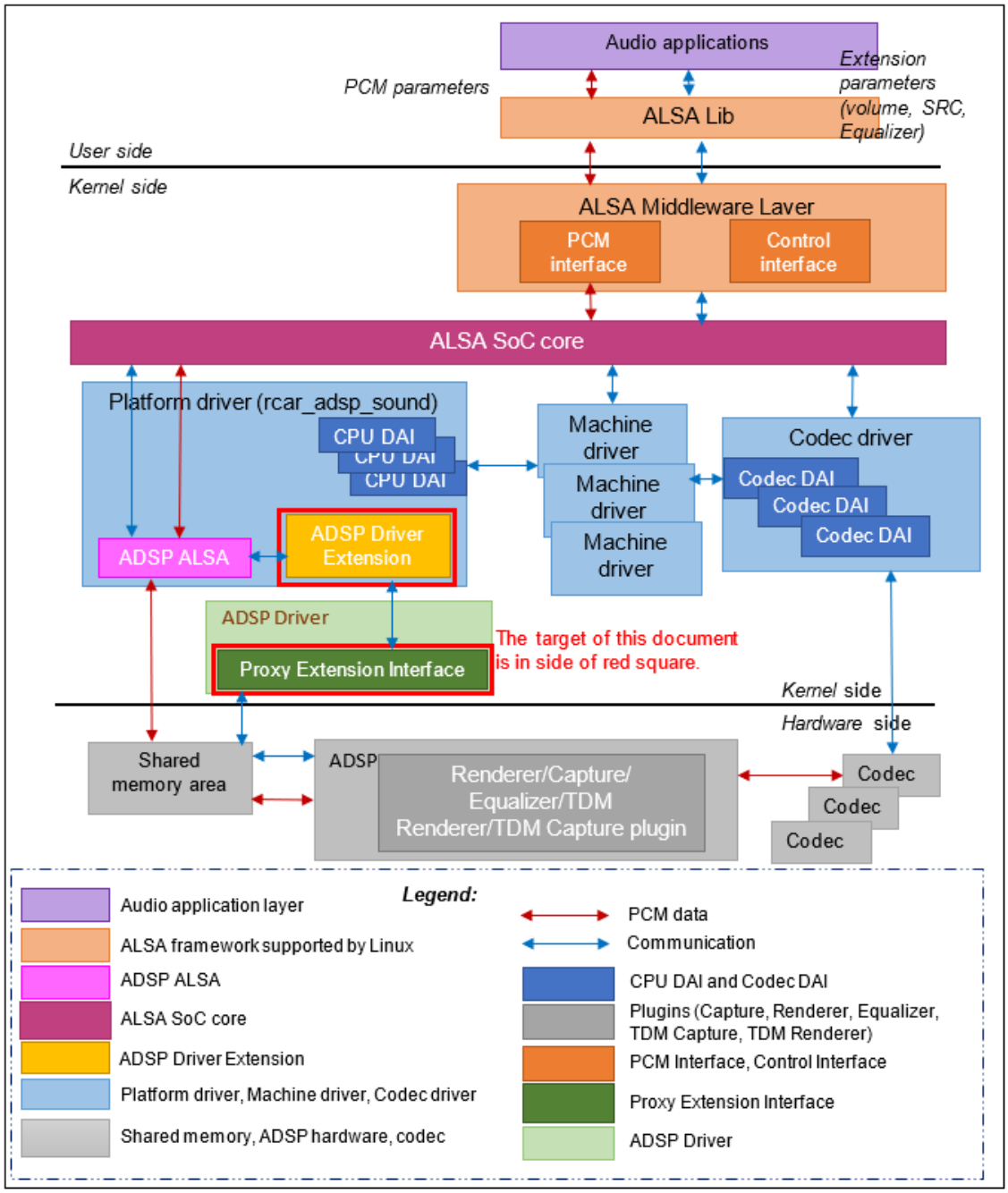
Parameters	**Value range**	**Number**
Equalizer type	1 (for Graphic)	1
Filter index	1 - 5	5
Graphic gain	$10^{-10/20} \times 2^{28} – 10^{10/20} \times 2^{28}$	5
		Total: 11

Order of parameters to set for Graphic Equalizer:

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

B. ADSP Driver Extension.

1.Overview of ADSP Driver Extension

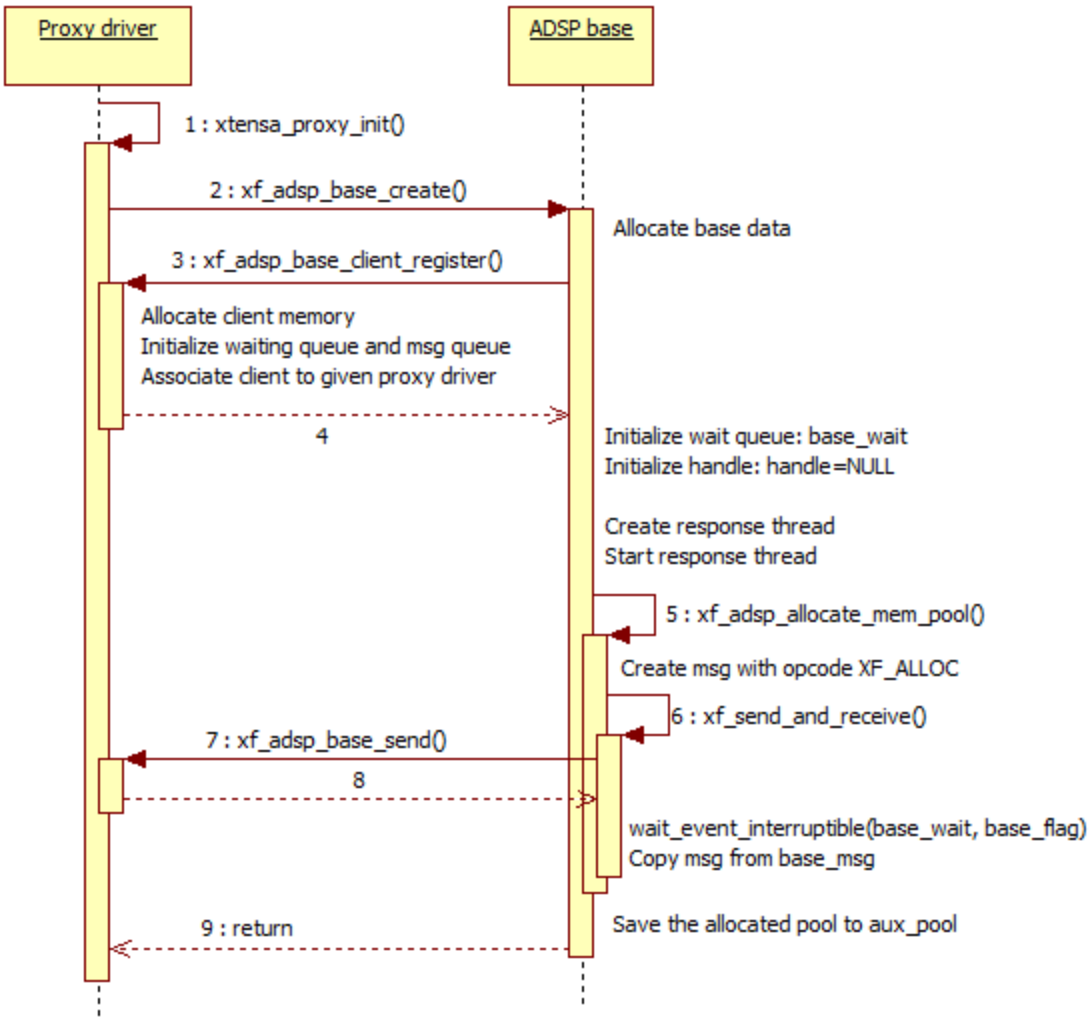


Overview configuration of ADSP driver

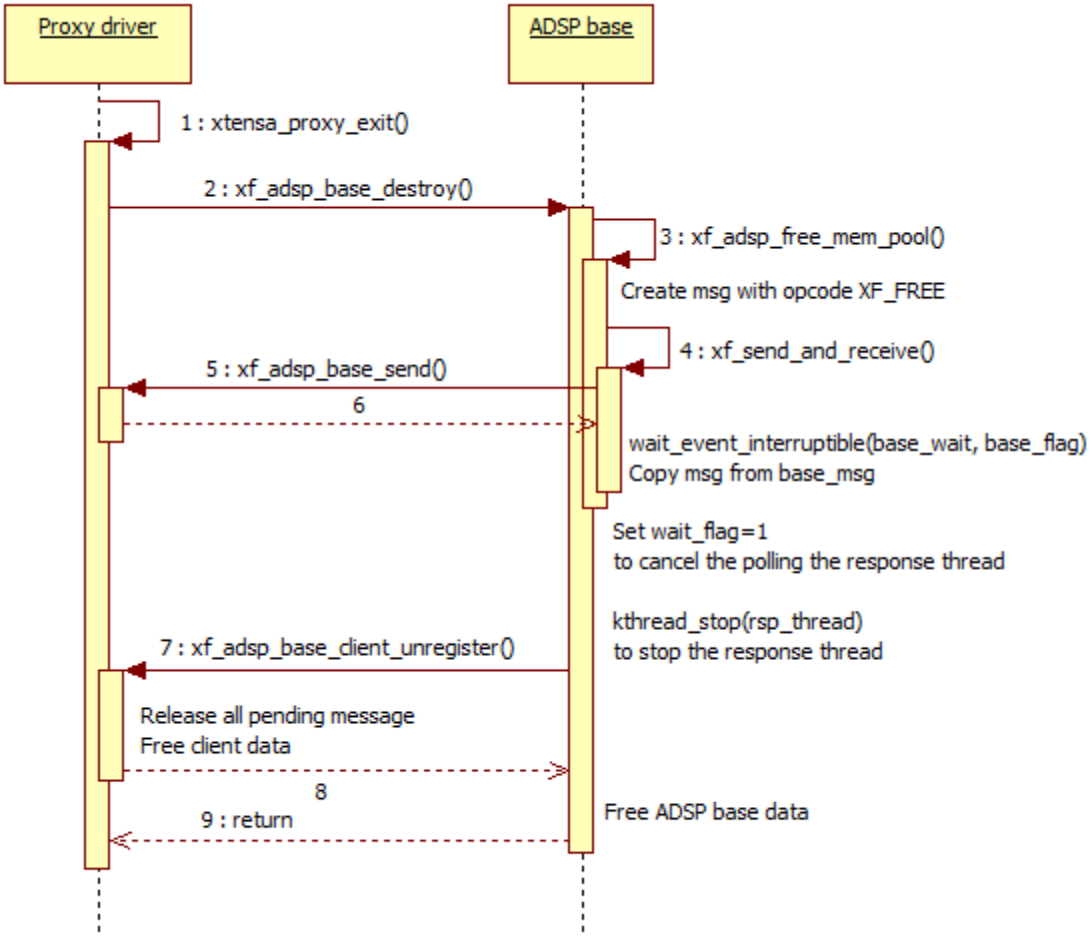
2. Processing flow

2.1. ADSP Base Flow

2.1.1.ADSP Base Creation

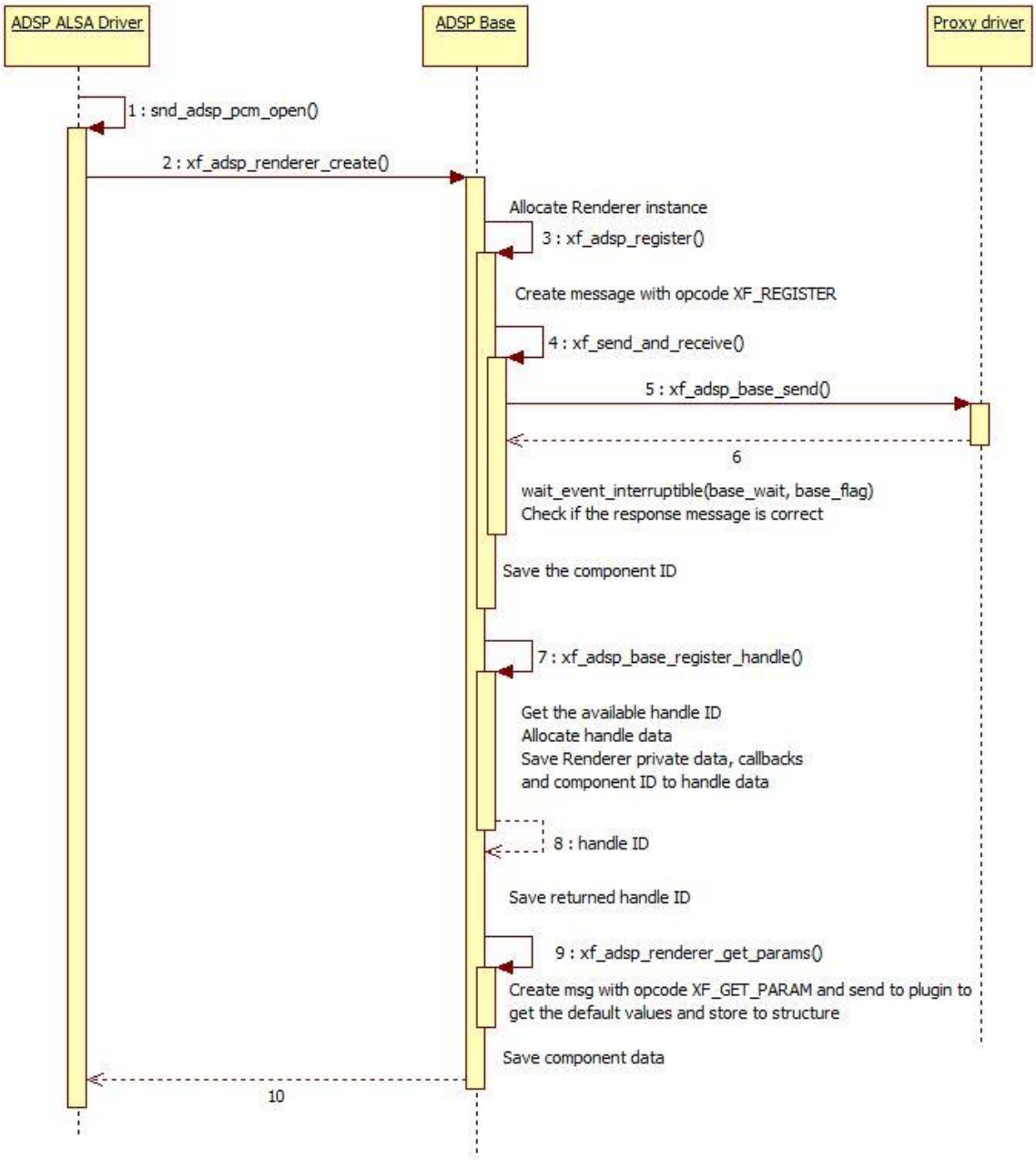


2.1.2.ADSP Base Destruction

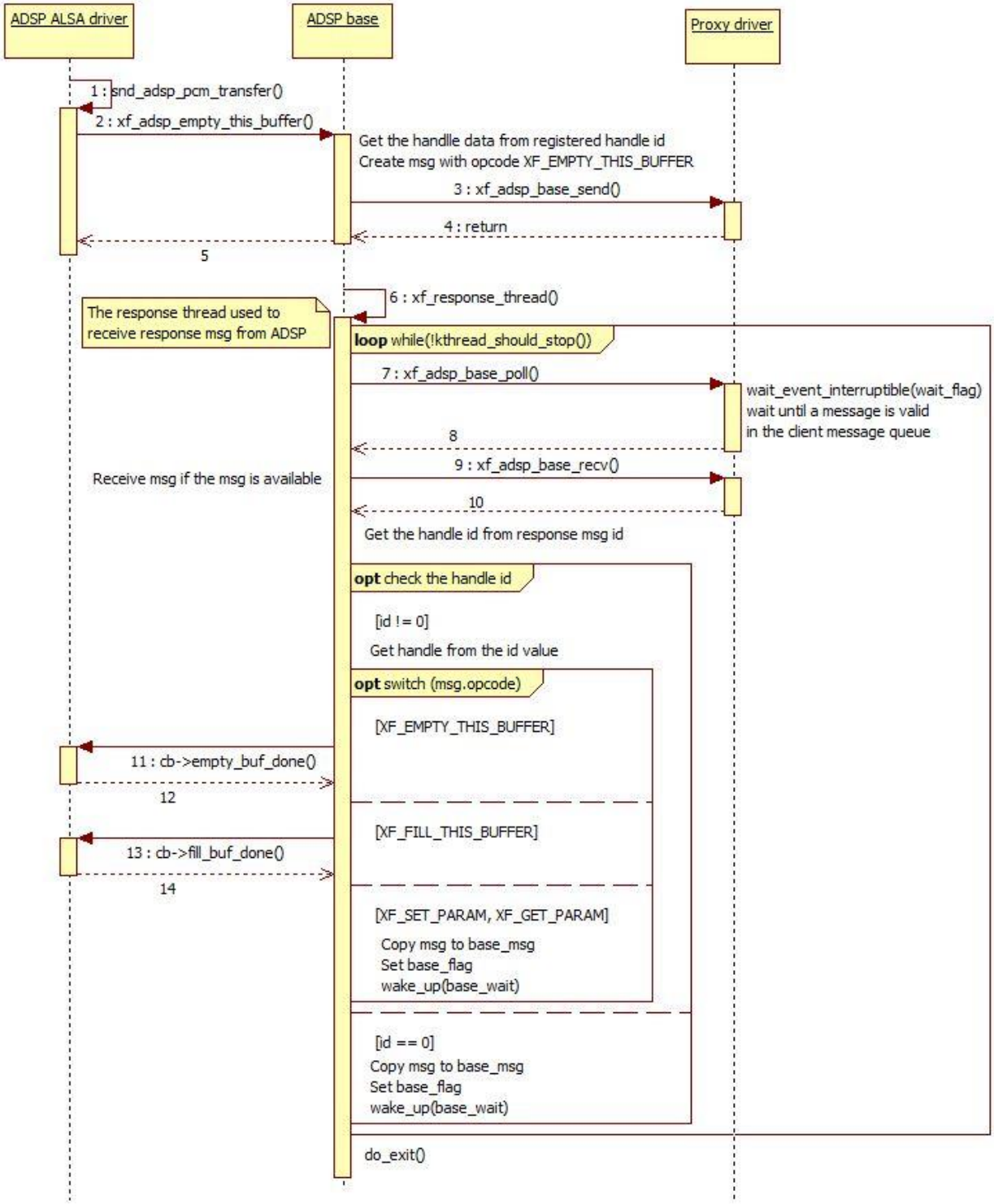


2.2. Renderer flow

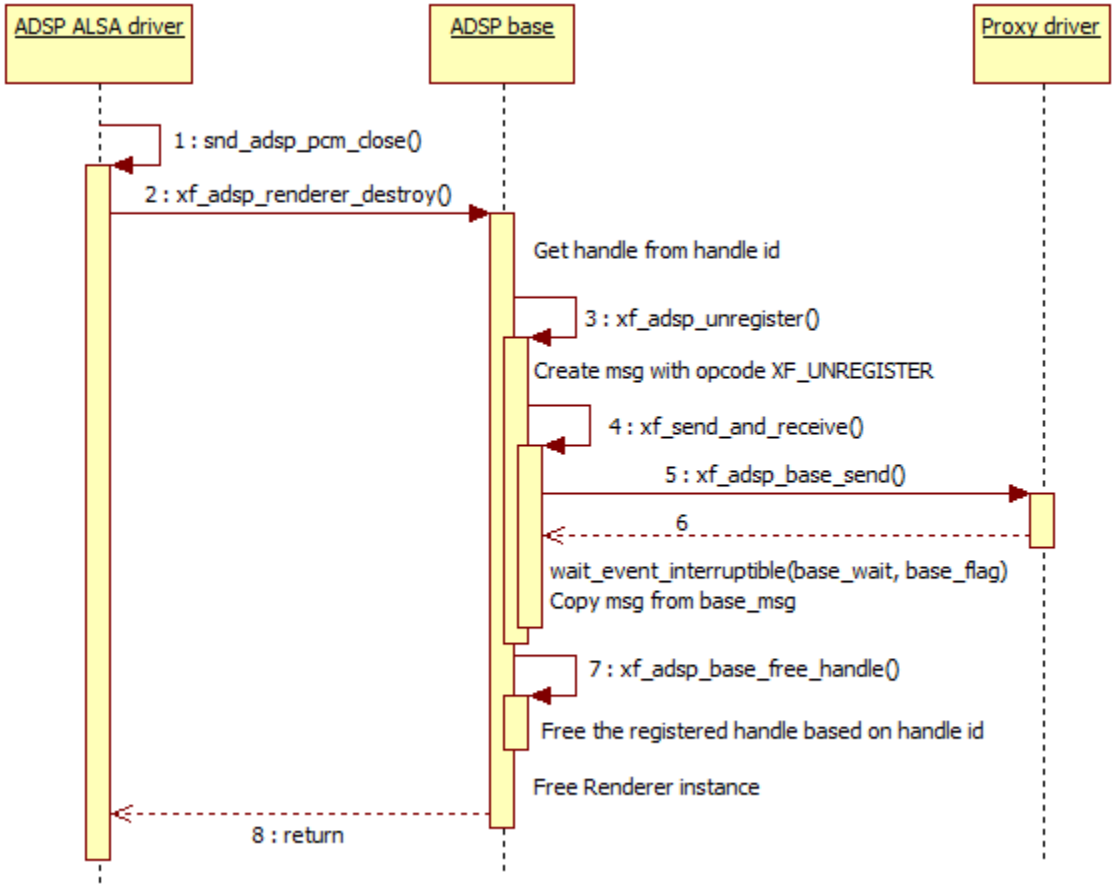
2.2.1. Renderer Rreation



2.2.2. Renderer Execution

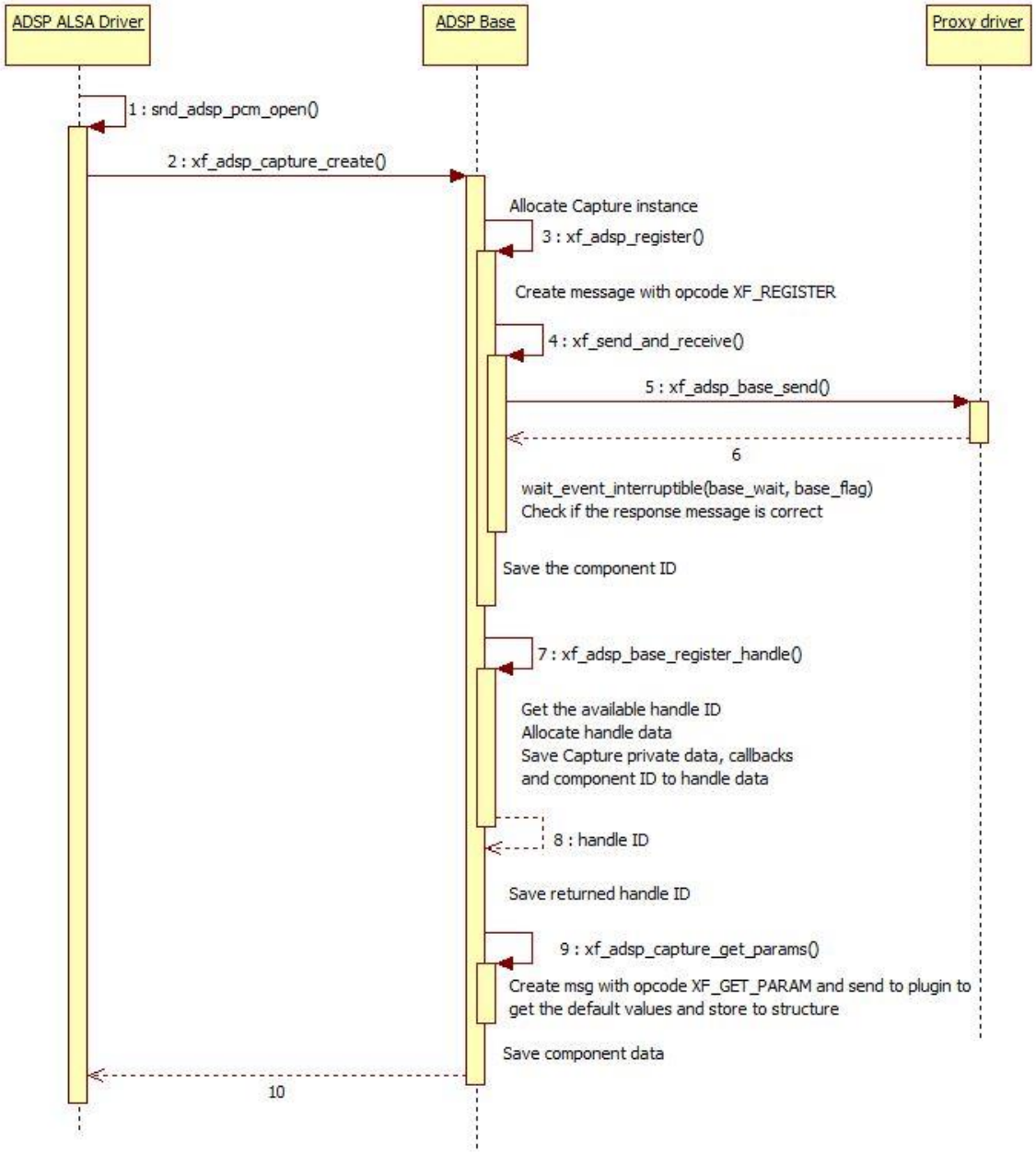


2.2.3. Renderer Destruction

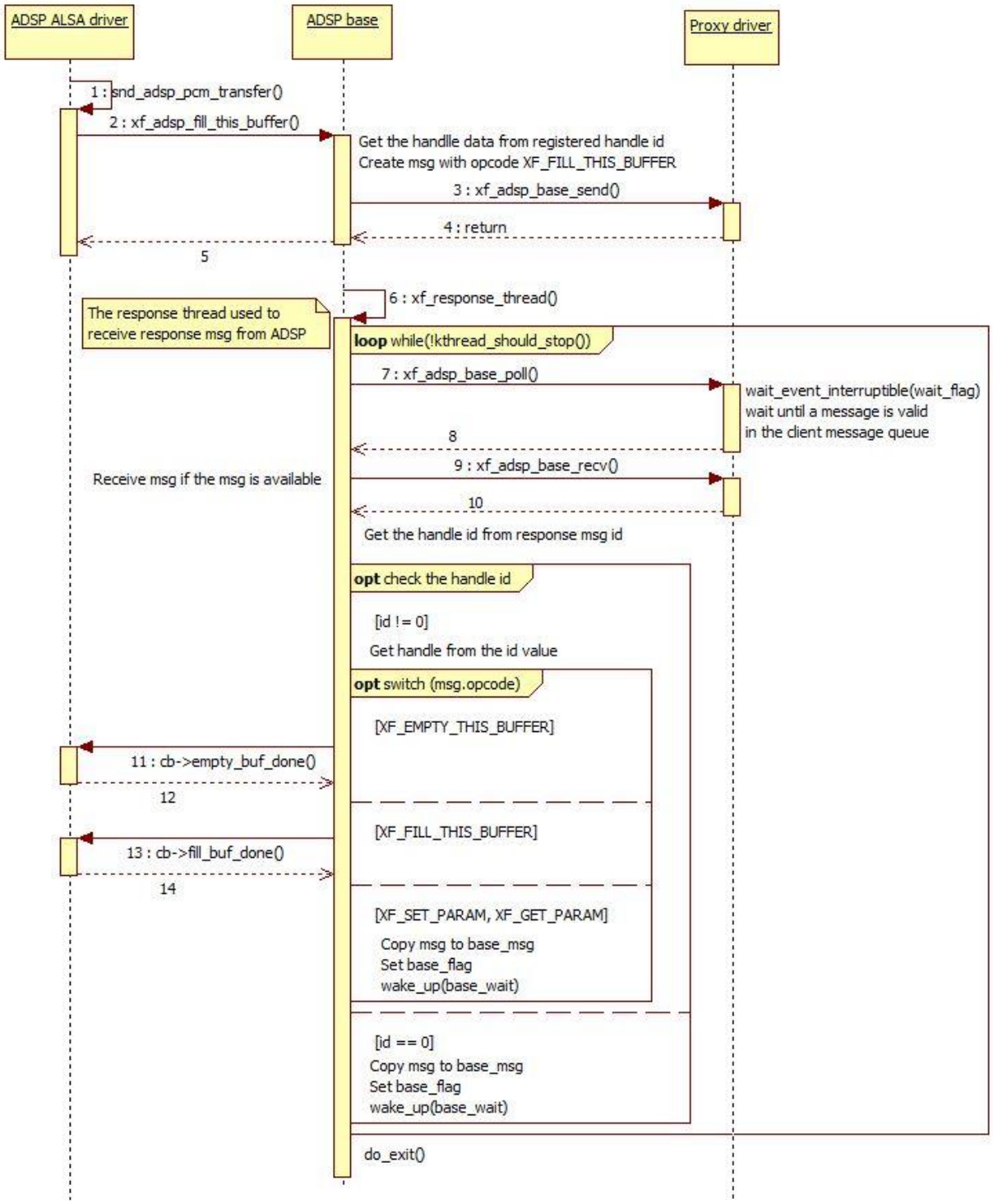


2.3. Capture flow

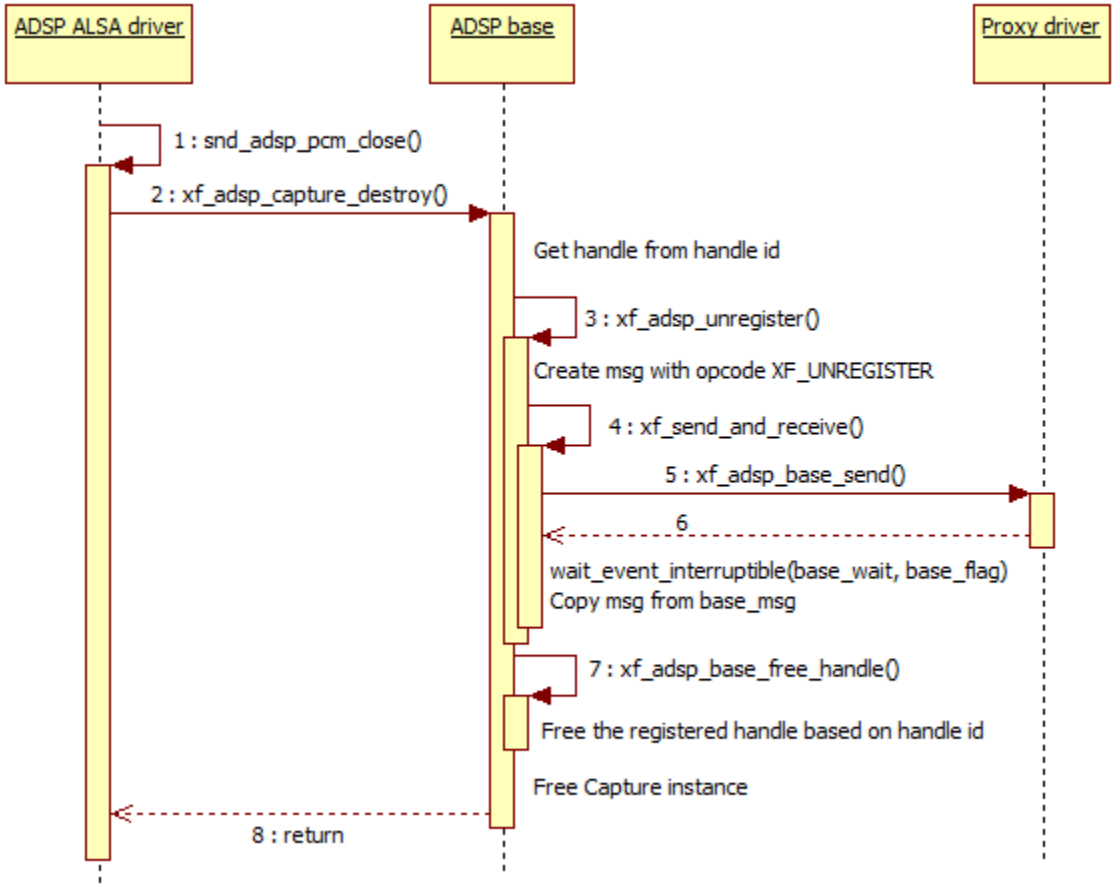
2.3.1. Capture Creation



2.3.2. Capture Execution

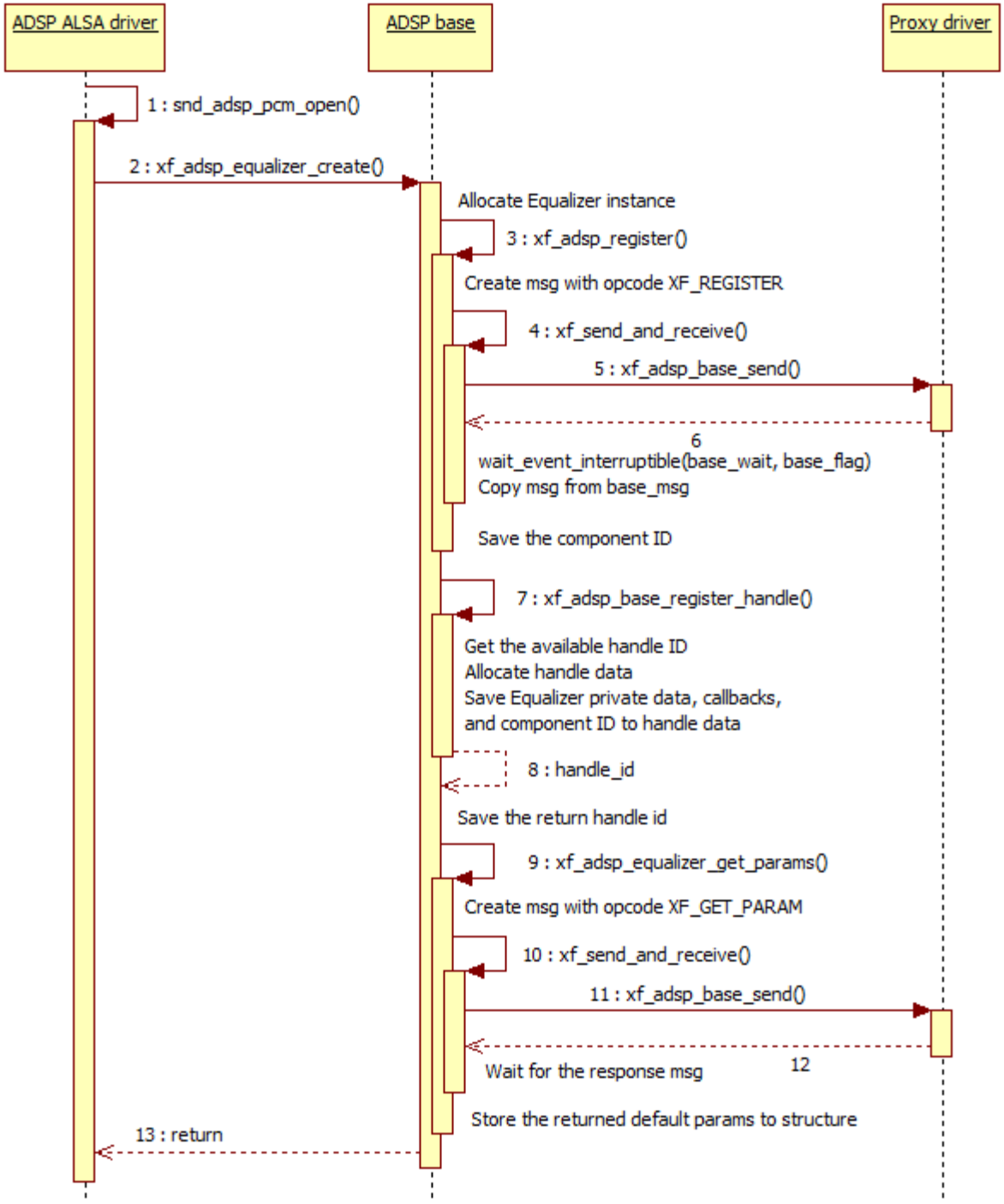


2.3.3. Capture Destruction

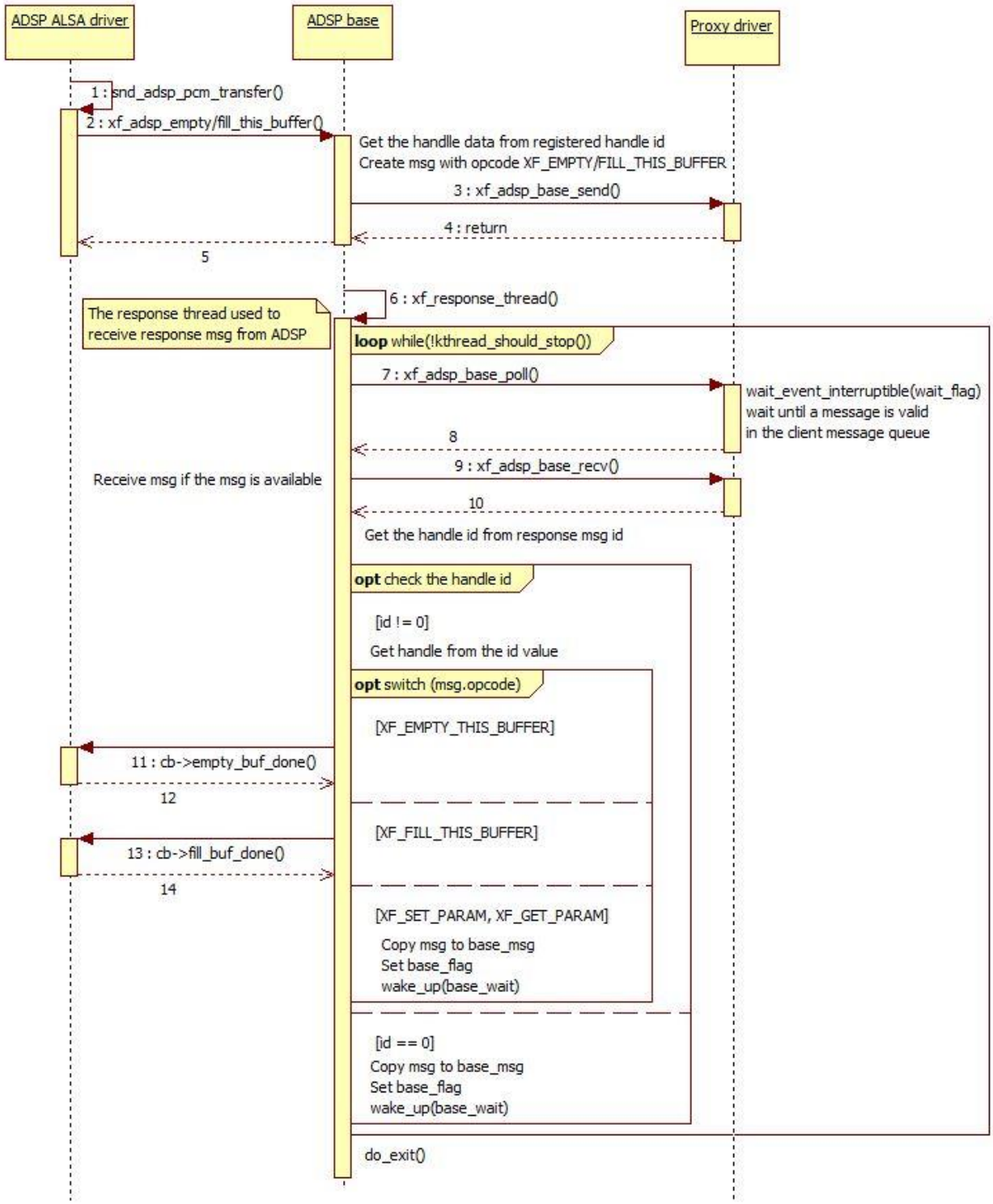


2.4. Equalizer Flow

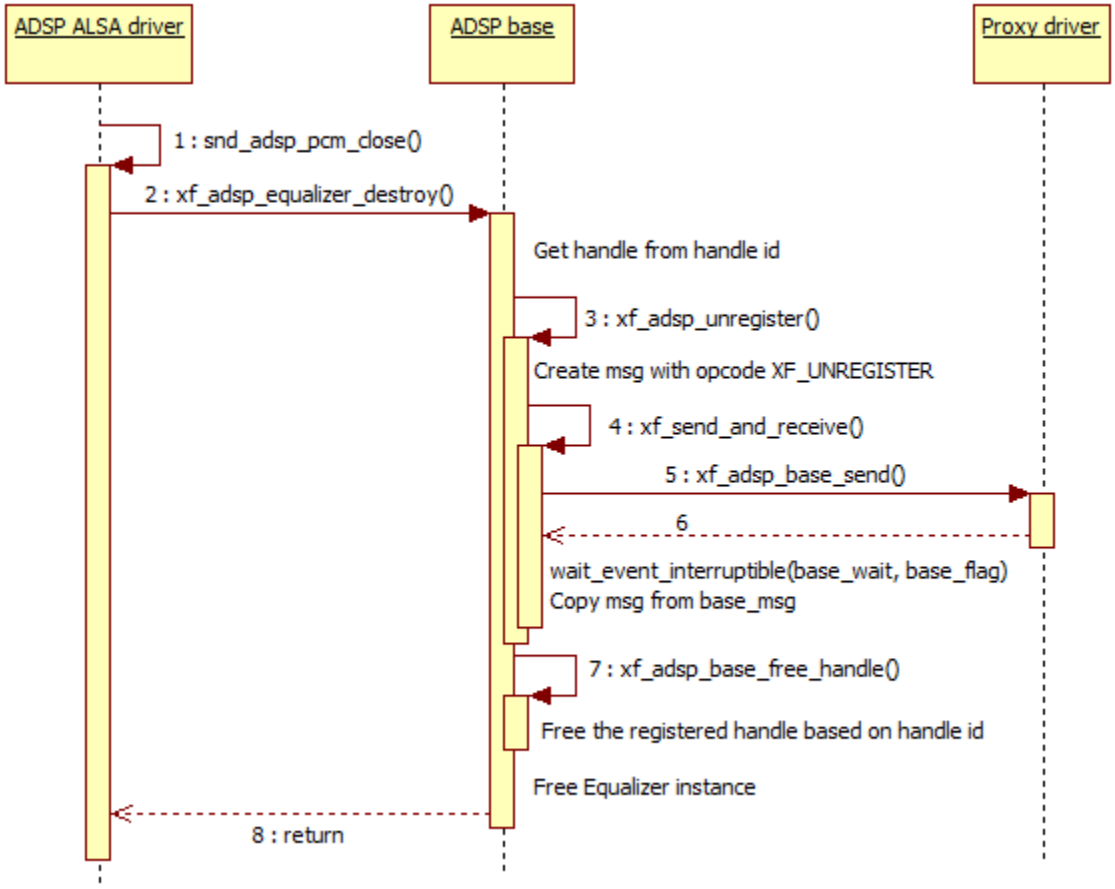
2.4.1. Equalizer Creation



2.4.2. Equalizer Execution

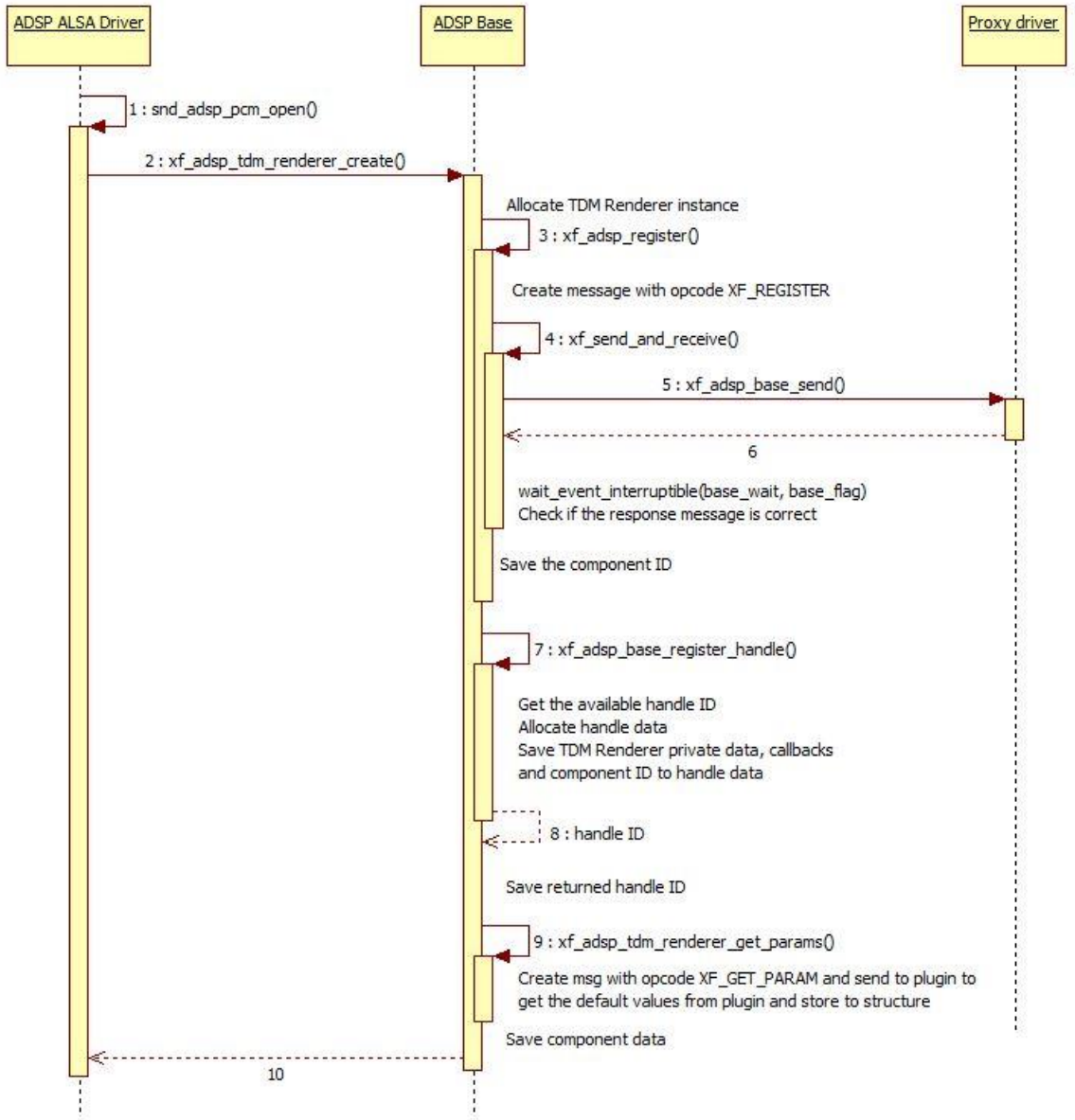


2.4.3. Equalizer Destruction

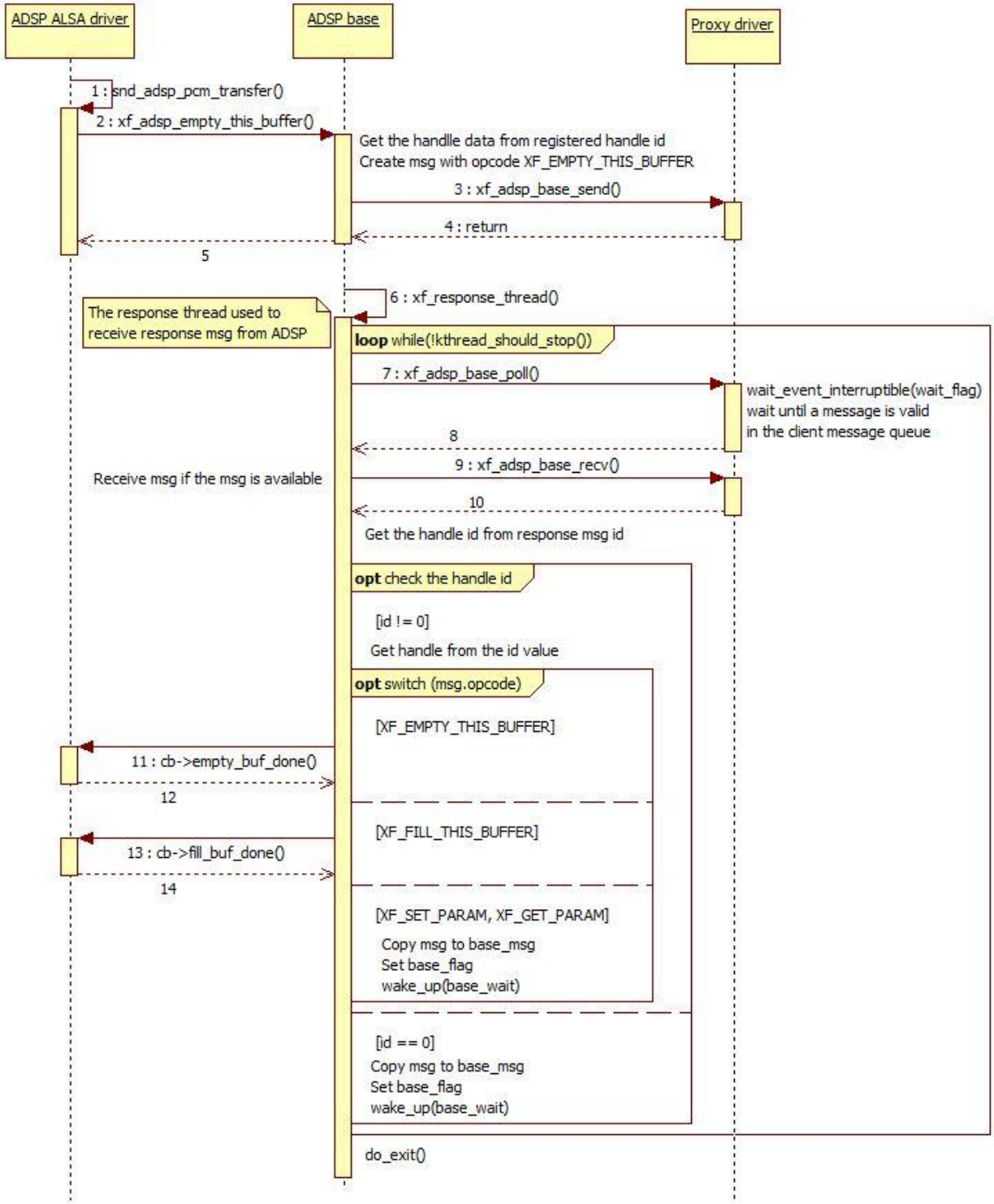


2.5. TDM Renderer Flow

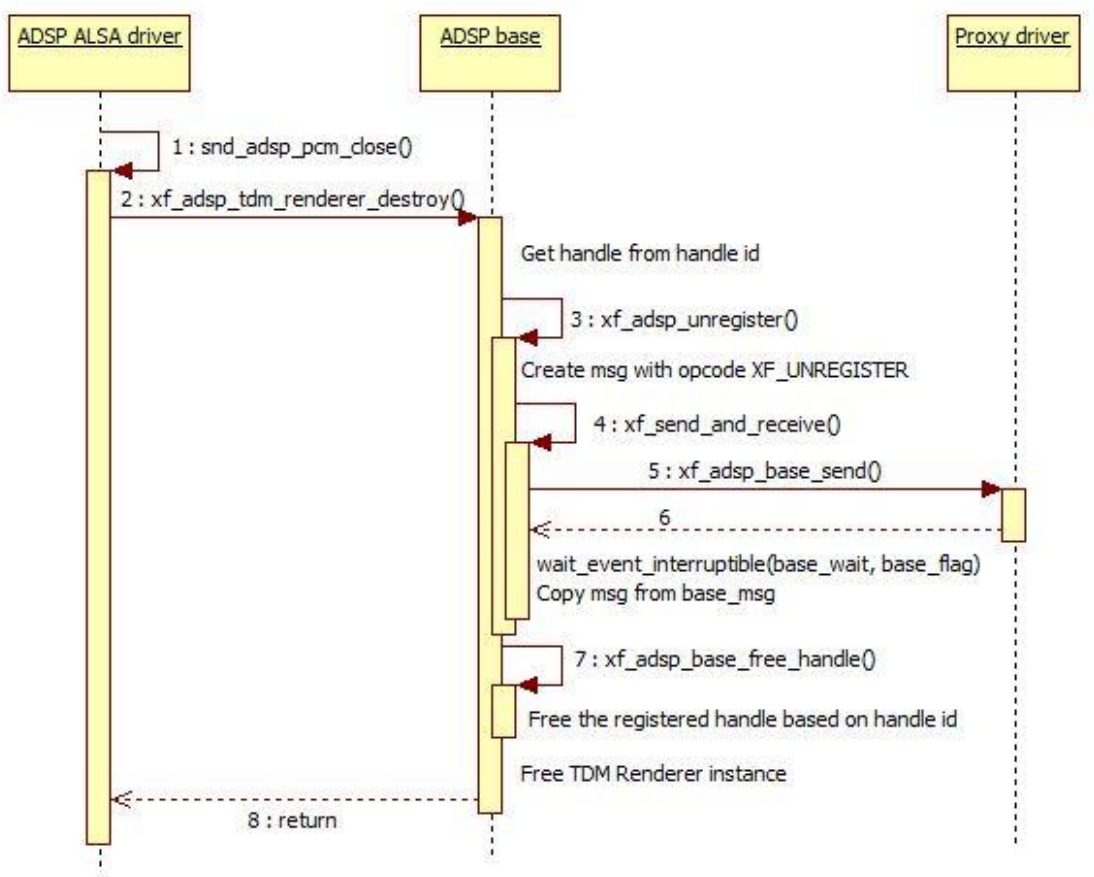
2.5.1. TDM Renderer Creation



2.5.2. TDM Renderer Execution

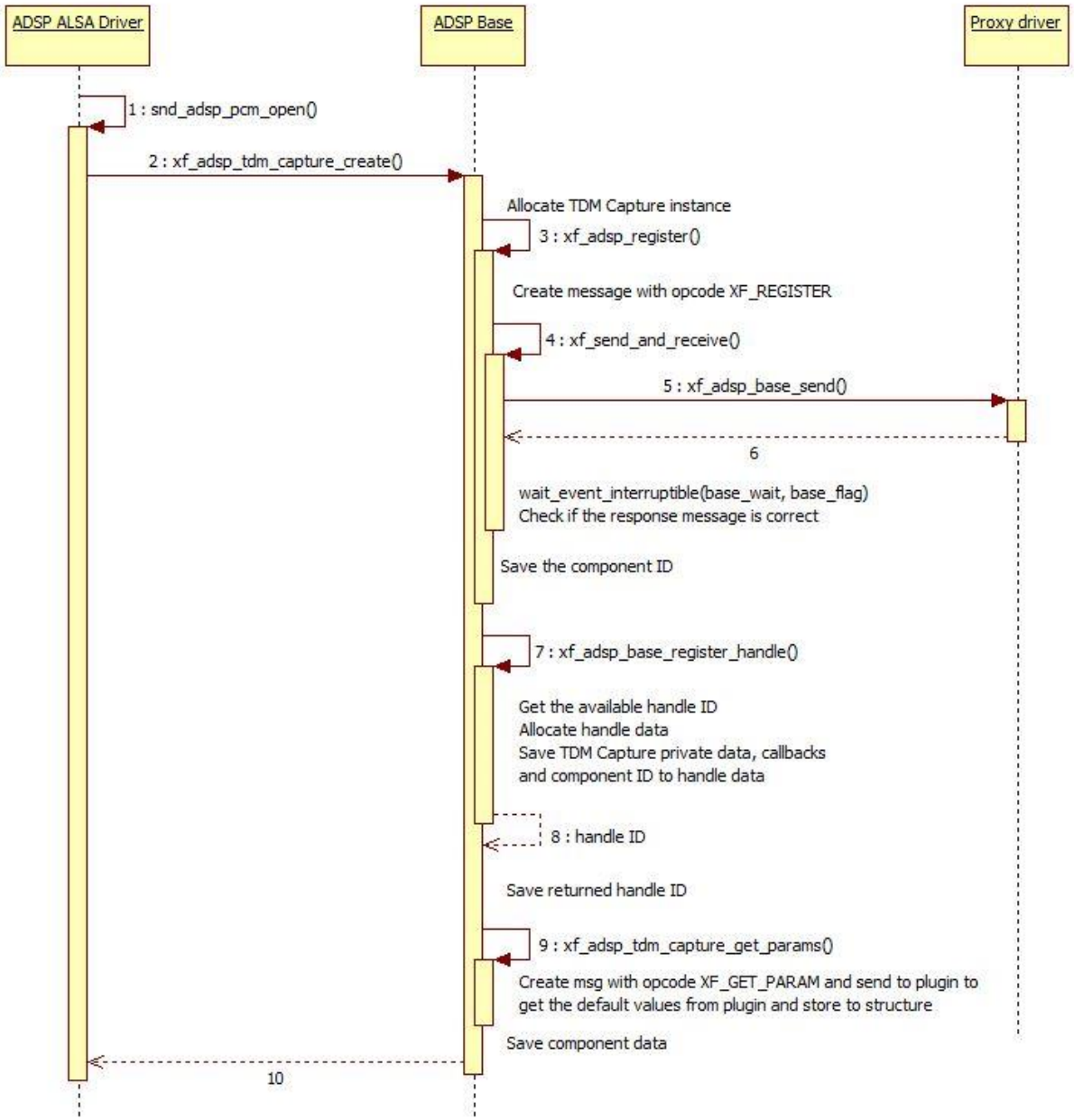


2.5.3. TDM Renderer Destruction

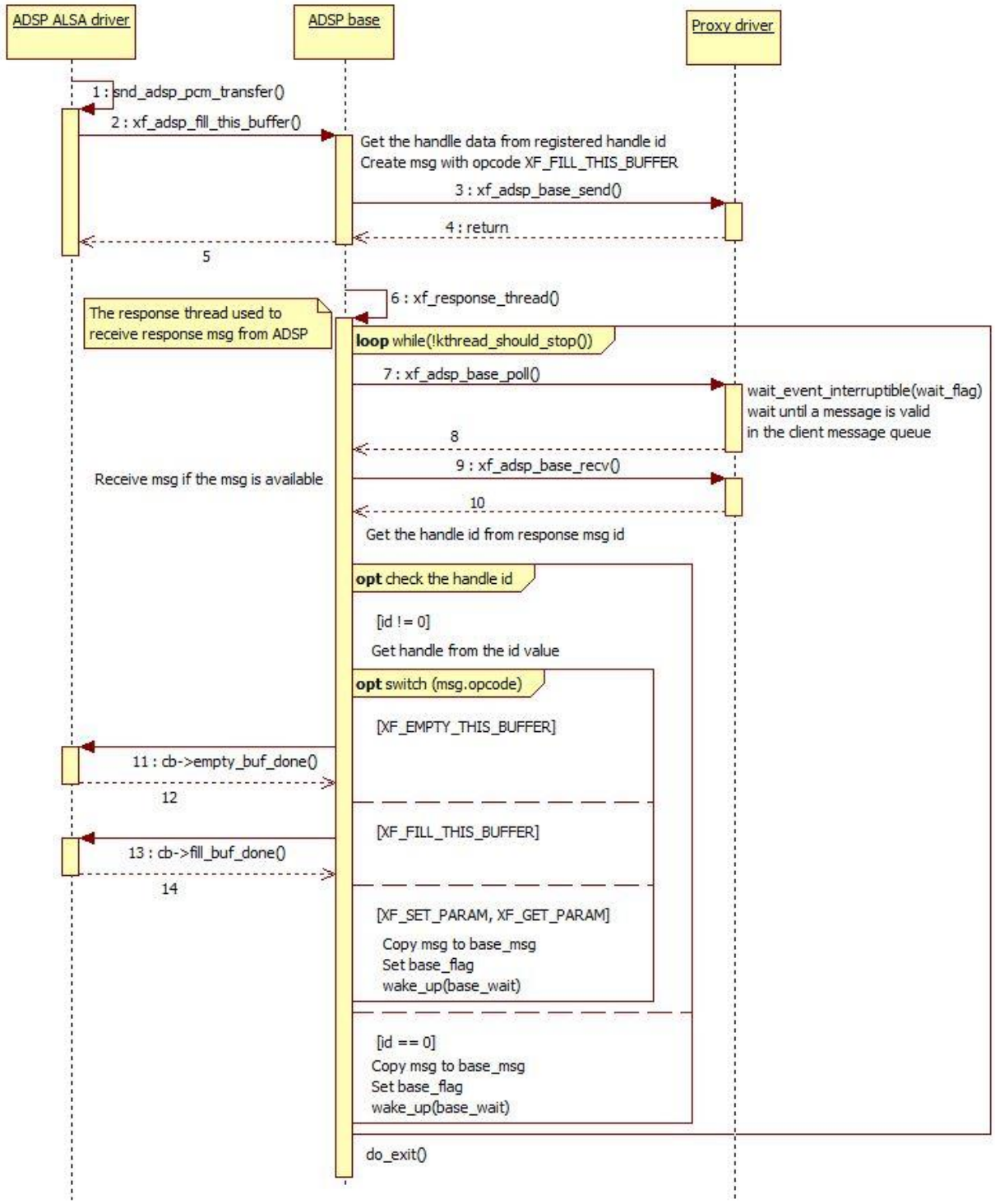


2.6. TDM Capture Flow

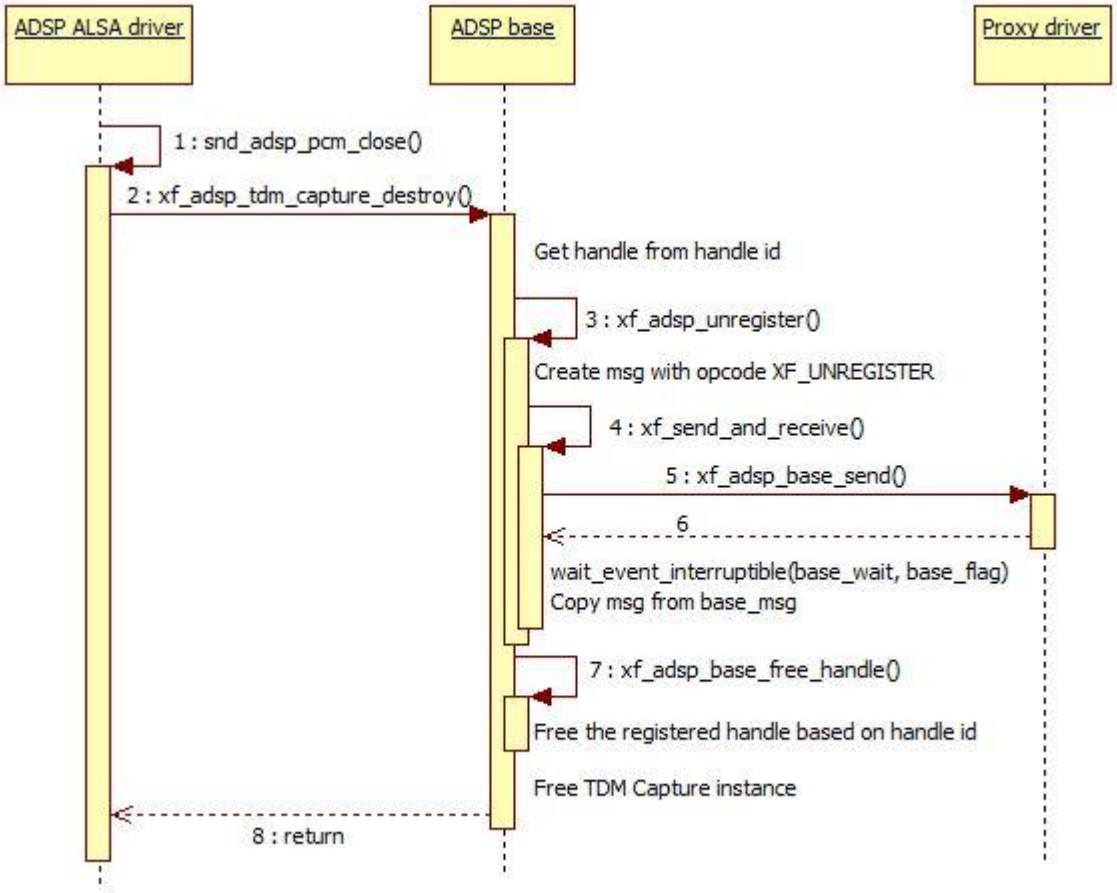
2.6.1. TDM Capture Creation



2.6.2. TDM Capture Execution



2.6.3. TDM Capture Destruction



C. ADSP Framework

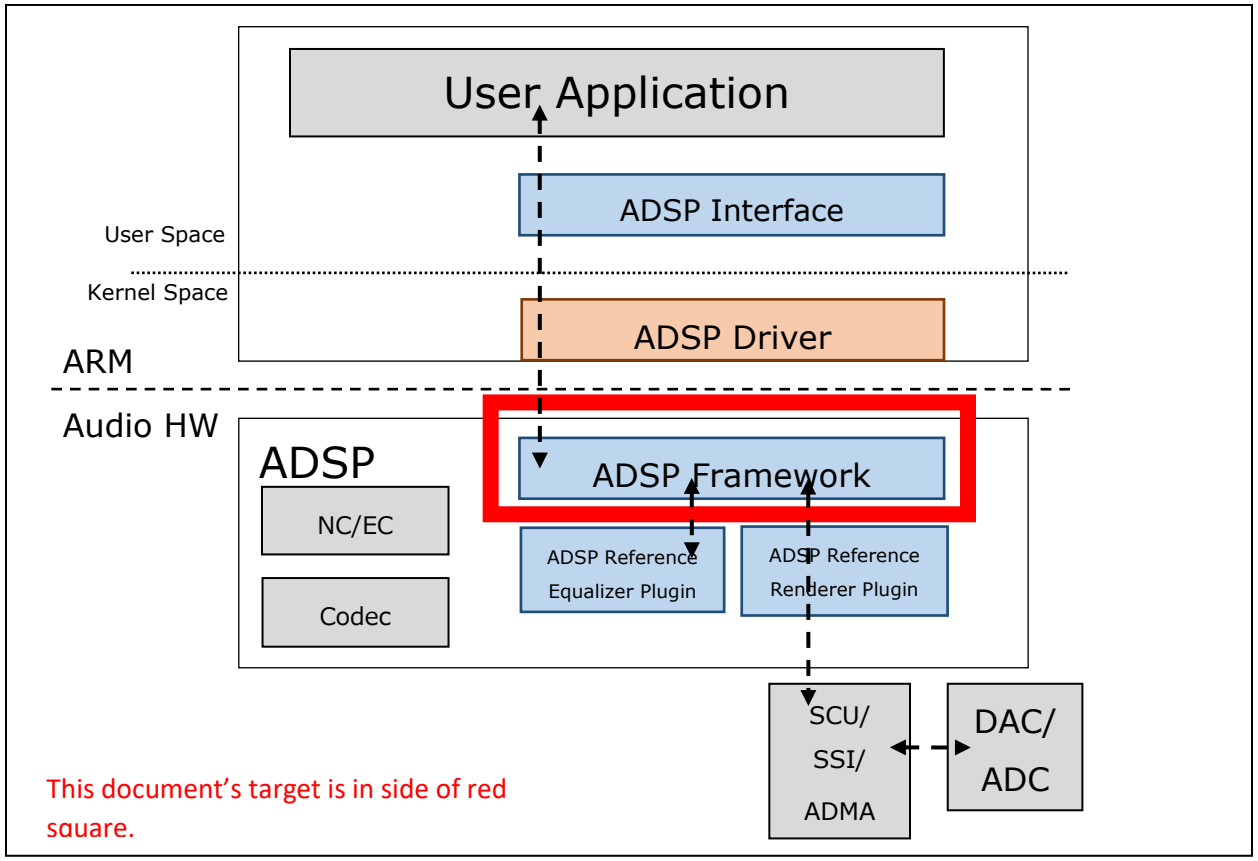
1. Overview

1.1. Overview

In this chapter, overview of ADSP Framework is explained. All expression in this manual is based on default parameters. If the software is customized from original source code, make the appropriate change according to the customization.

1.2. The architecture of the Software and scope of this document

The architecture of ADSP Framework is shown in below figure. ADSP Framework is a software which operates on ADSP and control ADSP Plugin according to messages from ADSP Interface.



1.3. Specification overview

Table 1-1. shows the specification overview of ADSP Framework.

Table 1-1. Specification overview

Item	Outline
DSP	HiFi2 by Cadence Design Systems, Inc.
IDE	Xtensa Xplorer 7.0.4 (RG-2016.4)
Configuration	hifi2_rcar_rg20164c
Memory map	adspfwk_r001c
Interrupt	call0 ABI
Compiler	Xtensa C and C++ Compiler (Version 12.0.4)
Endian	Little endian
OS	XOS by Cadence Design Systems, Inc.

[note1] This software adopts call0 ABI. Windowed ABI is not supported.

[note2] This software adopts XOS. XTOS is not supported. XTHAL can also be used.

1.4. Memory specification

Table 1- Memory specification of ADSP Framework.

Table 1-2 Memory map

Allocation	Item	Cache type	start address	size
Internal memory	I-TCM0	—	0xECE8_0000	0x0001_0000
	I-TCM1	—	0xECE9_0000	0x0001_0000
	D-TCM0	—	0xECE7_8000	0x0000_8000
	D-TCM1	—	0xECE6_0000	0x0001_0000
External memory	Debug area	Read Cache / Writeback Cache	0x5700_0000	0x0010_0000
	Code area	Read Cache / Writeback Cache	0x5710_0000	0x0030_0000
	Shared buffer area	Read Cache / Writeback Cache	0x5740_0000	0x00C0_0000

1.5. Interrupt specification

Table 1-3 Interrupt specification shows the interrupts used by ADSP Framework.

Table 1-3 Interrupt specification

No.	type	source	direction	usage
15	edge	IRQIN_SET[0]	CPU to ADSP	send command
—	Level	IRQOUT_SET	ADSP to CPU	send response
0	Level	Output FIFO 0	FIFO→ADSP	Output FIFO 0 Interrupt
1	Level	Input FIFO 0	FIFO→ADSP	Input FIFO 0 Interrupt
2	Level	Output FIFO 1	FIFO→ADSP	Output FIFO 1 Interrupt
3	Level	Input FIFO 1	FIFO→ADSP	Input FIFO 1 Interrupt
4	Level	Output FIFO 2	FIFO→ADSP	Output FIFO2 Interrupt
5	Level	Input FIFO 2	FIFO→ADSP	Input FIFO 2 Interrupt
8	Level	Output FIFO 3	FIFO→ADSP	Output FIFO 3 Interrupt
9	Level	Input FIFO 3	FIFO→ADSP	Input FIFO3 Interrupt

1.6. Related documents

Table shows related documents and references.

Table 1-4 Related documents

No.	Name	Published by
[1]	R-Car Series, 3rd Generation User's Manual: Hardware	Renesas Electronics Corporation
[2]	Xtensa® LX4 Microprocessor Data Book	Tensilica, Inc.
[3]	Xtensa® XOS Reference Manual	Cadence Design Systems, Inc.
[4]	Xtensa® System Software Reference Manual	Cadence Design Systems, Inc.

1.7. Type definitions

Table shows the type definitions used by ADSP Framework.

Table 1-5 Type definitions

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

[note] The size of pointer depends on the architecture this software is used.

2. Software specification

2.1. State transition

Figure 2.1 shows state transition of each ADSP Plugin. See 0, 0 or 0 for the flows in each stage.

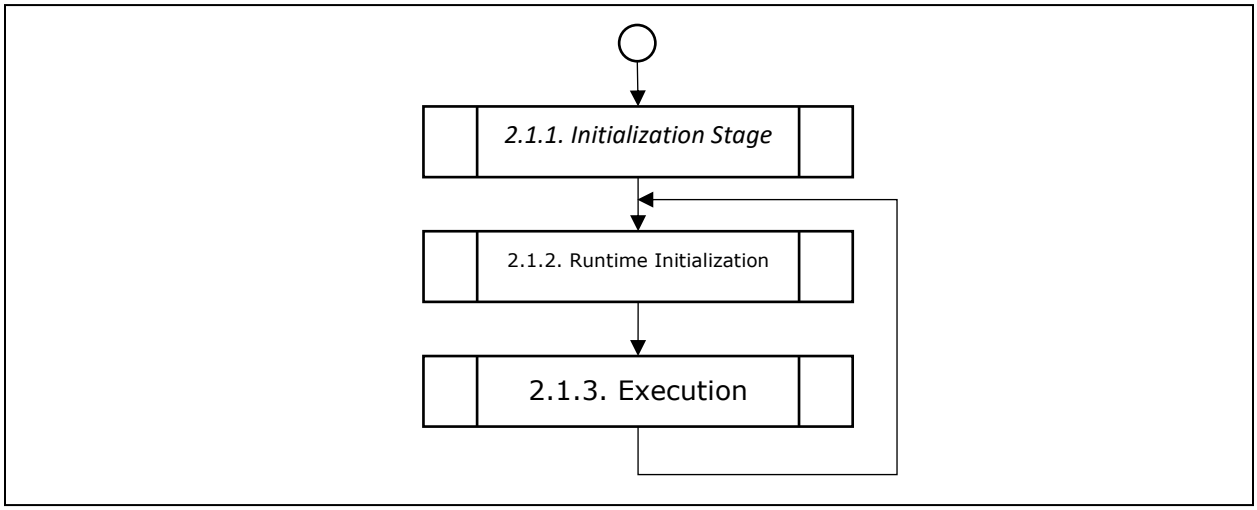


Figure 2.1 ADSP Plugin state transition

2.1.1. Initialization Stage

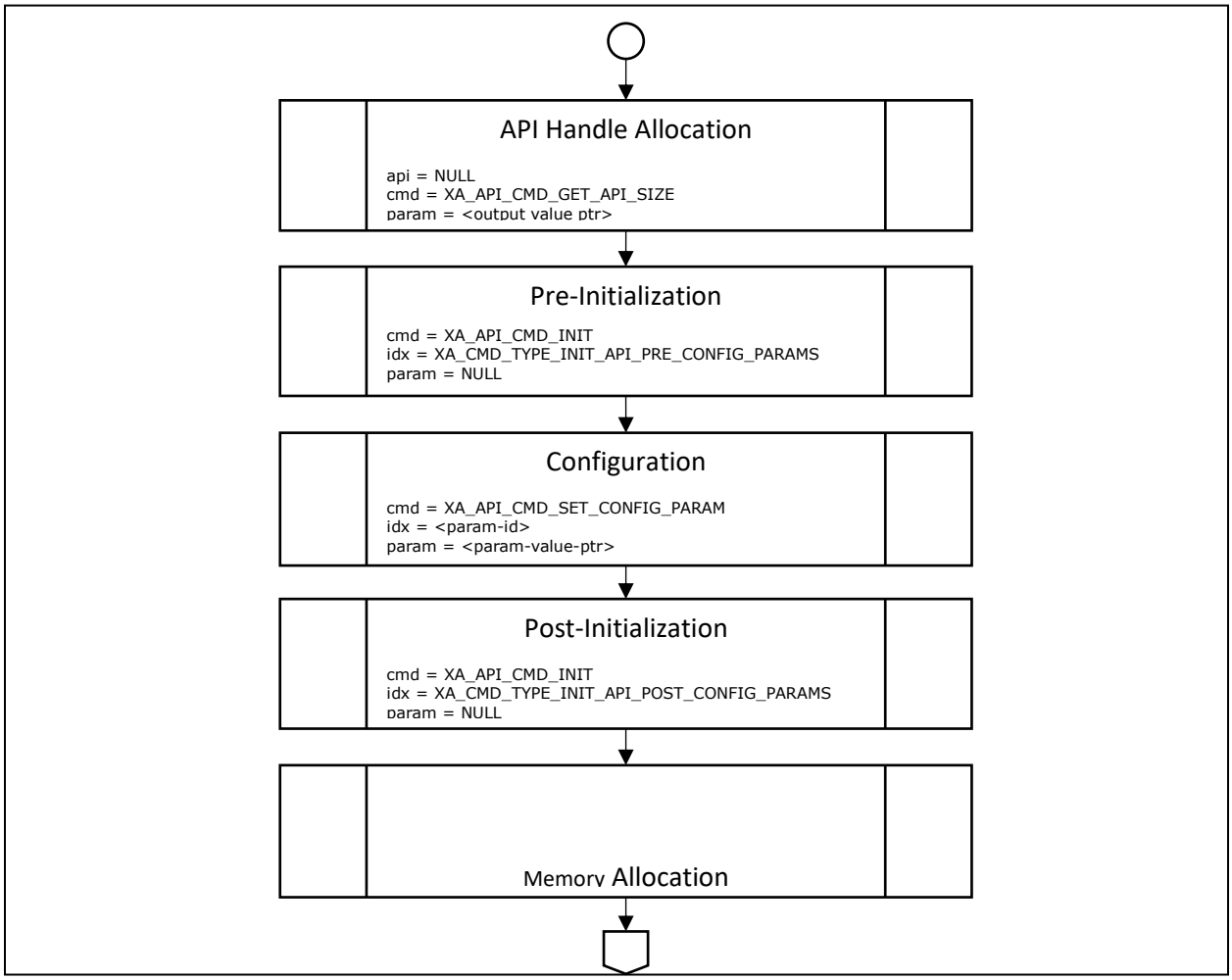


Figure *2.1.1. Initialization Stage* flow chart

2.1.1. Initialization Stage performs allocation of resources for ADSP Plugin and configures static parameters. This stage consists of the following steps.

(1) API Handle Allocation
It is the start of ADSP Plugin lifecycle. It requires the size of ADSP Plugin API structure.
(2) Pre-Initialization
It is the entry point for ADSP Plugin. It initializes the API structure allocated by ADSP Framework and configures the default value.
(3) Configuration
It configures the static parameters of ADSP Plugin.
(4) Post-Initialization
It means the configure of the static parameters is finished. It fixes persistent area, scratch area and input buffer area. The size of each buffer can’t be increased after this step.

(5) Memory Allocation
ADSP Framework allocates ADSP Plugin buffer and registers it to ADSP Plugin. It is possible to exchange data with ADSP Plugin after this step.

2.1.2. Runtime Initialization Stage

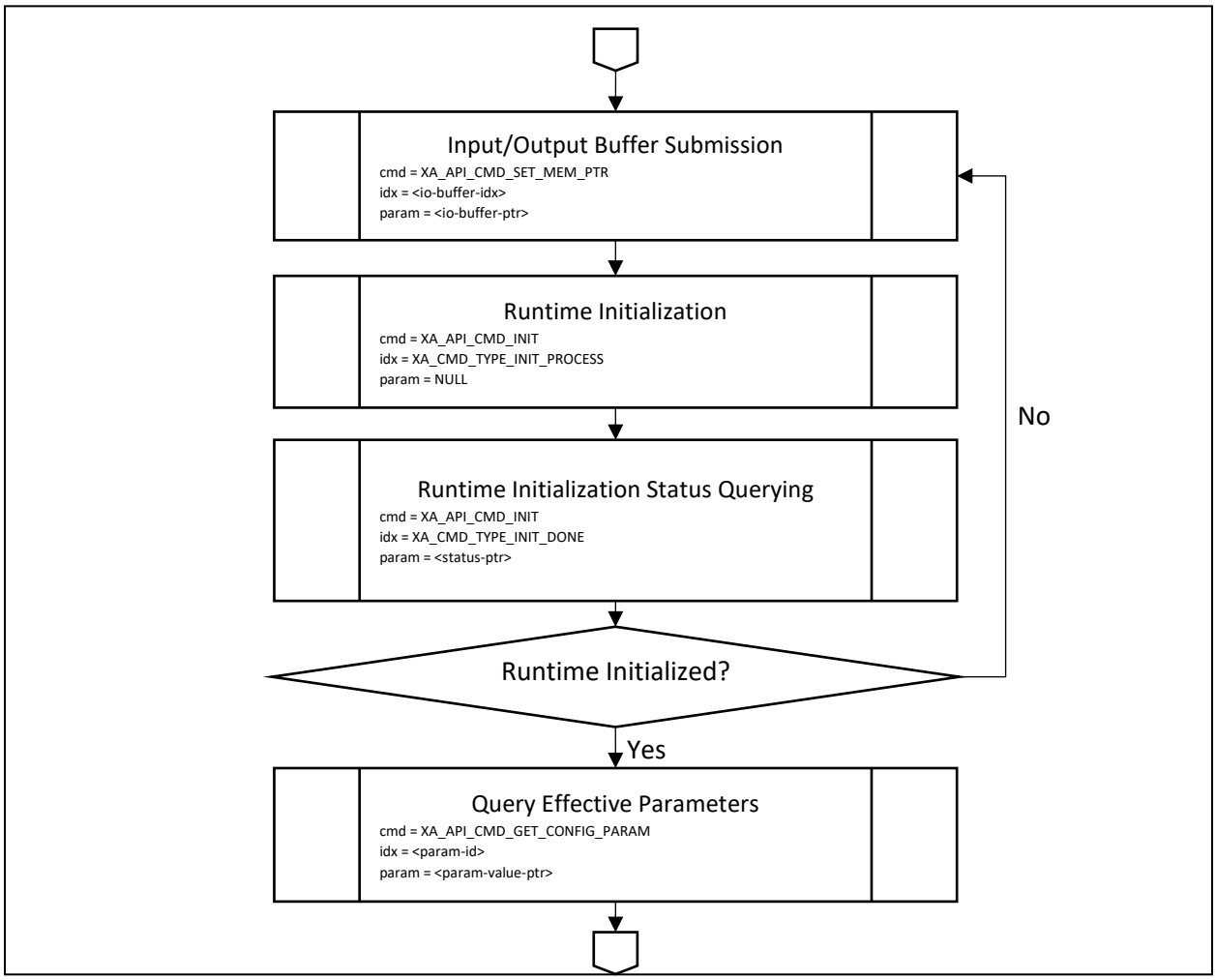


Figure *2.1.2. Runtime Initialization Stage* flow chart

2.1.2. Runtime Initialization Stage performs configuration of parameter which depends on input data and so on. It checks input data but never make output data.

2.1.3. Execution Stage

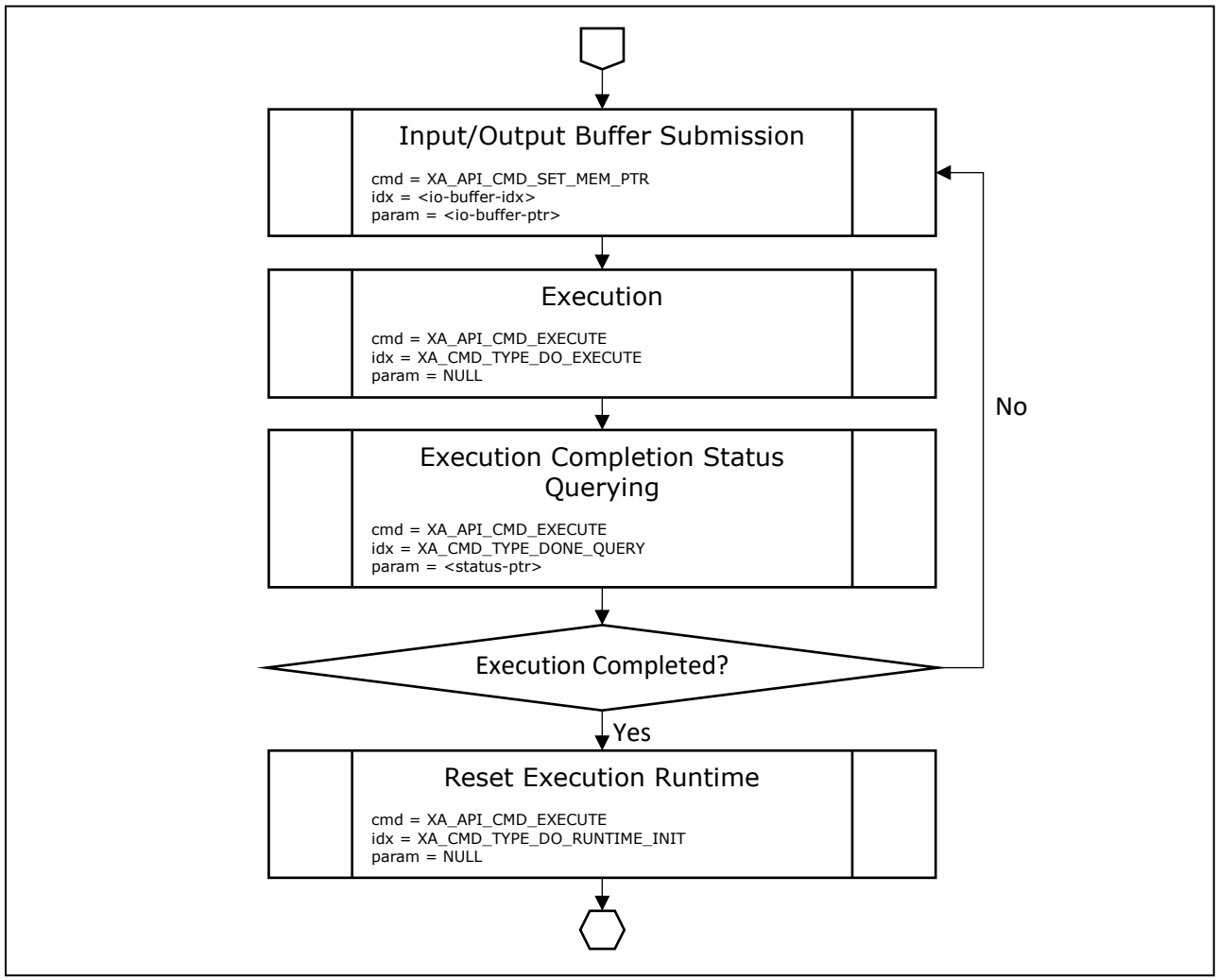


Figure *2.1.3. Execution Stage* flow chart

2.1.3. Execution Stage performs ADSP Plugin processing. The way to process input and output data in ADSP Plugin should be determined at the beginning of this stage.

ADSP Framework usually processes input and output data successively except when a physical cause inputs or outputs data e.g. Renderer plugin and Capturer plugin.

When ADSP Framework receives notification that input data is, it provides all input data, output all output data, and then finishes this stage. After this stage is finished, it is necessary to shift the state to *2.1.2. Runtime Initialization Stage* to run ADSP Plugin process.

It is possible to shift the state to *2.1.2. Runtime Initialization Stage* at any time in this stage.

D. ADSP Plugin

1. ADSP Renderer/Capture plugins

1.1. Overview

This section provides an overview of the Renderer plugin. It contains renderer and capture function.

1.1.1. Specifications Outline

Renderer function plays the audio signal based on the parameter that was set.

Capture function capture/record the audio signal based on the parameter that was set.

Table 1.1.1. Basic Specification

Item	Description
DSP	Cadence Design Systems, Inc. HiFi2
Compiler	Xtensa C and C++ Compiler (version 12.0.4)
Endian	Little Endian

Table 1.1.2. Supported Specifications for capture function

Item	Description
Input data format	16-bit/24-bit linear PCM (fixed point)
Output data format	16-bit/24-bit linear PCM (fixed point)
Sampling frequency (Hz) supported	48000 / 44100 / 32000
Number of channels supported	Max 2 channels
Reentrant	Supported
Volume control	Support volume update during plugin execution
Other	-
Restrictions	-

Table 1.1.3. Supported Specifications for renderer function

Item	Description
Input data format	16-bit/24-bit linear PCM (fixed point)
Output data format	16-bit/24-bit linear PCM (fixed point)
Sampling frequency (Hz) supported	48000 / 44100 / 32000
Input data channel	Monaural, stereo, 4 channels, 6 channels, 8 channels
Output data channel	Max 2 channels
Reentrant	Supported
Volume control	Support volume update during plugin execution.
Other	-
Restrictions	-

[Note] In case of channel transfer from input monaural to output stereo, the sound will output to both two channels.

Table 1.1.4 Memory Size Requirements

Memory type	Location	Memory area name		Size (in bytes)	
Instruction	ROM	Instruction area			78393
Data		Constant table area			
		Other area(Depended on the compiler)			
	RAM Capture	Software work area			1756
		Area breakdown	Persistent area	Size breakdown	1692
			Scratch area		0
			Built-in descriptor area		64
		User work area			9576
		Area breakdown	Input buffer	Size breakdown	0
			Output buffer		8192
			Structure		1384
		Stack area			1168
		Other area(Depended on the compiler)			0
	RAM Renderer	Software work area			1748
		Area breakdown	Persistent area	Size breakdown	1684
			Scratch area		0
			Built-in descriptor area		64
		User work area			9616
		Area breakdown	Input buffer	Size breakdown	8192
			Output buffer		0
			Structure		1424
		Stack area			1440
		Other area(Depended on the compiler)			0

[Note] Area whose location is shown as ROM in the location column can be included in RAM or ROM.
[Note] Area whose location is shown as RAM in the location column can be included in RAM only.
[Note] Built-in is a memory area to allocate descriptor memory, which need in the DMAC transfer type of plugin.

Table 1.1.5 Version Information

Item	Description
Library Version information	Version 1.0.4
API Version information	Version 1.0.0

1.1.2. Configuration

Figure 1.1.2.1 shows an example of the ADSP system configuration which uses renderer function.

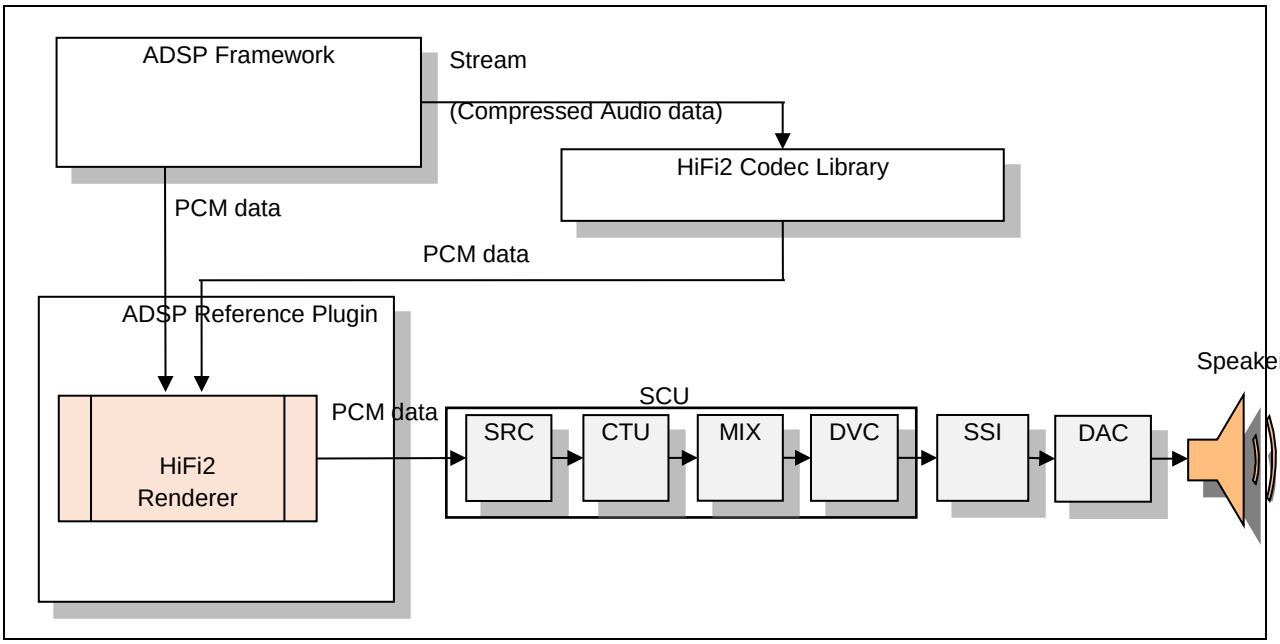


Figure 1.1.2.1 Example of the ADSP System Configuration for renderer function

Figure 1.1.2.2 show an example of using channel converter and mixer function to mix 4 streams from 4 Renderer plugins.

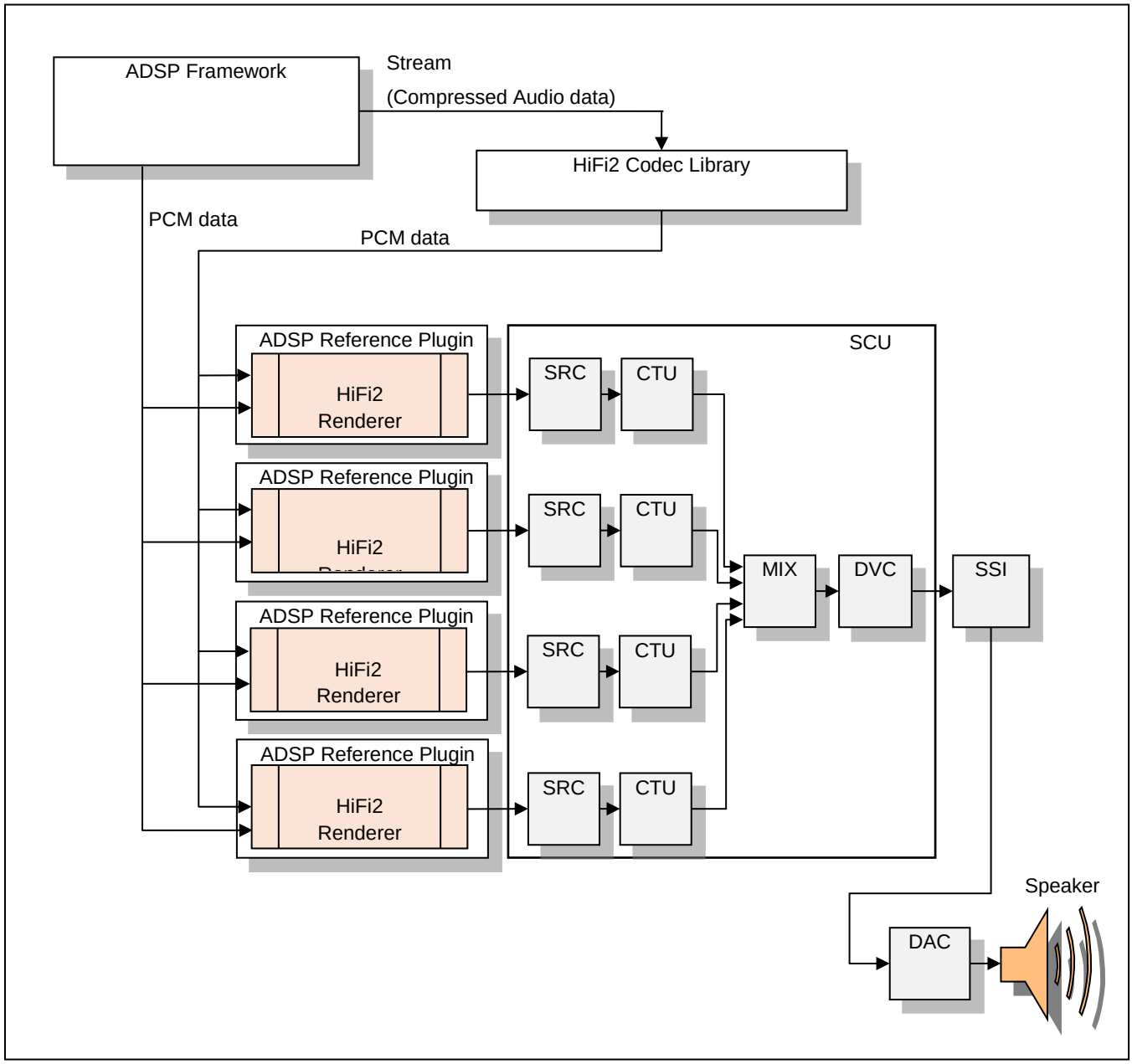


Figure 1.1.2.2 Example of the ADSP System Configuration uses CTU/MIX for renderer function

Figure 1.1.2.3 shows an example of the ADSP system configuration which uses capture function.

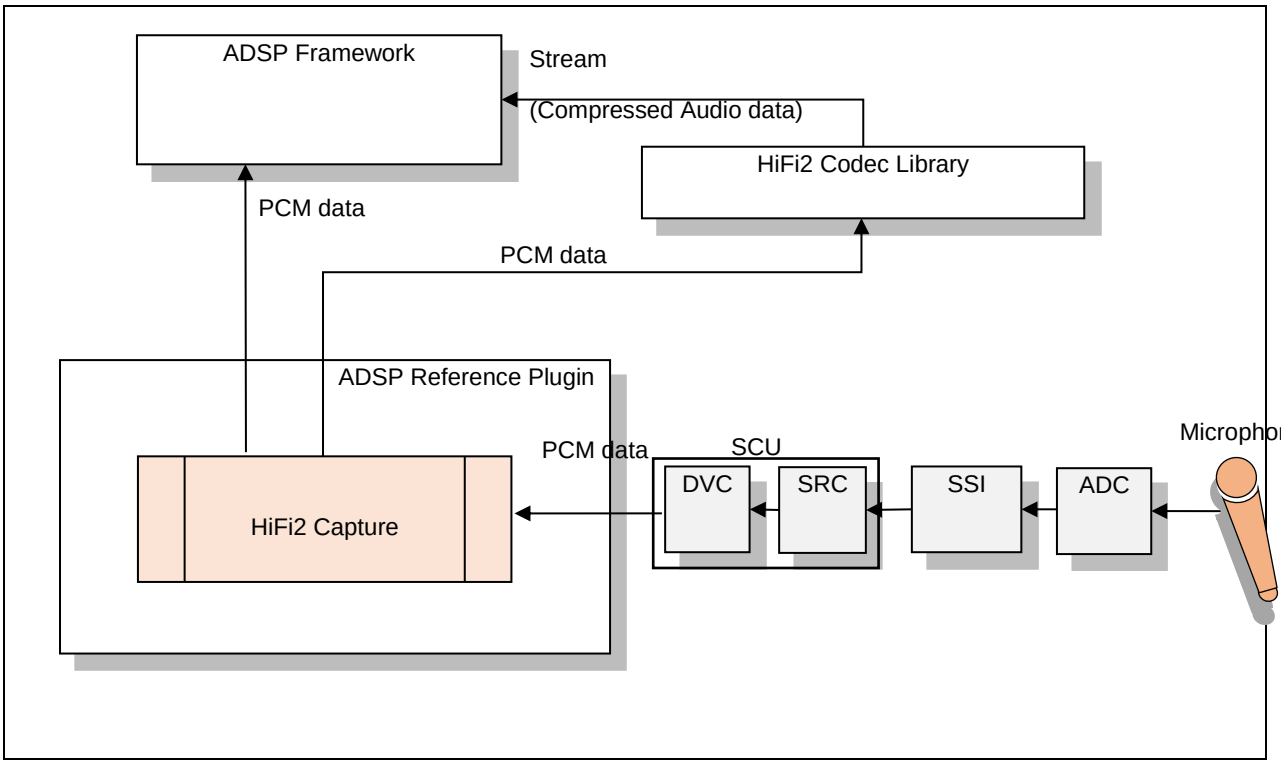


Figure 1.1.2.3 Example of the ADSP System Configuration for capture function

Stream (Compressed Audio data)

Compressed Audio data is a linear PCM data sample compressed according to the compression Audio specifications. For these specifications, depends on HiFi2 Codec Library.

HiFi2 Codec Library

It is Codec Library for HiFi2. It decodes the compression Audio in renderer case and encodes in capture case. The user should procure to suit the target system.

ADSP Framework

It controls ADSP Plugin. It is software provided separately as Framework.

HiFi2 Renderer (ADSP Reference Plugin)

It performs output handling of PCM data to other Audio device. It is this software set up as ADSP Reference Plugin.

HiFi2 Capture (ADSP Reference Plugin)

It performs input handling of PCM data from other Audio device. It is this software set up as ADSP Reference Plugin.

PCM data

16-bit/24-bit linear PCM data which is a processing by this software.

SCU

It performs sampling rate converters (SRC), channel transfer (CTU), mixing (MIX), and volume control (DVC).

SRC

It performs sampling rate conversion function.

CTU

It performs channel transfer unit function such as down-mixing and splitter functions.

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.

DVC

It performs mute and volume control functions.

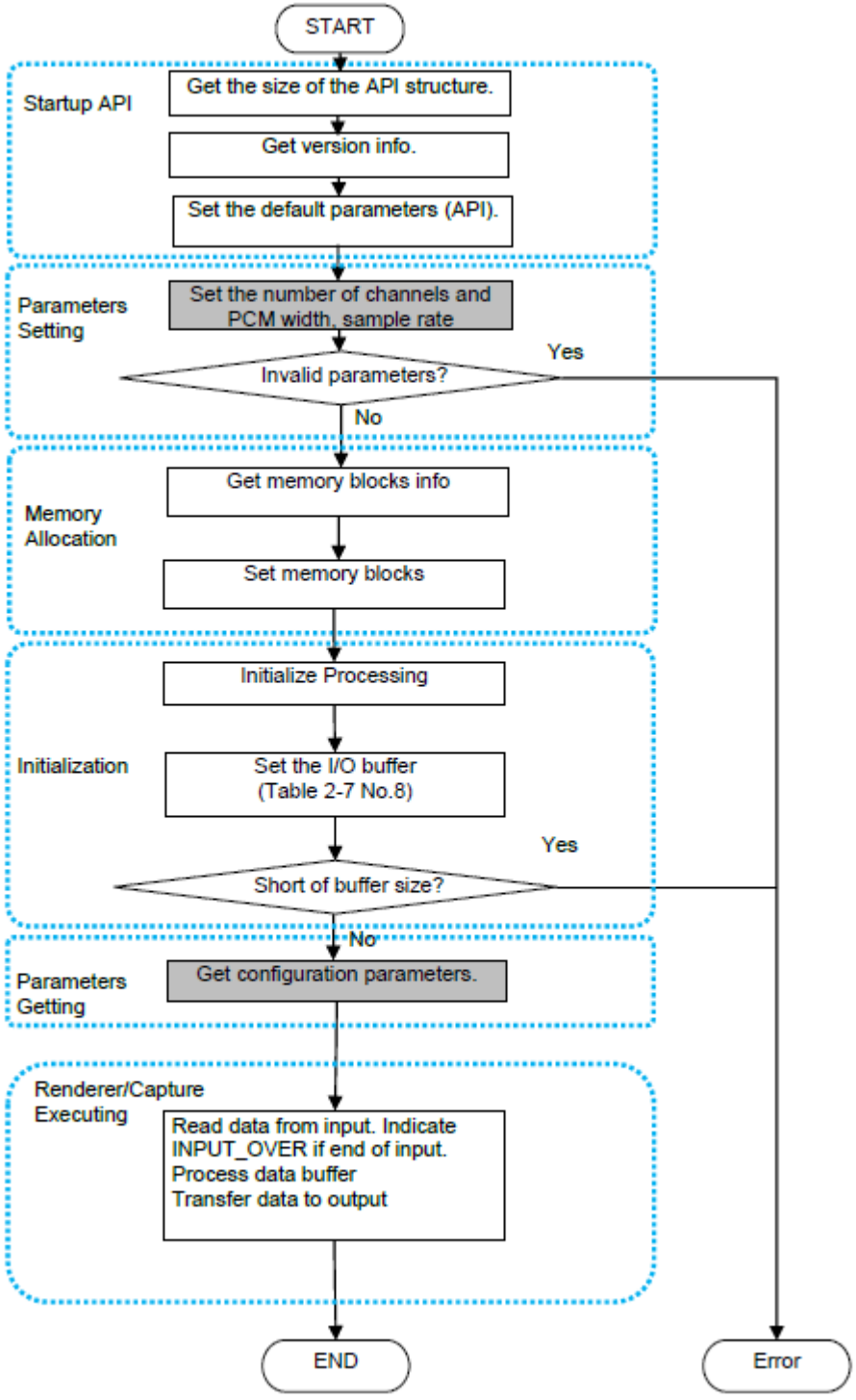
SSI

Send or receive audio data interfacing with a variety devices of offering I2C format.

ADC

The ADC converts an analog signal into 16-bit linear PCM data.

1.2.Processing flow



1.3. Apendix

This section will explain more detail about the configuration to using CTU, MIX module to perform channel transfer and mixing functions.

1.2.1. CTU (Channel transfer unit)

CTU implements the channel transfer unit function. It converts PCM format from “x” channel (for input PCM of Renderer plugin) to “y” channel (expected output channel for speaker).

(x = monaural, stereo, 4 channels, 6 channels, 8 channels; y = monaural, stereo)

The setting of input, output channel can be referred in XA_API_CMD_SET_CONFIG_PARAM

To use CTU functions, the input channel’s and output channel’s values must be different, if not, it implies that CTU module is not used, except when Renderer is using MIX function. Because CTU and MIX are electrically connected as figure 1.2.1.1

There are two discrete CTU modules available. And each module includes four sub-modules. The output of these sub-modules internally connects a MIX module (see Figure 1.2.1.1).

So, if you play two PCMs (from 2 Renderers) and mix them together, two sub-modules of CTU0 will be enabled, while CTU1 module is still available. But if you use CTU function in playing two PCMs (from 2 Renderers) separately to two different outputs, then both CTU0 and CTU1 are enabled. Note that, the index of CTU will be selected inside Renderer plugins automatically.

Figure 1.2.1.1 shows all available of CTU modules.

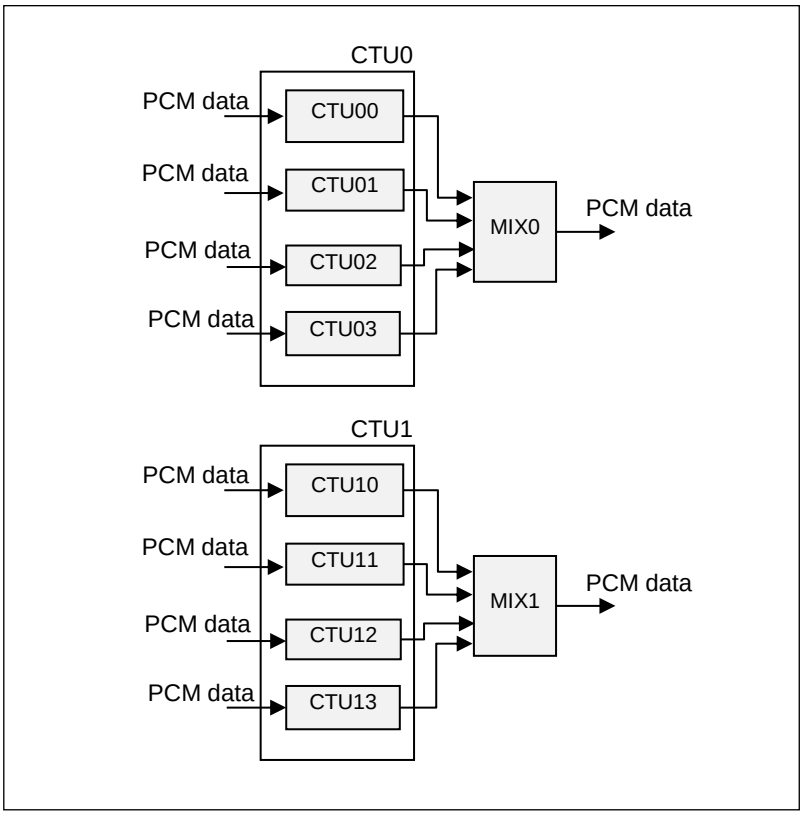


Figure 1.2.1.1 Block diagram of CTU

1.2.2. MIX (Mixing)

MIX implements the mixing (adding) two to four streams from Renderer plugins into a single stream, which will be output to a speaker.

The setting of plugin to use MIX functions can be referred to XA_API_CMD_SET_CONFIG_PARAM command.

As figure 1.2.1.1, there are two MIX modules available. And each one supports maximum of 4 inputs.

The below conditions must be assured when mixing PCMs output from Renderers.

(1) Set mix control flag
- Plugin has to set the mix control flag to assure that this stream want to mix with others.

(2) Same output destination
- All plugins have to set the same output SSI module index. Otherwise, the plugin will play independently as PCM 3 in figure 1.2.2.1.

(3) Input mixing available
- A Mix module supports maximum of 4 streams. So when the 5th stream performs mixing (condition (2)), the busy error of hardware will be returned.

(4) Consistent PCMs format
- The output PCM format (output channels, output sampling rate, and PCM width) has to have same configurations. Otherwise, this plugin will return an error and stop immediately.

Below are figures that present some use cases in using CTU/MIX modules in ADSP Renderer plugin.

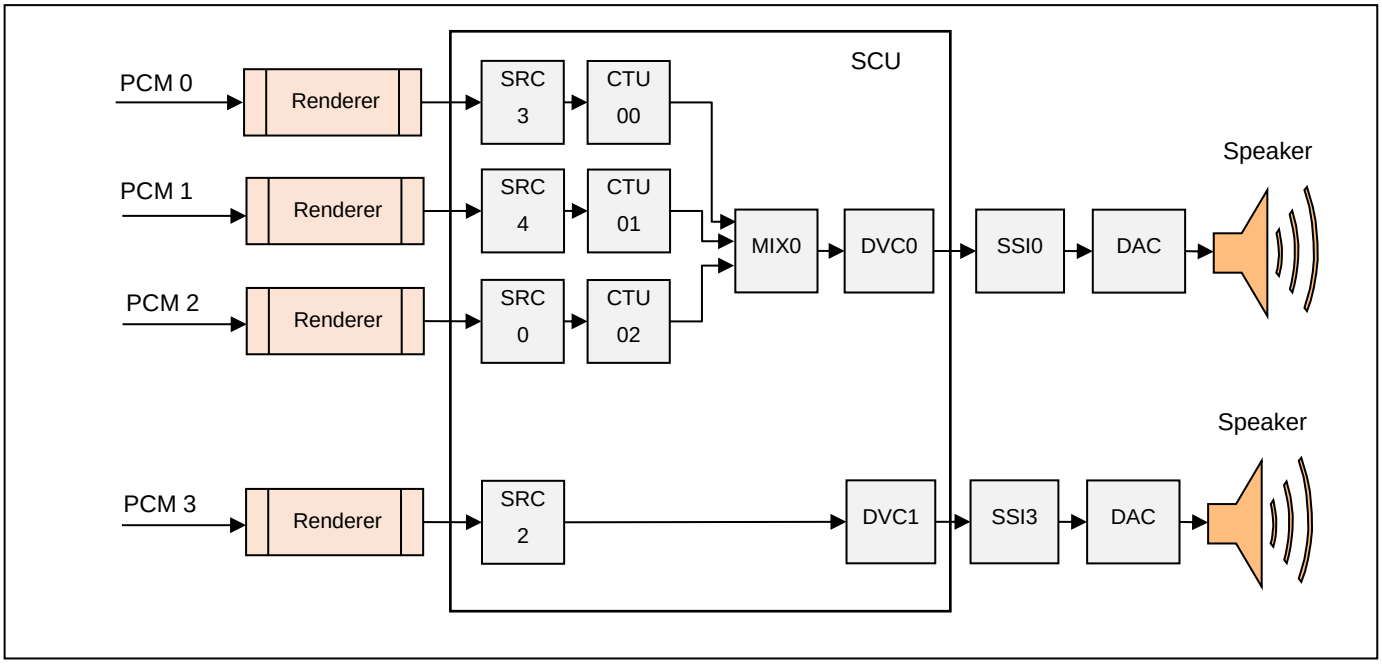


Figure 1.2.2.1. Example of using CTU/MIX for 3 streams to SSI0 and another stream to SSI3

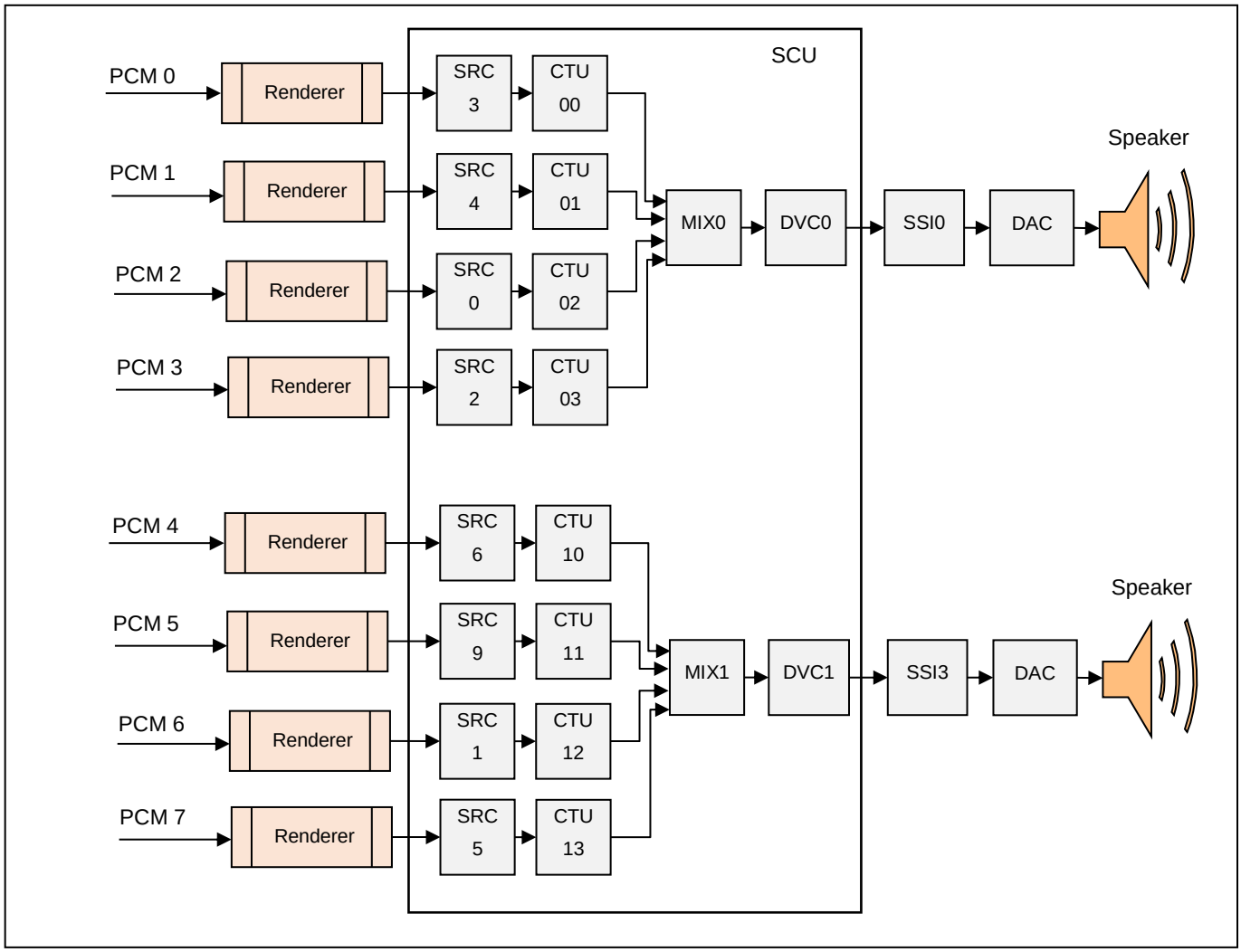


Figure 1.2.1.3 Example of using CTU/MIX to play 8 streams to SSI0 and SSI3.

About the volume control feature, MIX supports to control the decibel (gain level) of each input stream. From that, the output volume will be controlled by both DVC and MIX (Figure). On the other hand, by setting MIX, user can set different volume rates for four Renderer plugins when mixing. The output sound is a combination of the inputs and their expected volumes.

1.2.3. 16-bit monaural

Figure 1.2.3.1 Data format of 16-bit monaural in SW and HW

For 16-bit monaural case, it is a difference in data size that it is in SW and HW. Therefore, it is impossible to convert between SW and HW. Then, only FIFO is used for the 16-bit monaural case. If the 1st device is assigned ADMAC to transfer data, the plugin itself automatically changes it to ADMACpp to use FIFO for data transfer.

1.2.4. Conversion functions

If the conversion function is used:

- The volume is set.
- Input sampling rate and output sampling rate are different.
- Input channel and output channel are different, or MIX function is enabled (only in Renderer case).

But XA_RDR_CONFIG_PARAM_OUTPUT2, or XA_CAP_CONFIG_PARAM_INPUT2 are not set by user. Renderer/Capture plugins will automatically enable SRC module and using ADMAC transfer type to transfer data between the plugin and Audio HWs.

1.2.5. State transformation

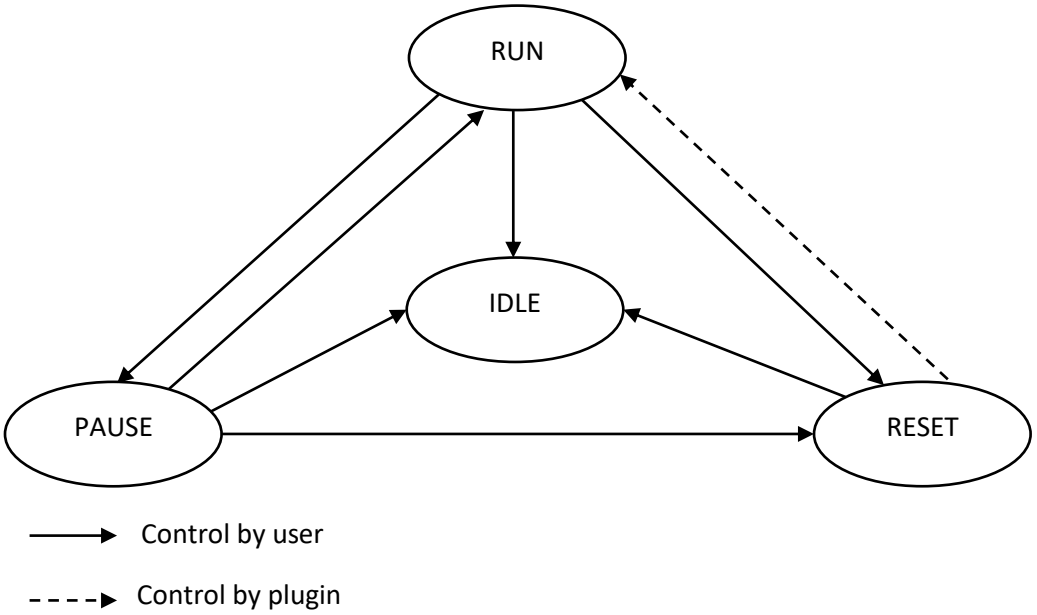


Figure 1.2.5.1 State transformation in the plugin

Figure above shows how state is transformed in the plugin. When the current state is RESET, the plugin itself changes the state internally to RUN in the next execution.

2. Equalizer plugin

2.1. Overview

2.1.1. Overview of this document.

In this chapter, overview of ADSP Equalizer plugin is explained.

2.1.2. The architecture of the Software and scope of this document

The architecture of ADSP Equalizer is shown in Figure 2.1.2.1. ADSP Equalizer is an ADSP plugin which is controlled by ADSP Framework.

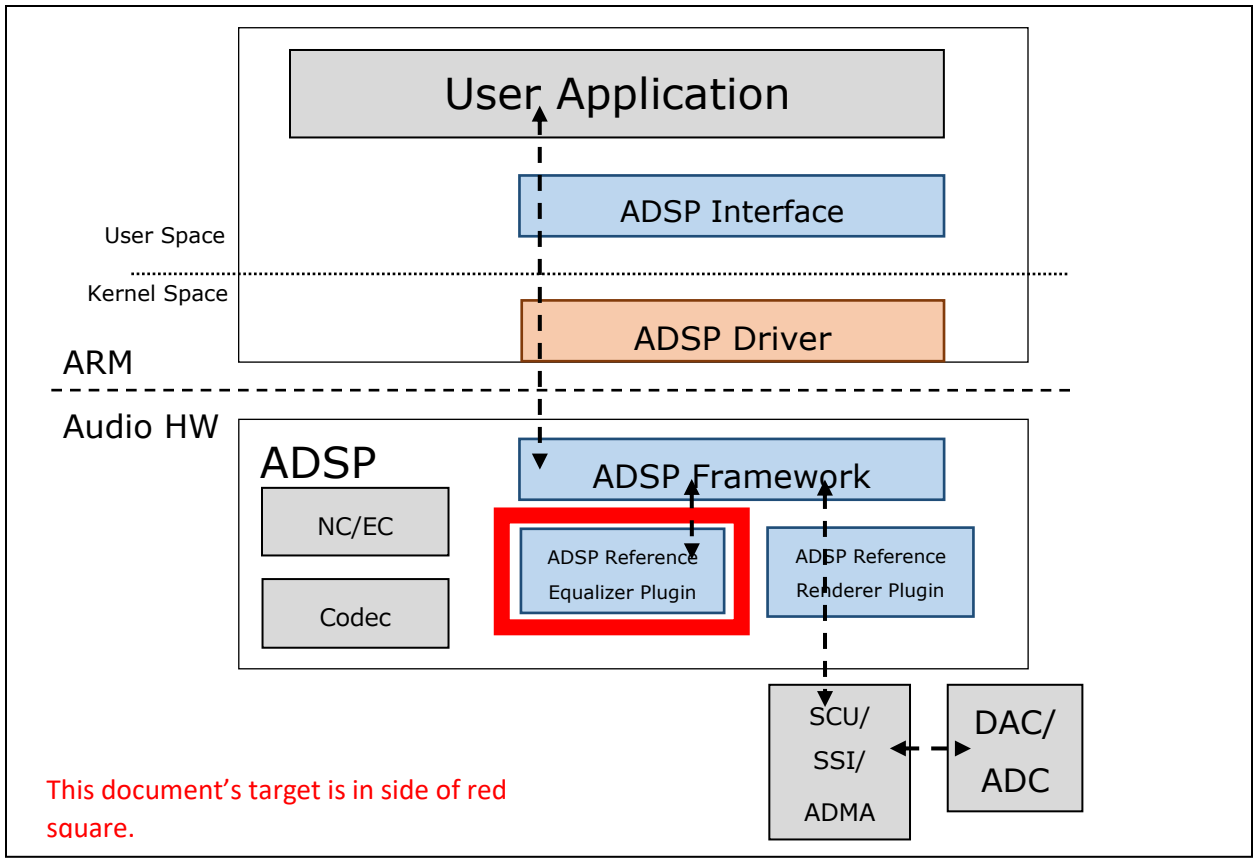


Figure 2.1.2.1 The software architecture

2.1.3. Specification overview

Equalizer changes frequency characteristic of the audio signal based on the parameter that was set and performs sound quality correction. Filter processing is performed for the PCM data which were input, and coordinates ingredient of the specific frequency band.

Table 2.1.3.1 shows the basic specification and Table 1-2 shows the support specification of Equalizer.

Table 2.1.3.1 Basic Specifications

Item	Description
DSP	Cadence Design Systems, Inc. HiFi2
Complier	Xtensa C and C++ Compiler (version 12.0.4)
Endian	Little Endian

Table 2.1.3.2 Supported Specifications

Item	Description
Input data format	16-bit / 24-bit linear PCM (fixed point)
Output data format	16-bit / 24-bit linear PCM (fixed point)
Sampling frequency (Hz) supported	48000 / 44100 / 32000
Number of channels supported	Max 2 channels
Filter	Direct form II(second-order IIR digital biquad filter)
Band number of partitions	9 Band
Reentrant	Supported
Other	Coefficient change function

2.1.4. Memory specification

Table 2.1.4.1 Memory Size Requirements

Memory type	Location	Memory area name	Size (in bytes)
Instruction	ROM	Instruction area	16293
Data		Constant table area	
		Other area(Depend on the compiler)	
	RAM	Persistent area	596
		Input buffer	8192
		Output buffer	8192
		Scratch area	8192
		Structure	240
		Stack	336
		Other area(Depended on the compiler)	0

[Note] Area whose location is shown as ROM in the location column can be included in RAM or ROM.

[Note] Area whose location is shown as RAM in the location column can be included in RAM only.

2.1.5. Related documents

Table 2.1.5.1 Related documents

No.	Name	Published by
[1]	ADSP Framework User's Manual	Renesas Electronics Corporation

2.2. Prcoessing flow

Figure 3.1 shows a flow diagram of processing performed by an application which uses this software. These steps are grouped into 6 stages: Startup API, Parameters Setting, Memory Allocation, Initialization, Parameter Getting and Equalizer Executing.

The basic steps executed by the framework are shaded. The steps defined by the user framework are white. Design the process to suit the target system.

[Note] **Set the equalizer parameter** phase performs for only Parametric Equalizer or Graphic Equalizer. If we set them currently it will return error code: (XA_EQZ_CONFIG_NONFATAL_ERR_OVERWRITE).

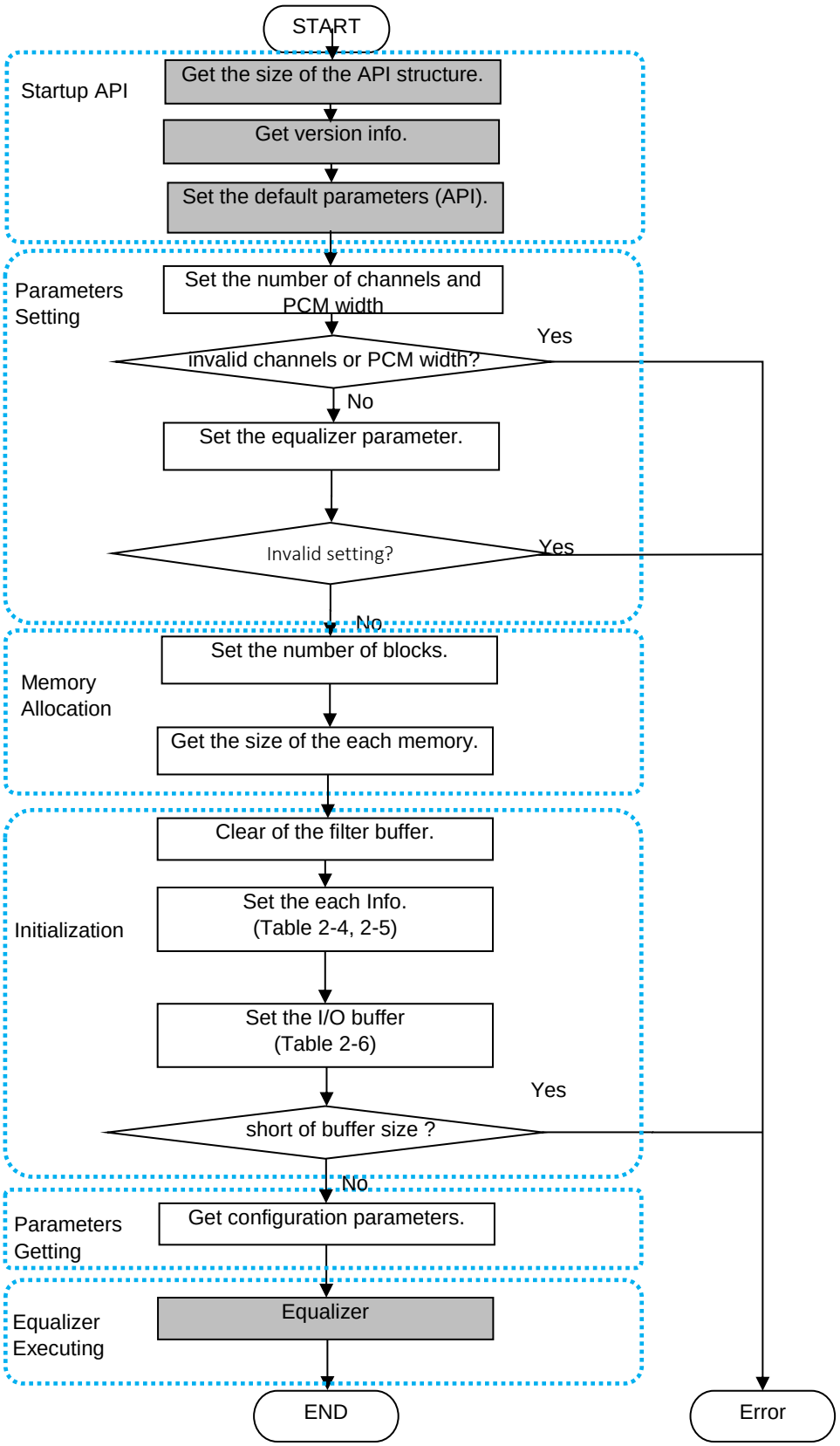


Figure 0-2 Application Processing Flow

3. ADSP TDM Renderer/Capture plugins

3.1. Overview

This section provides an overview of the Time-Division Multiplexing (TDM) Renderer plugin. It contains TDM renderer and capture function.

3.1.1. Specifications Outline

TDM Renderer function plays the multiplexing audio signal based on the parameter that was set.

TDM Capture function capture/record the multiplexing audio signal based on the parameter that was set.

Table 3.1.1.1 Basic Specification

Item	Description
DSP	Cadence Design Systems, Inc. HiFi2
Compiler	Xtensa C and C++ Compiler (version 12.0.4)
Endian	Little Endian

Table 3.1.1.2 Supported TDM Renderer function Specifications

<table>
<tr><td>Item</td><td colspan="4">Description</td></tr>
<tr><td rowspan="6">Input data format</td><td colspan="2" rowspan="2">Channel number</td><td colspan="2">PCM bit-width (fix-point)</td></tr>
<tr><td>16-bit</td><td>24-bit</td></tr>
<tr><td rowspan="2">6ch</td><td>3 * 2ch</td><td>○</td><td>○</td></tr>
<tr><td>1 * 6ch</td><td>○</td><td>○</td></tr>
<tr><td rowspan="2">8ch</td><td>4 * 2ch</td><td>○</td><td>○</td></tr>
<tr><td>1 * 8ch</td><td>○</td><td>○</td></tr>
<tr><td>Output data format</td><td colspan="4">Time-division Multiplexing 16-bit/24-bit linear PCM (fixed point)</td></tr>
<tr><td>Input Sampling frequency (Hz) supported</td><td colspan="4">48000 / 44100 / 32000</td></tr>
<tr><td>Output Sampling frequency (Hz) supported</td><td colspan="4">48000 / 44100</td></tr>
<tr><td>Number of channels supported</td><td colspan="4">TDM format channel (6 / 8)</td></tr>
<tr><td>Reentrant</td><td colspan="4">Supported</td></tr>
<tr><td>Other</td><td colspan="4">-</td></tr>
<tr><td>Restrictions</td><td colspan="4">-</td></tr>
</table>

Table 3.1.1.3 Support TDM Capture function Specification

<table>
<tr><td>Item</td><td colspan="4">Description</td></tr>
<tr><td rowspan="6">Output data format</td><td colspan="2" rowspan="2">Channel number</td><td colspan="2">PCM bit-width (fix-point)</td></tr>
<tr><td>16-bit</td><td>24-bit</td></tr>
<tr><td rowspan="2">6ch</td><td>3 * 2ch</td><td>○</td><td>○</td></tr>
<tr><td>1 * 6ch</td><td>○</td><td>○</td></tr>
<tr><td rowspan="2">8ch</td><td>4 * 2ch</td><td>○</td><td>○</td></tr>
<tr><td>1 * 8ch</td><td>○</td><td>○</td></tr>
<tr><td>Input data format</td><td colspan="4">Time-division Multiplexing 16-bit/24-bit linear PCM (fixed point)</td></tr>
<tr><td>Output Sampling frequency (Hz) supported</td><td colspan="4">48000 / 44100 / 32000</td></tr>
<tr><td>Input Sampling frequency (Hz) supported</td><td colspan="4">48000 / 44100</td></tr>
<tr><td>Number of channels supported</td><td colspan="4">TDM format channel (6 / 8)</td></tr>
<tr><td>Reentrant</td><td colspan="4">Supported</td></tr>
<tr><td>Other</td><td colspan="4">-</td></tr>
<tr><td>Restrictions</td><td colspan="4">-</td></tr>
</table>

Table 3.1.1.4 Memory Size Requirements

Memory type	Location	Memory area name		Size (in bytes)	
Instruction	ROM	Instruction area		53747T.B.D.	
Data		Constant table area			
		Other area(Depended on the compiler)			
	RAM (TDM Capture)	Software work area		198284T.B.D.	
		Area breakdown	Persistent area	Size breakdown	67208T.B.D.
			Scratch area		65536T.B.D.
			DTCM area		6553664K
			Built-in descriptor area		42K
		User work area		34208T.B.D.	
		Area breakdown	OutputInput buffer	Size breakdown	32768T.B.D.
			Structure		1440
		Stack area		944	
		Other area(Depended on the compiler)		0	
	RAM (TDM Renderer)	Software work area		165516	
		Area breakdown	Persistent area	Size breakdown	67208
			Scratch area		32768
			DTCM area		65536
			Built-in descriptor area		4
		User work area		34224	
		Area breakdown	InputOutput buffer	Size breakdown	32768T.B.D.
			Structure		1456 T.B.D.
		Stack area		896T.B.D.	
		Other area(Depended on the compiler)		0T.B.D.	

[Note] Area whose location is shown as ROM in the location column can be included in RAM or ROM.
[Note] Area whose location is shown as RAM in the location column can be included in RAM only.
[Note] Built-in is a memory area to allocate descriptor memory, which need in the DMAC transfer type of plugin.

Table 3.1.1.5 Version Information

Item	Description
Library Version information	Version 1.0.0About T.B.D. MHz [Note] Frequency = 48kHz/Number of channels = 2/There is no coefficient change
API Version information	Version 1.0.0

3.1.2. Configuration

Figure shows an example of the ADSP system configuration which uses renderer function.

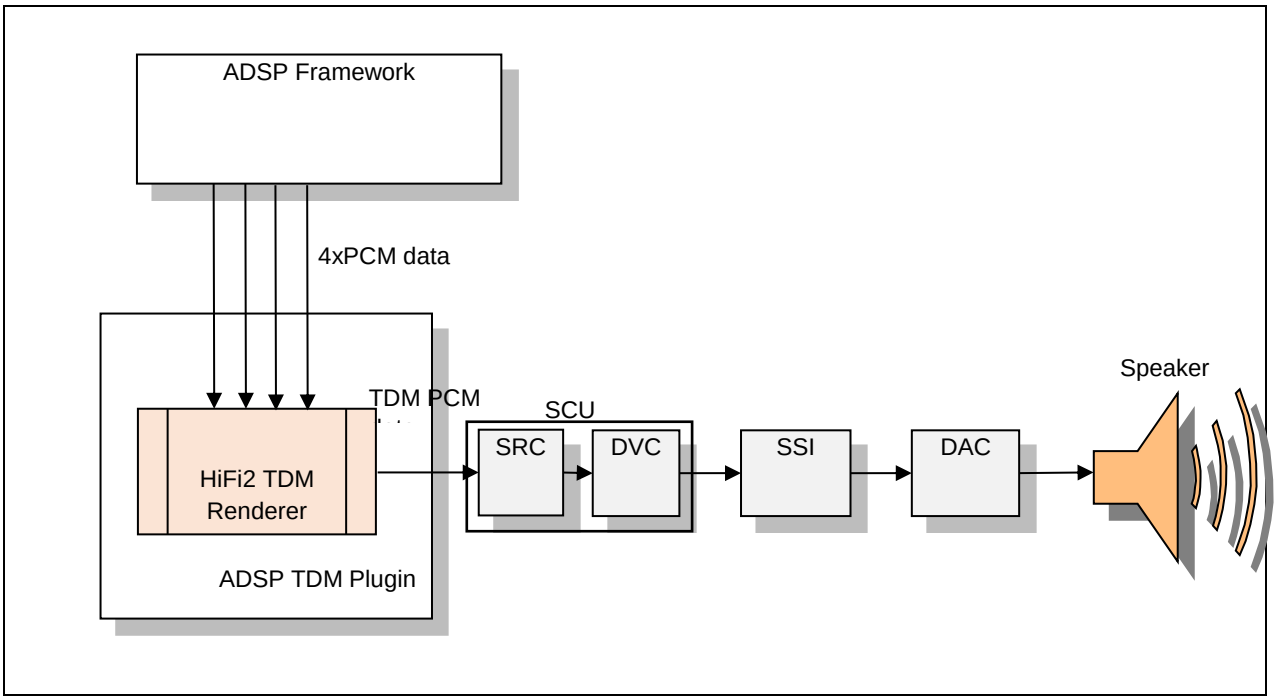


Figure 3.1.2.1 Example of the ADSP System Configuration for TDM renderer function

Figure shows an example of the ADSP system configuration which uses capture function.

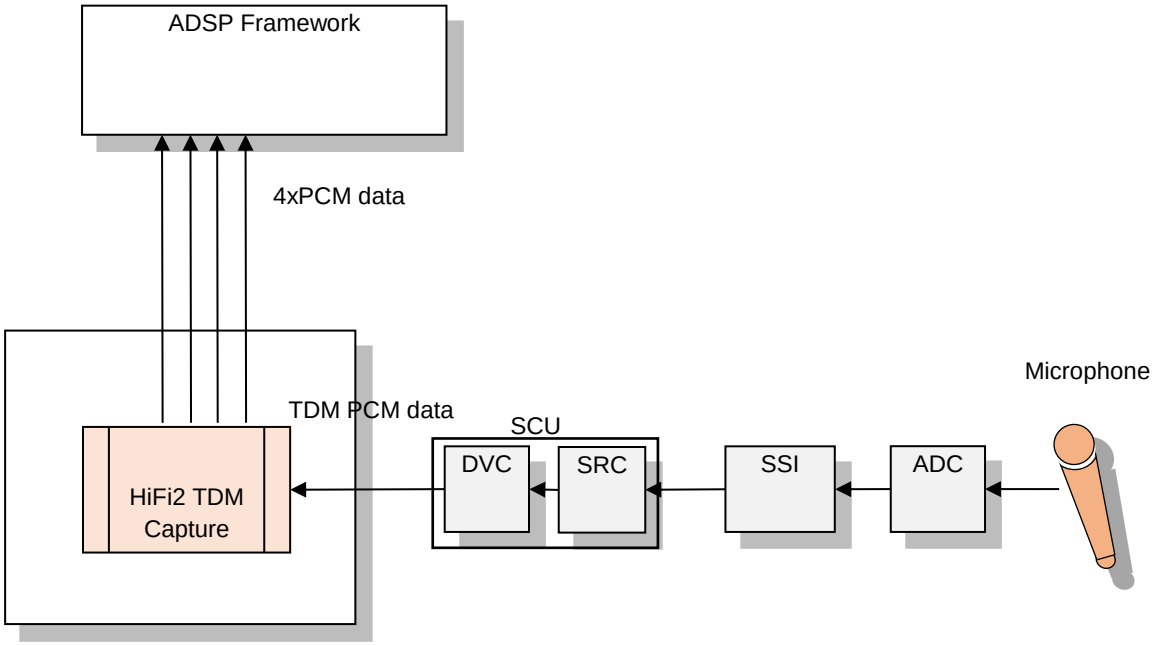


Figure 3.1.2.2 Example of the ADSP System Configuration for capture function

ADSP Framework

It controls ADSP Plugin. It is software provided separately as Framework.

HiFi2 TDM Renderer (ADSP TDM Plugin)

It performs merge multiple input PCM data and output to other audio device. It is this software set up as ADSP TDM Plugin.

HiFi2 TDM Capture (ADSP TDM Plugin)

It performs split multiple output PCM data from TDM input received from other audio device. It is this software set up as ADSP TDM Plugin.

PCM data

16-bit / 24-bit linear PCM data which is a processing by this software.

SCU

It performs sampling rate converters (SRC) and volume control (DVC).

SSI (*)

Send or receive audio data interfacing with a variety devices of offering I2C format.

DAC/ADC

The DAC/ADC converts a digital 16-bit/24-bit linear PCM data into analog signal and vice versa.

3.2. Processing flow

Figure shows a flow diagram of processing performed by an application which uses this software. It applies for both case: TDM renderer and TDM capture.

The basic steps executed by the framework are white. The steps defined by the user framework are shaded. Design the process to suit the target system.

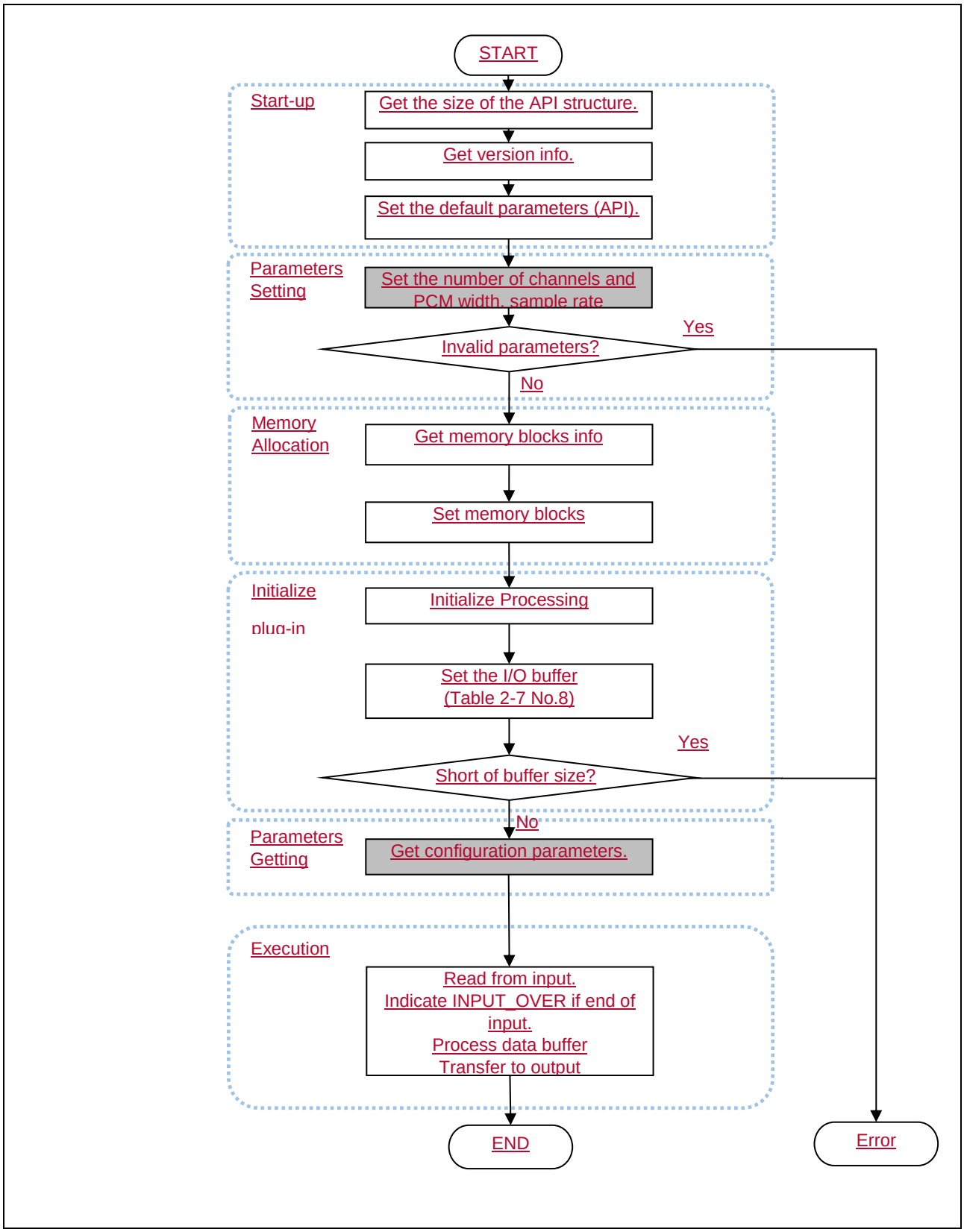


Figure 3.2.1 Example of the Application Processing Flow

3.3. Apendix

Below matrix tables show behavior of TDM plugin when user sets different sampling rate (Fs) (Hz) to plugin.

Table 3.3.1 Matrix table for sampling rate setting of TDM Renderer

Input Fs / Output Fs	32000	44100	48000
32000	-	-	-
44100	○	○	○
48000	○	○	○
0 (Non-use SRC)	*	○	○

Table 3.3.2 Matrix table for sampling rate setting of TDM Capture

Output Fs / Input Fs	32000	44100	48000
32000	-	-	-
44100	○	○	○
48000	○	○	○
0 (Non-use SRC)	*	○	○

○ : Plugin runs as normal

- : Plugin returns error due to invalid sample rate setting

* : Plugin enables SRC module automatically to perform sample rate conversion

III. Audio Framework Specification

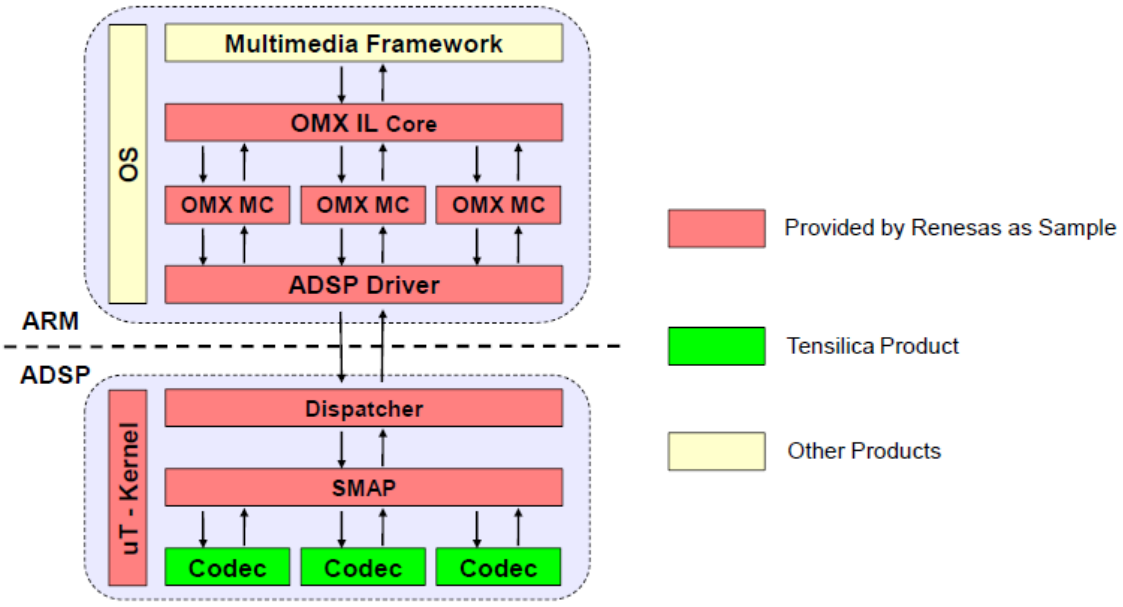
1. Technical Terms and Abbreviations

Technical term /Abbreviation	**Description**
OS	Operating system
CPU	Central processing unit
ADSP	Audio digital signal processor
OMX IL	OpenMax integration layer
OMX MC	OpenMax media component
IST	Interrupt service thread
OMX	OpenMax

2. Overview of audio framework

The audio framework is an audio solution for a whole system, which has the capability to execute multiple codecs as well as easily to customize. The framework is divided into 2 main parts as the below figure: the

upper part runs on ARM core and is called CPU side; the lower part is executed on Tensilica Audio DSP (ADSP) and called ADSP side.



The audio framework includes following modules:

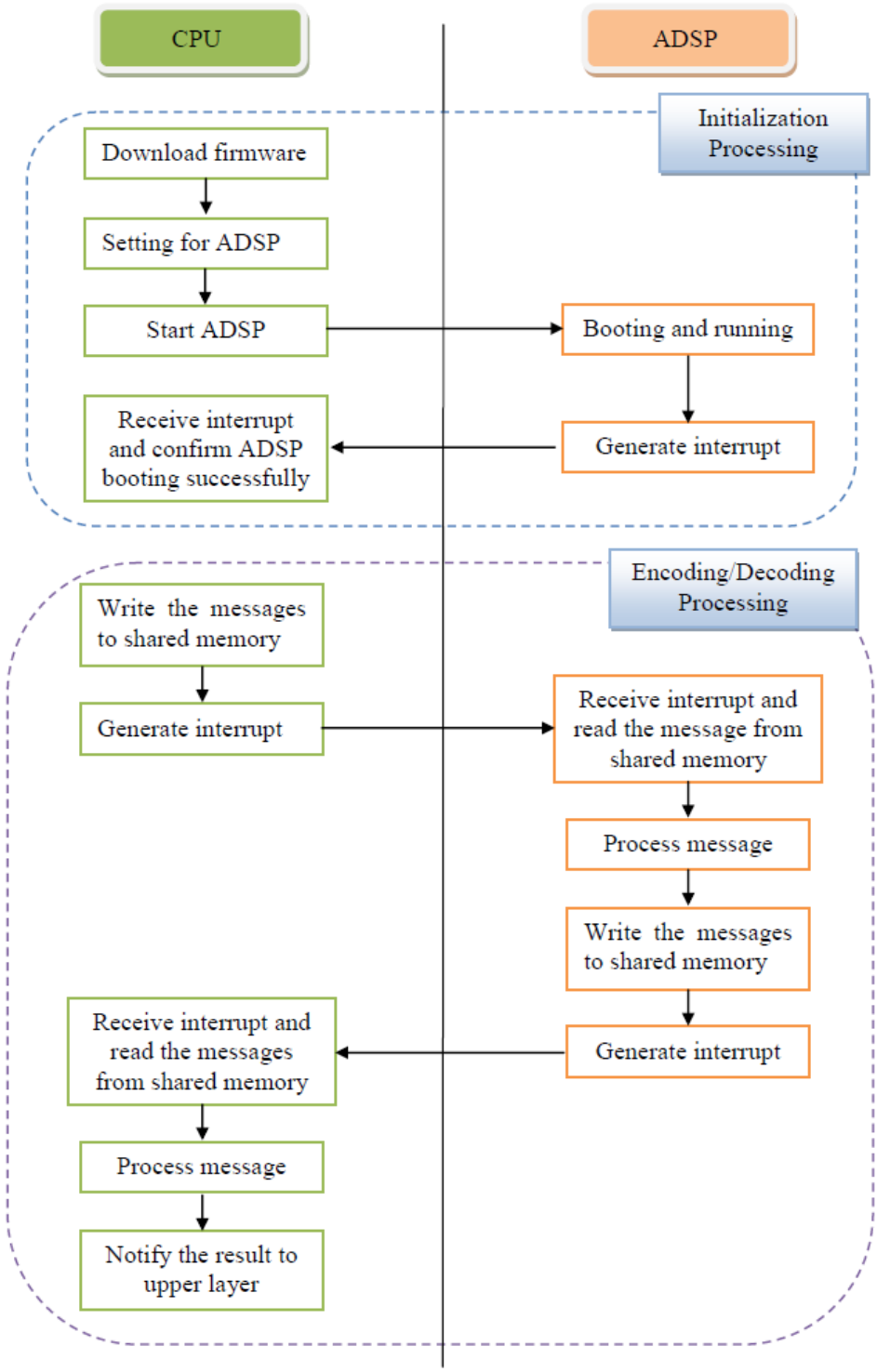
- **Multimedia Framework:** The framework is provided by OS to supportaudio playback. For WEA7, DirectShow is the target multimedia framework.
- **OMX IL Core:** OMX interface to communicate between multimedia framework and OMX MC.
- **OMX MC:** OMX IL media component of each codec.
- **ADSP Driver:** Driver to control ADSPmodule.
- **Dispatcher:** Communicate with ADSP Driver to get the command from CPU side (ARM) and send the response from ADSP side (ADSP).
- **uT-kernel:** Real time OS is used for small embedded system.
- **SMAP:** The framework which utilizes the services provided by uT-Kernel to control the codec execution.
- **Codec:** Tensilica audio codec libraries

3. Communication process

3.1. Communication between CPU and ADSP

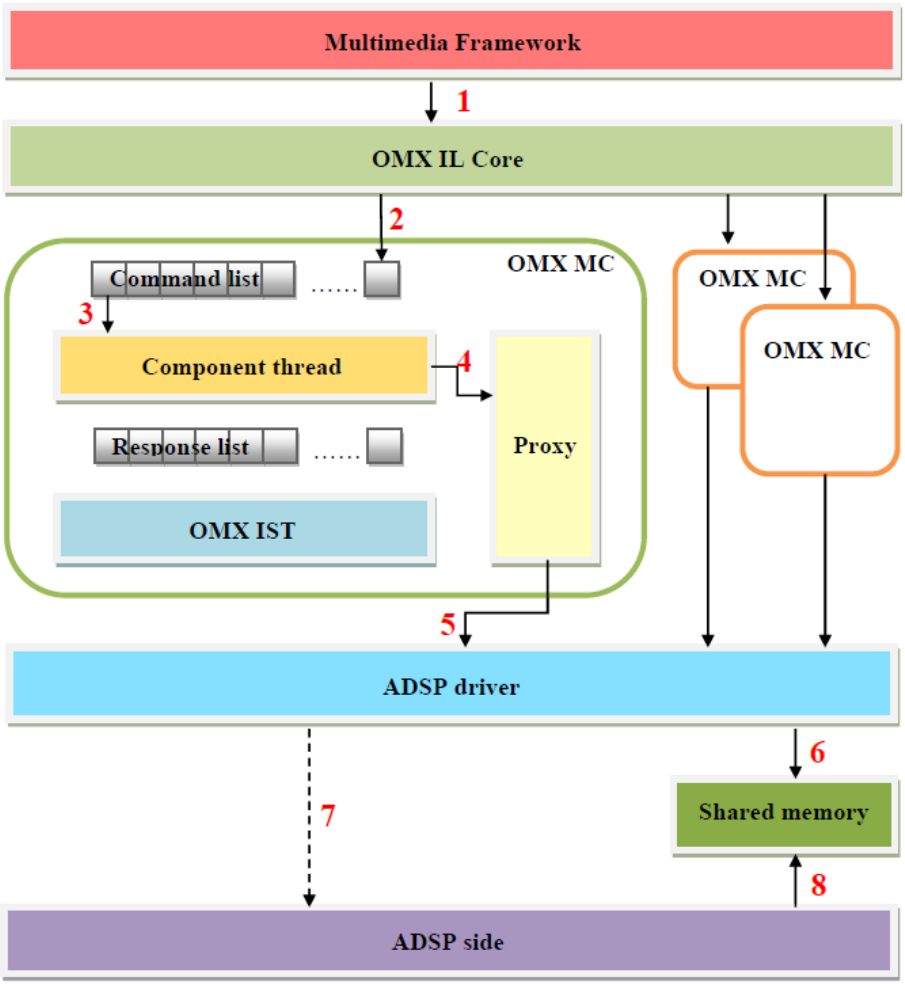
In the audio framework, CPU and ADSP communicate with each other by using registers and shared memory.

Below is the diagram of communication between CPU and ADSP



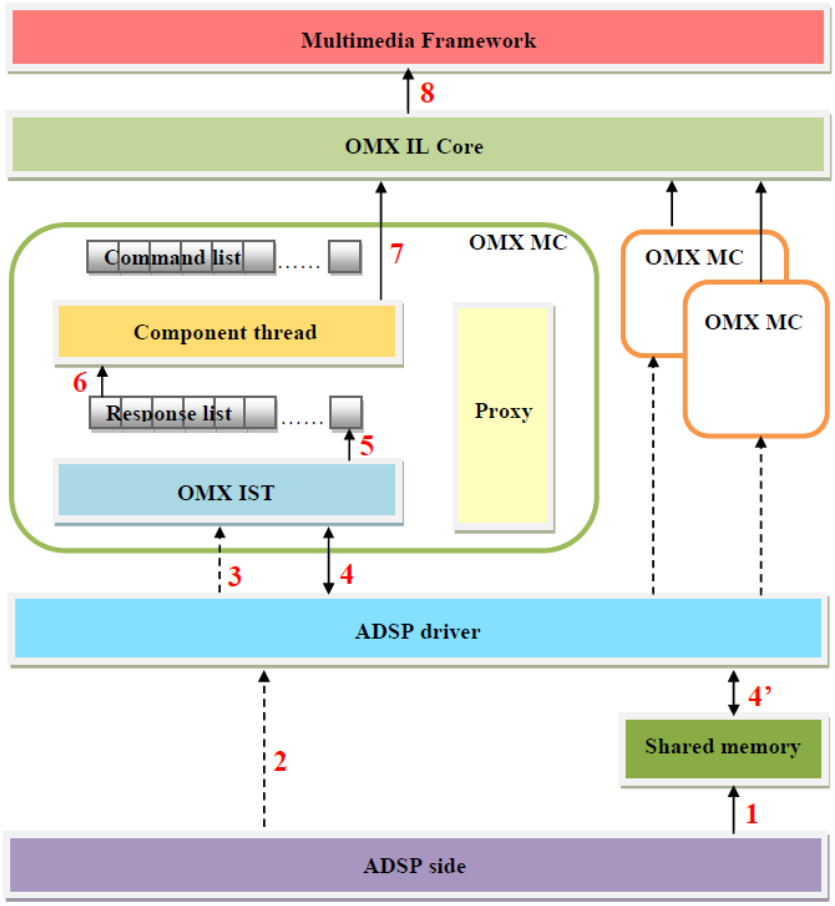
3.2. Communication inside CPU

3.2.1. Send data from CPU to ADSP



When the multimedia framework submits the messages (buffers or commands) through the functions or macros of OMX IL core (1), the messages are added to the tail of command list of an OMX MC (2). Besides, the Component thread polls to read the command and response list ontinuously. If the command list is not empty, the Component thread reads the first element of command list (3) and sends it to Proxy. Proxy writes this message the shared memory through the functions ADSP driver (4, 5, and 6). Then ADSP driver enables interrupt to ADSP side (7) to get data for encoding or decoding of ADSP side (8).

3.2.2. Receive data from ADSP side



When ADSP side finishes encoding or decoding a buffer, it writes message to shared memory (1) and enables interrupt to CPU side (2). When ADSP driver receives the interrupt from ADSP side, it wakes up all OMX IST of OMX MCs (3) by an event (RECEIVE_EVENT).

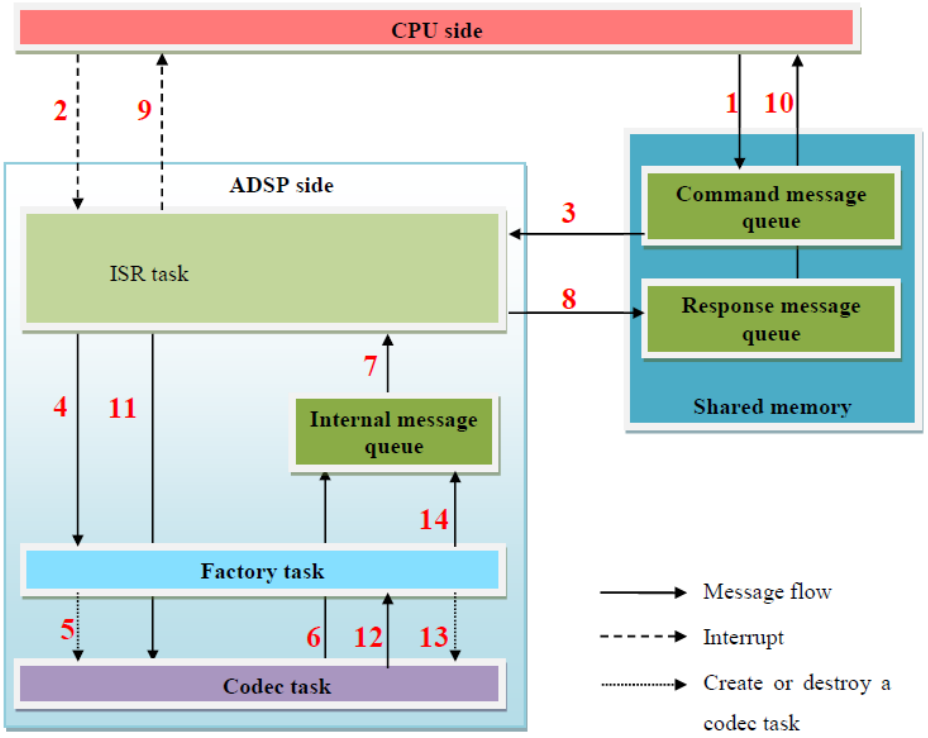
When all OMX MCs are waked up, they find the messages appreciated to themselves from shared memory (4, 4') (Means that OMX MC of AAC codec only processes messages related to AAC codec). If they find comfortable messages, these messages are processed and added the tail of their response list (5). All OMX MCs read messages until the response list is empty, then one of OMX MCs clear RECEIVE_EVENT event and status interrupt sent from ADSP side.

Besides, the Component thread polls to read the command and response list continuously. If the response list is not empty, the Component thread reads the first element of response list (6) and sends it to multimedia framework (7, 8).

3.3. Communication inside ADSP

ADSP side is organized as a set of tasks; each task is executed in its own thread. The ISR of shared memory is in charge of receiving interrupts from CPU. For each incoming command message, it signals the ISR task to dispatch the message to appropriate recipient task. After the command message is processed and the response message is ready, ISR sends an interrupt to inform CPU.

There are three kinds of task: ISR task, factory task and codec task. These tasks communicate with CPU by interrupt, command message queue and response message queue of shared memory. Besides, they use an internal message queue to contain the response messages before pushing these messages to shared memory.



3.3.1. Initialize codec task

When CPU wants to initialize a codec, it pushes a register message to the command message queue (1) and raises an interrupt (2). ISR task receives the interrupt and gets the message from command message queue (3), then sends it to factory task (4). Factory task will create codec task (5). After being created, codec task will push a response message to the internal message queue (6). Then ISR task gets the message (7), copies it to the response message queue (8) and raises an interrupt to CPU (9). CPU receives the interrupt, gets the message (10) and knows that the codec is created successfully.

3.3.2. Process the commands from CPU

When the encoding / decoding process begins and CPU wants to send a command to ADSP, three first steps (1), (2), (3) are the same with section 3.3.1. Then ISR task will send the message directly to the corresponding codec task based on the codec ID in the message (11). The codec task processes the message and the last steps (6), (7), (8), (9), (10) are also the same with section 3.3.1.

3.3.3. Destroy codec task

When the encoding / decoding process ends and CPU wants to stop a codec task, CPU pushes an unregister message to the command message queue (1) and the steps (2), (3), (11) are the same with section 0. The

codec task will sends the message to factory task (12), then factory task will destroy the codec task (13) and push a response message to the internal message queue. The last steps (7), (8), (9), (10) are the same with section 3.3.1 and 3.3.2 and CPU will know that the codec has stopped successfully.