d

Audio Primer

System High-Level Design Document

Analysis & Design

MT6771

Doc No: AD6771-AJ4C-SHD-V1.0EN
Version: V1.0
Release date: 2017-08-31
Classification: Internal

Keywords
System High-Level Design Document

MediaTek Inc.

Postal address
No. 1, Dusing 1st Rd. , Hsinchu Science Park, Hsinchu City, Taiwan 30078

MTK support office address
No. 1, Dusing 1st Rd. , Hsinchu Science Park, Hsinchu City, Taiwan 30078

Internet
http://www.mediatek.com/

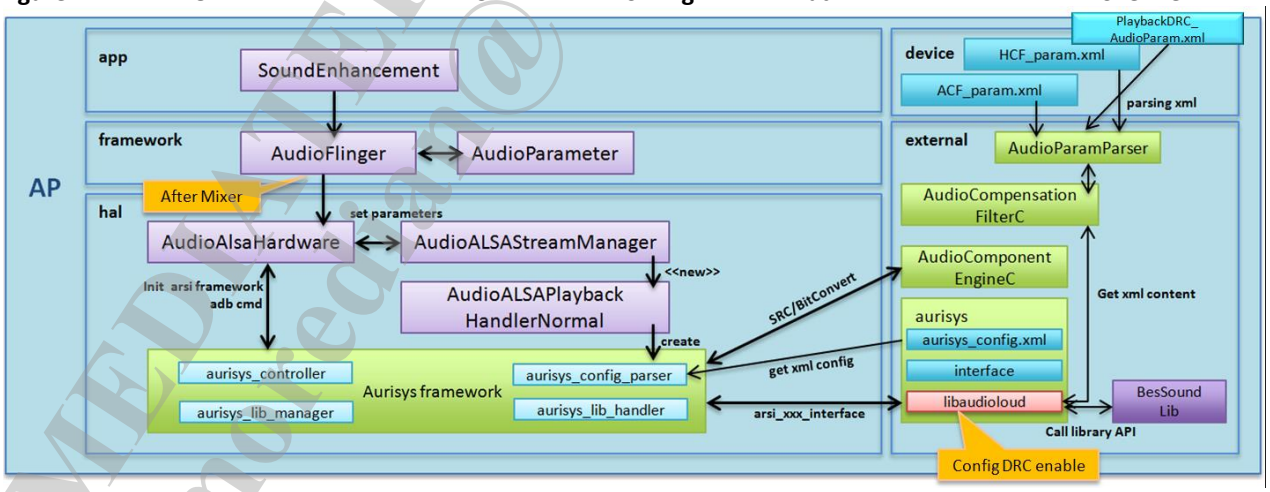
Document Revision History

Revision	Date	Author	Description
V1.0	2017-08-31		Initial Release

Table of Contents

Lists of Tables

Lists of Figures



1 Audio Overview

In this document, we'll introduce audio features such as volume control, audio effects, audio frameworks, etc. First, we'll give an overview of supported features and audio block diagram. Then, we'll introduce the volume control (including customization and EM settings). In audio effect chapter, the HCF/ACF and SRS integration guide are described. The integration guideline of audio features will then be introduced (BT/FM/I2S...). To verify the audio sound, we also have an introduction for factory mode and audio loopback test functions in Audio Test Suite Chapter. Finally, the audio framework and stagefright player will be described.

1.1 Audio Feature Brief Summary

1.1.1 Audio Feature and Blocks

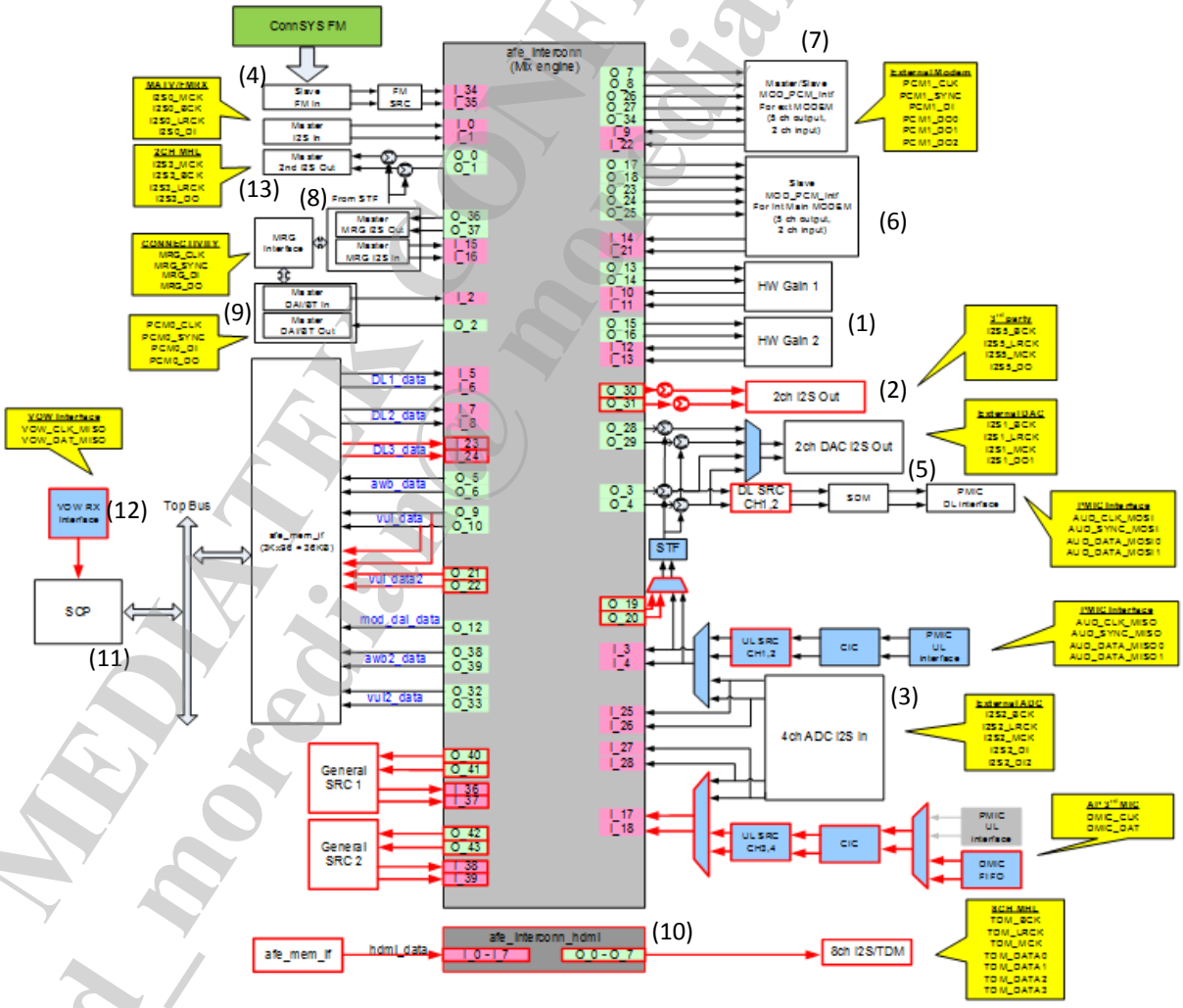


Figure 1. MT6771 Audio Blocks

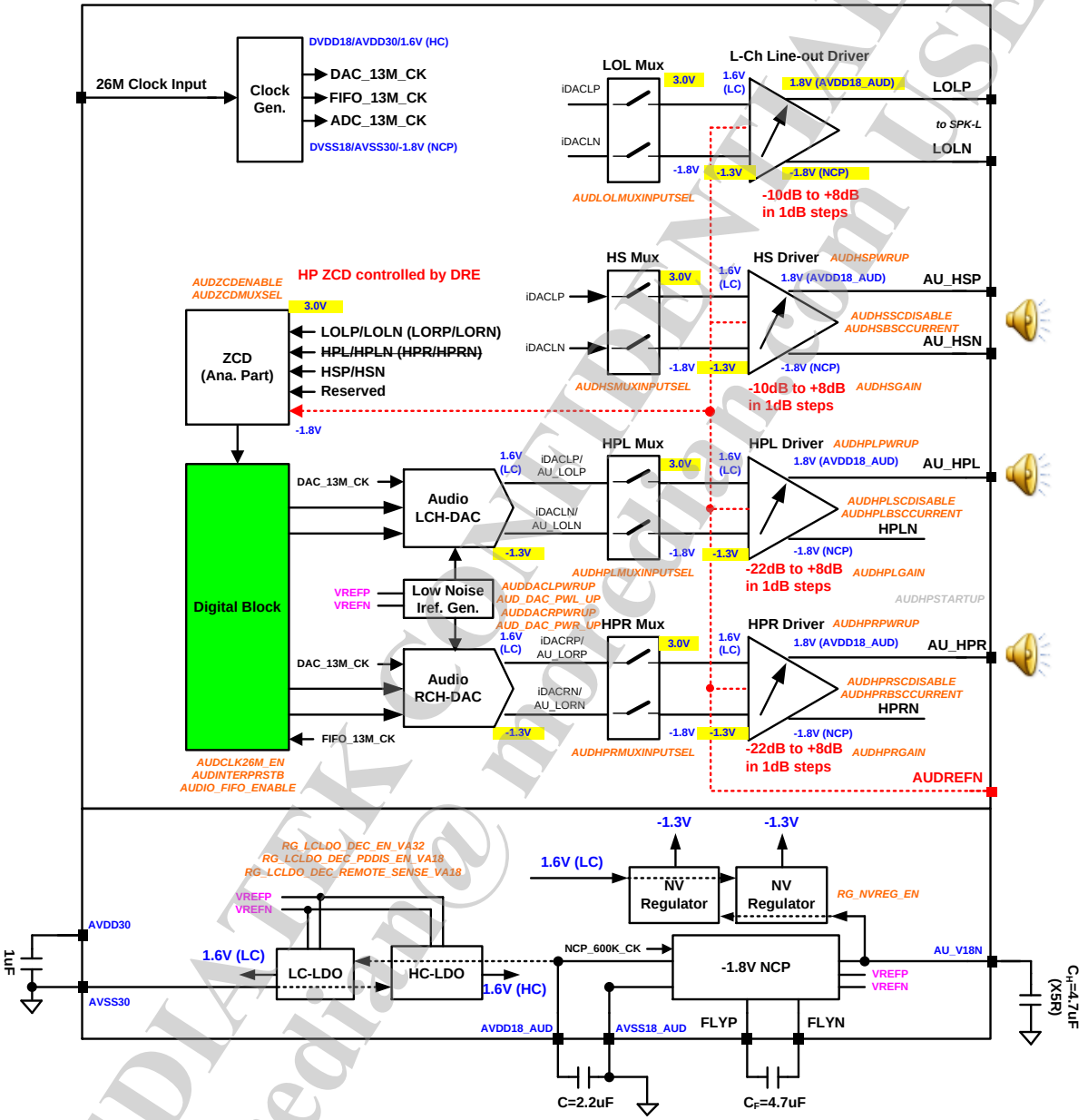


Figure 2. MT6358 Audio Blocks

- **MT6771**
  - Mediatek Proprietary audio interface connect to PMIC MT6358
  - Audio Playback with MTK PMIC – MT6358
    - Support 8k, 11.025k, 12k, 16k, 22.05k, 24k, 32k, 44.1k, 48kHz, 96kHz, 192kHz sample rate
  - Audio Recording with MTK PMIC – MT6358
    - Support 8k, 16k, 32k, 48kHz sample rate
  - Internal high resolution , high flexible HW gain in AFE interconnection……..(1)
  - 5 set Inter IC sound interface (I^2S)

- Master Output *3 (one is 4 ch) ……(2)(5)(13)
- Master Input * 1 ……(3)
- Master/Slave input(with SRC) ……(4)
- Master Mode supports 8, 11.025, 12, 16, 22.05, 24, 32, 44.1, 48, 88, 96, 176, and 192KHz sampling rate
- Slave Mode supports 8, 11.025, 12, 16, 22.05, 24, 32, 44.1, 48KHz sampling rate
- Master Output with 8ch TDM *1 ……(10)
- 16/32 bit bus width support
- Philip standard and Left just

- 2 set Pulse coded modulation interface(PCM)
  - Slave PCM for internal Modem*1……(6)
  - Master/Slave PCM interface for external modem * 1……(7)
- 1 set PCM/I2S merged interface (BTPCM + FM RX/TX I2S)……(8)
  - 4-pin interface for concurrently supporting I2S and PCM
  - PCM supports 8k/16k Hz sampling rate
  - I2S supports 32, 44.1, and 48 kHz sampling rate
- BT CVSD-Removal non-RF HW in AP-side……(9)
- 16/24bit stereo data format support
- Audio codecs
  - MP3, AAC, AAC+, AMR-NB, AMR-WB, OGG, WAV, APE
- Audio Post-Processing
  - BesLoudness
  - ACF
- 3rd support (need contact with 3rd by customer)
  - Dolby mobile
- Low power audio in MP3 playback, FM playback, FM record.
- Integrated Cortex-M4 DSP, open for development……(11)
- Voice Wake up, mic activity detector……(12)

➢ **MT6358**

- Mediatek proprietary audio interface connect to MT6771
- RX processing feature
  - 4 analog output; earpiece, left and right headphone, lineout
  - Stereo DAC
  - Differential earpiece driver up to 50mW in 32ohm, 88mW in 16ohm
  - Stereo single-ended headphone driver up to 11.25mA in 32ohm
  - Adjustable earpiece, headphone and SPK Amp gain setting
  - Sample rate of 8K,16K,44.1K,48K
  - Best 109dB SNR in headphone output
  - Best -95dB THD
- TX processing feature

1.1.2 Audio Format Support

➢ Playback

- AAC / HE-AAC v1 / HE-AAC v2

- Android Orientated Support / Not use Mediatek IP
- 8 kHz ~ 96 kHz; 8 kbps ~ 320 kbps
- Mono/Stereo Support
- Bitrate Mode: VBR/CBR
- File Extension: .aac(ADTS, ADIF), .m4a, .mp4, .3gp, .ts
- Profile: 1) LC, HEAAC V1, V2; 2) LD, ELD
- LC, 48kHz, 128kbps, stereo → MCPS = 11; HEv1, 22.05kHz, 128kbps, stereo → MCPS = 31; HEv2, 22.05kHz, 32kbps, stereo → MCPS = 40; ELD, 44.1kHz, 128kbps MCPS = 13

■ AMR
- Android Orientated Support / Use Mediatek IP
- 8 kHz, 4.75kbps~12.2kbps
- Mono Support
- Bitrate mode: CBR
- File Extension: .amr
- 8kHz, 12.2kbps, mono -> Mediaserver:50 MCPS, AMR Dec: 7 MCPS

■ AWB
- Android Orientated Support / Use Mediatek IP
- 16 kHz, 6.6kbps~23.85kbps
- Mono Support
- Bitrate mode: CBR
- File Extension: .awb
- 16kHz, 23.85kbps, mono -> Mediaserver:60 MCPS, AMR Dec: 18 MCPS

■ MIDI
- Android Orientated Support / Not use Mediatek IP
- 22.05kHz
- Stereo Support
- File Extension: .mid, .midi, .smf, .rtttl, .xmf, .rtx, .ota, .imy

■ MP2
- Android Not Support / Use Mediatek IP
- 8 kHz ~ 48 kHz; 8 kbps ~ 320 kbps
- Mono/Stereo Support
- Bitrate Mode: VBR/CBR
- File Extension: .mp2
- Profile: MPEG1-Layer2, MEPG2-Layer2, MPEG2.5-Layer2
- MPEG1, 48kHz, 256kbps, stereo → 11 MCPS

■ MP3
- Android Orientated Support / Use Mediatek IP
- 8 kHz ~ 48 kHz; 8 kbps ~ 320 kbps
- Mono/Stereo Support
- Bitrate Mode: VBR/CBR
- File Extension: .mp3
- Profile: MPEG1-Layer3, MEPG2-Layer3, MPEG2.5-Layer3
- MPEG1, 48kHz, 256kbps, stereo → 15 MCPS

■ OGG VORBIS
- Android Orientated Support / Use Mediatek IP
- 8 kHz ~ 192 kHz; 10 kbps ~ 320 kbps

- Mono/Stereo Support
- Bitrate Mode: VBR
- File Extension: .ogg, .oga
- 48kHz, 250bps, stereo → MCPS = 25

- WAV (ADPCM)
  - Android Not Support / Use Mediatek IP
  - 8 kHz ~ 192 kHz; 4 kbps ~ 192 kbps
  - Bitrate Mode: CBR
  - File Extension: .wav
  - Profile: DVI/MA ADPCM; MS ADPCM
  - DVI/MA 48kHz, stereo → 3.44 MCPS; DVI/MA 192 kHz, stereo → 13.6 MCPS; MS, 48kHz, stereo -> 4.17 MCPS; MS, 192 kHz, stereo → 16.3 MCPS
- WAV (Alaw/Ulaw)
  - Android Orientated Support / Use Mediatek IP
  - 6 kHz ~ 96 kHz; 48 kbps ~ 3072 kbps
  - Mono / Stereo Support
  - File Extension: .wav
- WAV (Raw)
  - Android Orientated Support / Use Mediatek IP
  - 6 kHz ~ 96 kHz; 48 kbps ~ 3072 kbps
  - 1 channel ~ 8 channels support
  - File Extension: .wav
- APE
  - Android Not Support / Use Mediatek IP
  - 6 kHz ~ 96 kHz; 29 kbps ~ 836 kbps
  - Mono / Stereo Support
  - Bitrate Mode: VBR
  - File Extension: .ape
  - Profile: fast, normal, high, extra high
  - APE normal compress type, 44.1kHz, 675 kbps, stereo → MCPS = 50
- WMA (Need license)
  - Android Not Support / Use Mediatek IP
  - 8 kHz ~ 48 kHz; 5 kbps ~ 320 kbps
  - Mono / Stereo Support
  - Bitrate Mode: VBR
  - File Extension: .wma
  - Profile: wma v1; wma v2
  - 32 kHz, 22kbps, stereo → MCPS = 22.57
- FLAC
  - Android Orientated Support / Use Mediatek IP
  - 8 kHz ~ 48 kHz; 87 kbps ~ 396 kbps
  - Mono / Stereo Support
  - Bitrate Mode: VBR
  - File Extension: .flac
  - 44.1 kHz, 745kbps, level5, stereo → MCPS = 10

➢ Record

- AAC
  - Android Orientated Support / Not use Mediatek IP
  - 8 kHz ~ 48 kHz; 8kbps~160kbps
  - Mono / Stereo Support
  - Bitrate Mode: CBR
  - File Extension: .3gp, .aac
  - Profile: Low Complexity; High Efficiency; Enhanced Latency Delay
  - 48kHz, 128kbps, stereo (Low Complexity) → MCPS = 32; 48kHz, 128kbps, stereo (High Efficiency) → MCPS = 75; 48kHz, 128kbps, stereo (Low Latency Delay) → MCPS = 40;
- AMR
  - Android Orientated Support / Use Mediatek IP
  - 8 kHz; 4.75kbps~12.2kbps
  - Mono Support
  - Bitrate Mode: CBR
  - File Extension: .3gp, .amr
  - 8kHz, 12.2kbps, mono → MCPS = 39
- AWB
  - Android Orientated Support / Use Mediatek IP
  - 16 kHz; 6.60kbps~23.85kbps
  - Mono Support
  - Bitrate Mode: CBR
  - File Extension: .3gp, .awb
  - 16kHz, 23.85 kbps, mono → MCPS = 77
- OGG
  - Android Not Support / Use Mediatek IP
  - 8 kHz~ 48 kHz; 31.98kbps~202.96kbps
  - Mono / Stereo Support
  - Bitrate Mode: VBR
  - File Extension: ..ogg
  - 48kHz, 128 kbps,stereo → MCPS = 40
- ADPCM
  - Android Not Support / Use Mediatek IP
  - 8 kHz~ 48 kHz; 4kbps~192kbps
  - Mono / Stereo Support
  - Bitrate Mode: CBR
  - File Extension: ..wav
  - 16 kHz, stereo, DIV/MA → MCPS = 2.88; 48 kHz, stereo, DIV/MA → MCPS = 8.5; 16 kHz, stereo, MS → MCPS = 5.08; 48 kHz, stereo, MS → MCPS = 14.97

1.2 Audio SW Architecture

The following figures are the audio software architecture and data flow. The software components and data path can be found in the figures. The detailed descriptions can be found in Audio Frameworks chapter.

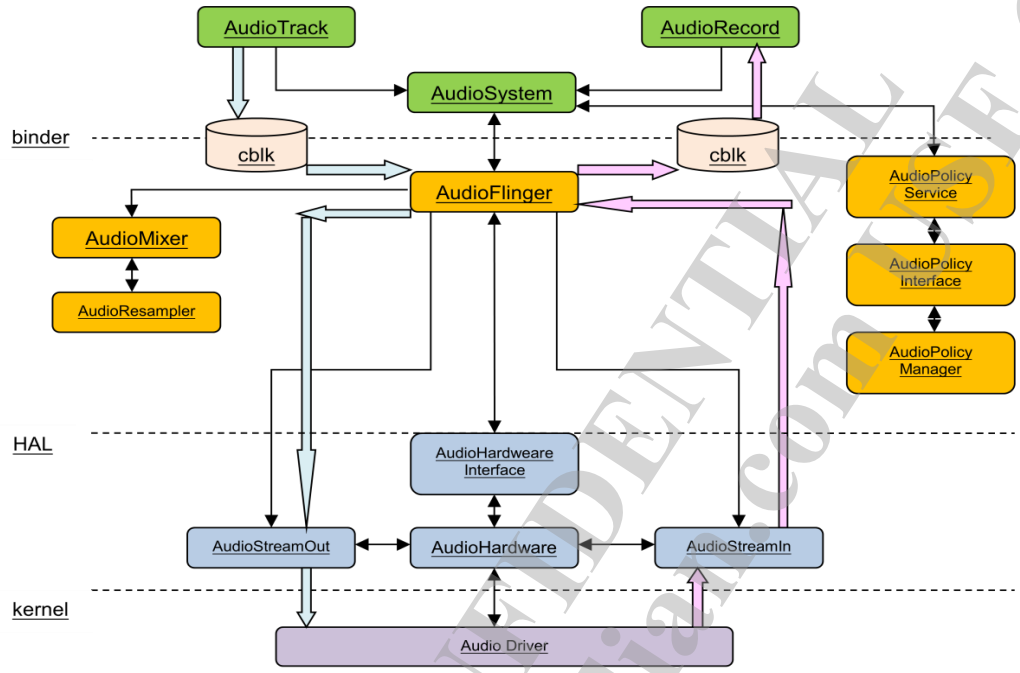


Figure 1. Audio Software Architecture

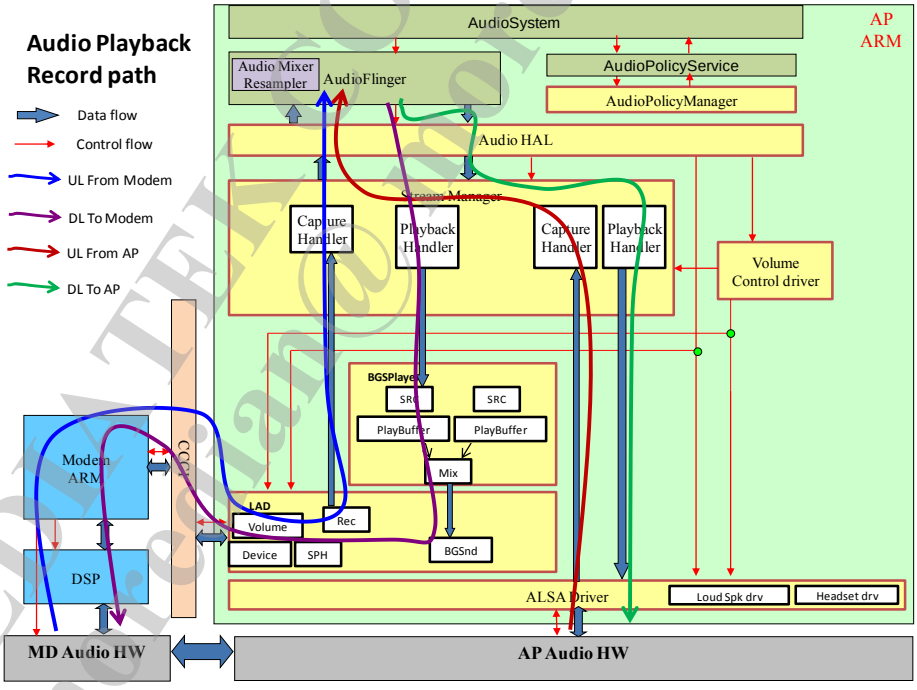
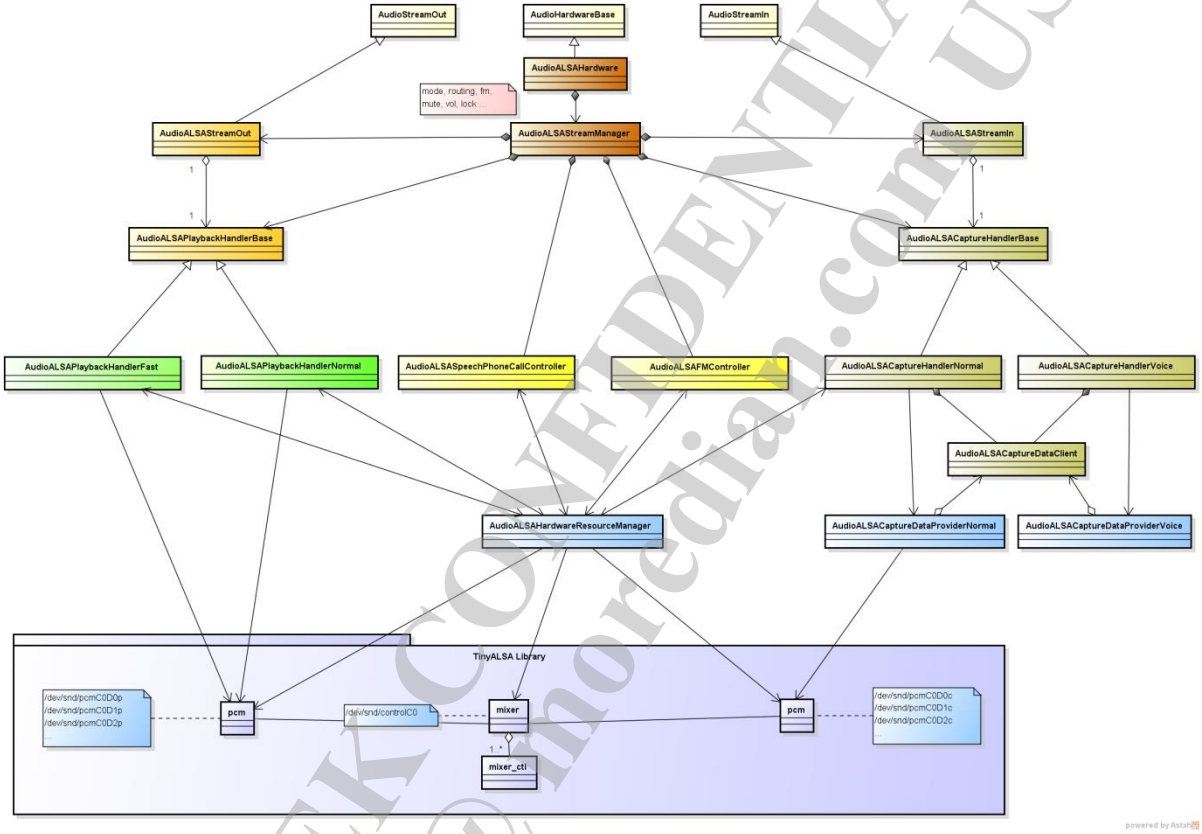


Figure 2. Audio SW Data Path

1.3 Audio HAL

The following figures are the audio HAL architecture.



The top control interfaces of HAL are AudioALSAHardware, AudioALSAStreamOut, and AudioALSAStreamIn. There classes all use AudioALSAStreamManager to control audio mode, open/close input/output streams. And hence AudioALSAStreamManager will have all audio environment info like mode, routing, volume, and mute information.

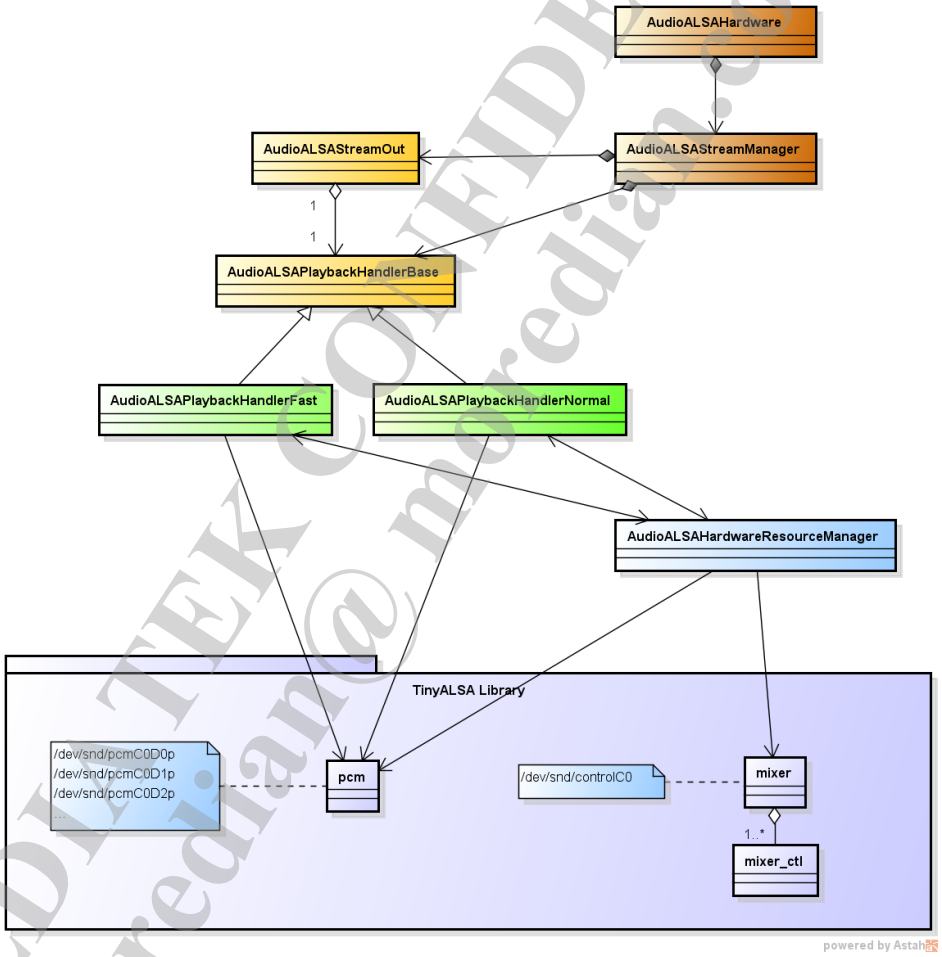
1.3.1 Audio HAL Playback

AudioALSAStreamOut: A class to do write / standby / routing operation but…

- Do not implement detail in itself but using a Playback Handler to bypass open/close/routing/write operations
- Different scenarios use different types of playback handlers
- Stream Out will call createPlaybackHandler() in StreamManager to acquire a pointer of playback handler when 1st write().
- Also, destroyPlaybackHandler() when standby()

Do not setting HW registers in playback handler but using TinyALSA Library to call

- pcm_open() / pcm_close()
  - To control AFE hw path, memory setting
- mixer_open() / mixer_close()
  - To get various types of mixer control like, Speaker_Amp_Switch, Voice_Amp_Switch, Audio_Amp_R_Switch, and Audio_Amp_L_Switch to control codec driver.
- pcm_write()
  - To write pcm data to sram/dram



1.3.2 Audio HAL Record

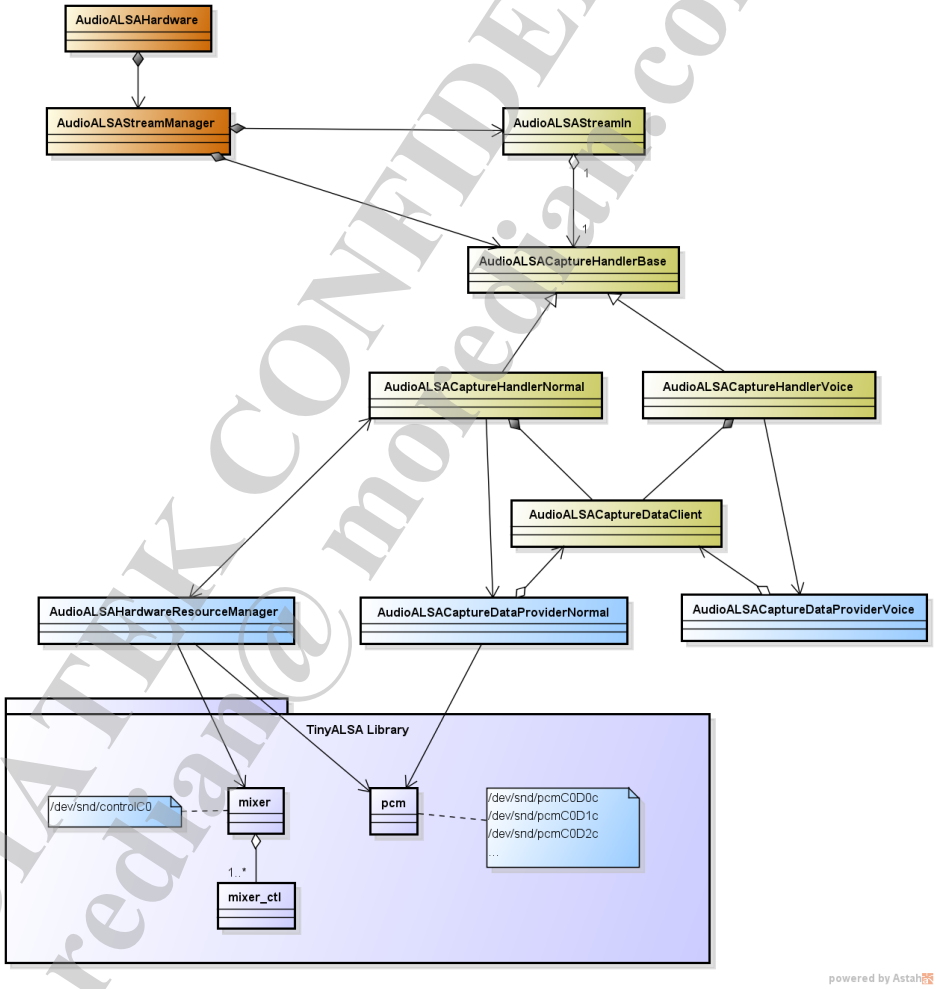
AudioALSAStreamIn: A class to do read / standby / routing operation but…

- Do not implement detail in itself but using a Capture Handler to bypass open/close/routing/read operations
- Different scenarios use different types of capture handlers
- Stream In will call createCaptureHandler() in StreamManager to acquire a pointer of capture handler when 1st read().

- Also, destroyCaptureHandler () when standby()

Do not setting HW registers in capture handler but using TinyALSA Library to call

- pcm_open() / pcm_close()
  - To control AFE hw path, memory setting
- mixer_open() / mixer_close()
  - To get various types of mixer control like, Audio_ADC_1_Switch, Audio_ADC_2_Switch, Audio_Preamp1_Switch, Audio_Preamp2_Switch to control codec driver.
- pcm_read()
  - To read pcm data to sram/dram



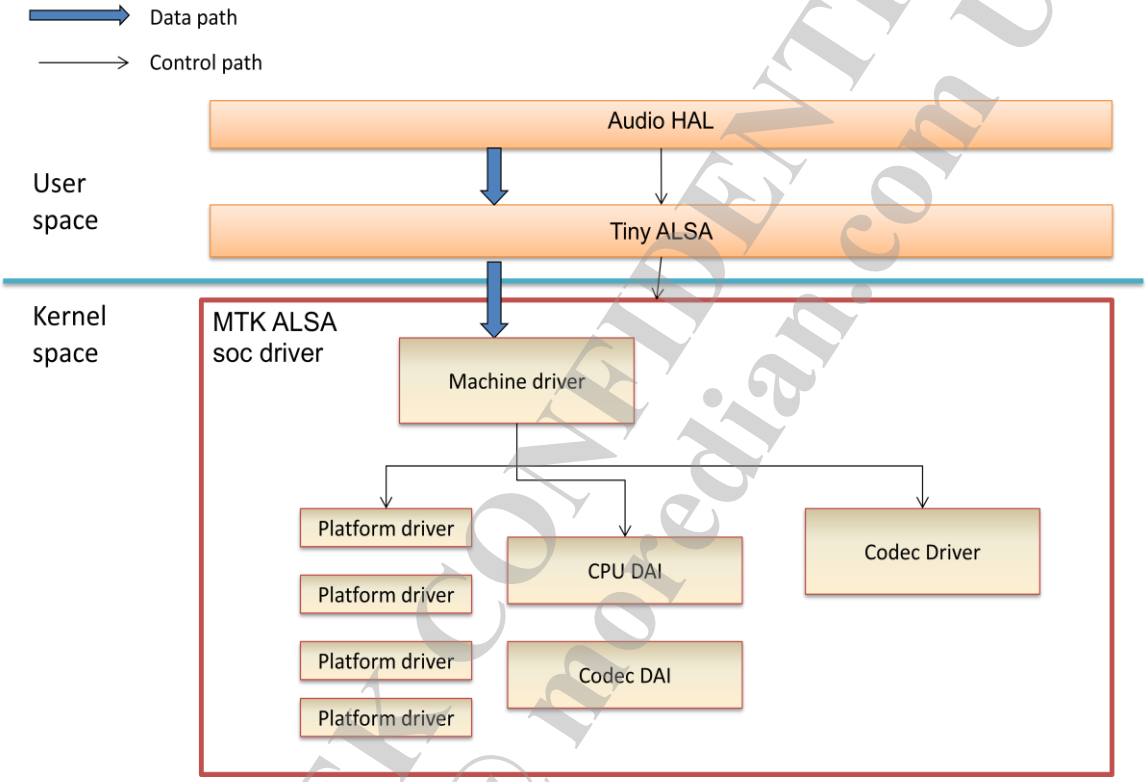
1.4 ALSA Driver Architecture Overview

The Advanced Linux Sound Architecture (ALSA) provides audio and MIDI functionality to the Linux operating system. ALSA has the following significant features:

- Efficient support for all types of audio interfaces, from consumer sound cards to professional multichannel audio interfaces.
- Fully modularized sound drivers.

- SMP and thread-safe design.
- User space library (alsa-lib) to simplify application programming and provide higher level functionality.
- Support for the older Open Sound System (OSS) API, providing binary compatibility for most OSS programs.

MTK Soc Alsa driver provide smart phone pcm interface for User space Hal using.



2 Volume Control

As shown in the following figure, it includes mainly three parts.

- *Audio and voice downlink path, right and left channel audio DACs which produces signal to earphone or other auxiliary output device. Amplifiers in these two blocks are equipped with multiplexers to accept signals from internal audio/voice or external radio sources.*
- *Audio or voice uplink path, which is the interface between microphone (or other auxiliary input device) input*

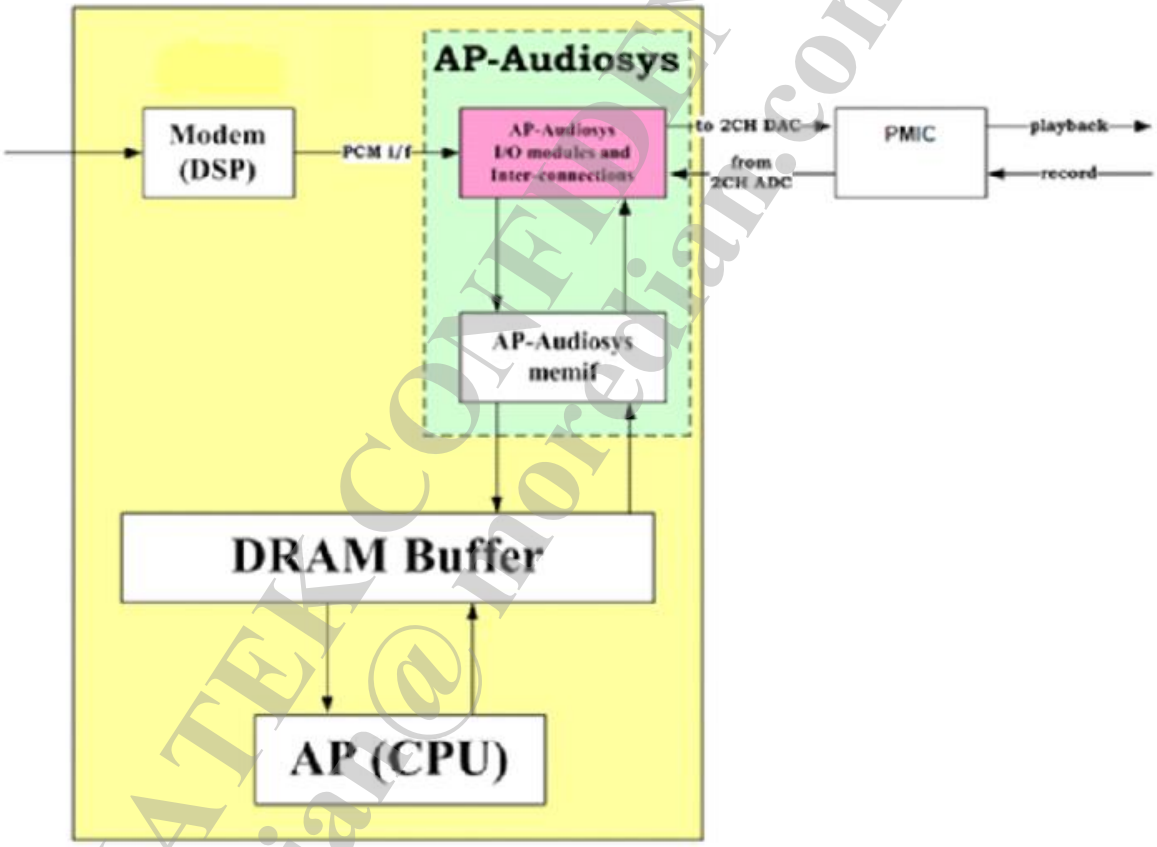


Figure 3. 6757+6351 Diagram

In this chapter, we'll describe the EM setting guide and volume customization guide first. Then the details of volume control and settings will be introduced. We'll also introduce the loudness settings.

2.1 EM Volume Setting Guide

2.1.1 How to Enter EM Mode

In dial pad, press *#*#3646633#*#* and enters EM Mode. Then please goes to Hardware testing page -> Audio. The menu will then be shown as the following figure:

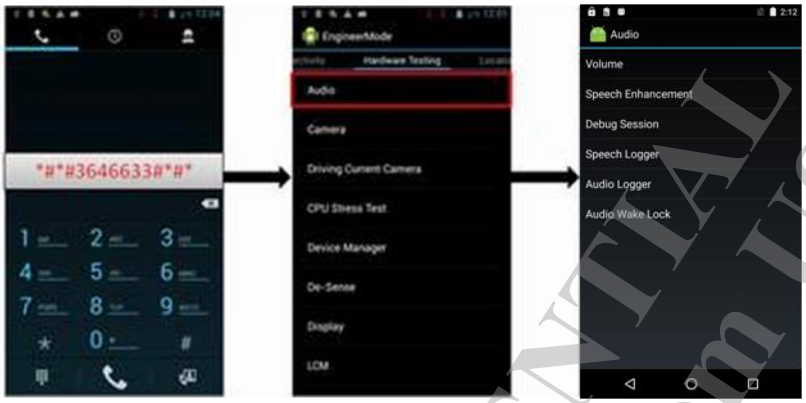


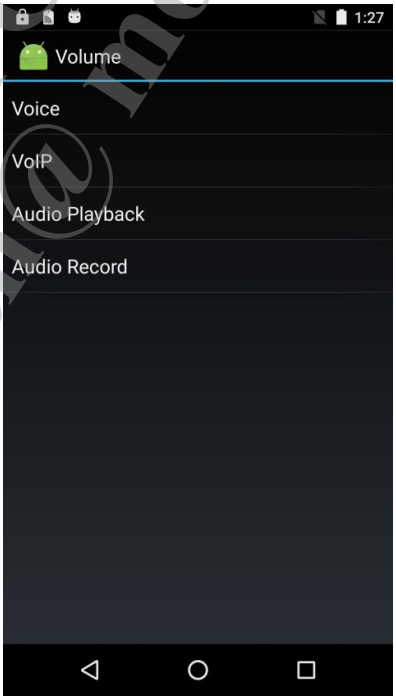
Figure 4. Audio EM Mode Page Menu

2.1.2 Audio EM Pages

2.1.2.1 Overview

The Volume page is used to set the volume for different scenario:

- Voice- Volume setting for in call scenario.
- VoIP- Volume setting for VoIP scenario.
- Audio Playback- Volume setting for audio playback scenario, e.g., music, DTMF, etc.
- Audio Record- Volume setting for record scenario, e.g., sound record, camera record, etc.



The Speech Logger and Audio Logger are used for debugging:

- Speech Logger - It is used to capture the voice VM log. → reference to FAQ03727

Audio Logger - It is used to capture the audio pcm raw data. → reference to FAQ03727

2.1.2.2 Voice Volume Setting

This page is the volume setting for speech scenario.

Gain Table can set different gain for following combination.

1) Bandwidth – Narrow Band, Wide Band
2) Device – Receiver, Speaker, Headset, etc.
3) Volume Index – downlink gain only

There is 3 gain in this scenario.

1) Downlink Gain (0 ~ 160, 4 per step)
   - Consist of digital gain and analog gain, 1dB per step.
   - Digital gain, set to modem DSP.
   - Analog gain, set to audio hardware.
   - Downlink total gain mapping see next page.
2) Uplink Gain (252 ~ 72, 4 per step)
   - Consist of SWAGC gain and analog gain.

Decimal	252	248	244	…	76	72
dB	45	44	43	…	1	0

3) Sidetone Gain (0 ~ 240, 16 per step)
   - Set to modem DSP

Decimal	240	224	208	…	48	32	16	0
dB	20	18	16	…	-4	-6	-8	-10

To adjust gain,

1) Select Bandwidth
2) Select Device
3) Set UL Gain, Sidetone Gain
4) Set DL Gain for each volume index

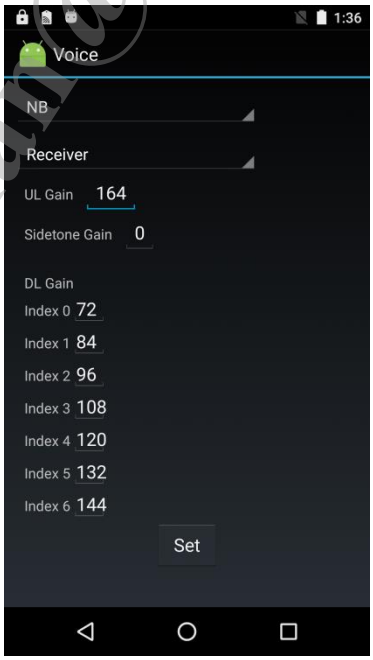


Figure 5. Audio EM Mode - Voice Volume

Downlink Total Gain Mapping

Receiver:

Total gain (dB)	8	7	6	5	4	3	2	1	0	-1
Analog gain (dB)	8	7	6	5	4	3	2	1	0	-1
Digital gain (dB)	0	0	0	0	0	0	0	0	0	0
Decimal	160	156	152	148	144	140	136	132	128	124

-2	-3	-4	-5	-6	-7	-8	-9	-10	-11	-12	-13
-2	-3	-4	-5	-6	-7	-8	-9	-10	-10	-10	-10
0	0	0	0	0	0	0	0	0	-1	-2	-3
120	116	112	108	104	100	96	92	88	84	80	76
-14	-15	-16	-17	-18	-19	-20	-21	-22	-23	-24	-25
-10	-10	-10	-10	-10	-10	-10	-10	-10	-10	-10	-10
-4	-5	-6	-7	-8	-9	-10	-11	-12	-13	-14	-15
72	68	64	60	56	52	48	44	40	36	32	28
-26	-27	-28	-29	-30	-31	-32					
-10	-10	-10	-10	-10	-10	-10					
-16	-17	-18	-19	-20	-21	-22					
24	20	16	12	8	4	0					

Headset:

Total gain (dB)	2	1	0	-1	-2	-3	-4	-5	-6	-7
Analog gain (dB)	2	2	2	2	2	2	2	2	2	2

Digital gain (dB)	0	-1	-2	-3	-4	-5	-6	-7	-8	-9
Decimal	160	156	152	148	144	140	136	132	128	124
-8	-9	-10	-11	-12	-13	-14	-15	-16	-17	-18	-19
2	2	2	2	2	2	2	2	2	2	2	2
-10	-11	-12	-13	-14	-15	-16	-17	-18	-19	-20	-21
120	116	112	108	104	100	96	92	88	84	80	76
-20	-21	-22	-23	-24	-25	-26	-27	-28	-29	-30	-31
2	2	2	2	2	2	2	2	2	2	2	2
-22	-23	-24	-25	-26	-27	-28	-29	-30	-31	-32	-33
72	68	64	60	56	52	48	44	40	36	32	28
-32	-33	-34	-35	-36	-37	-38					
2	2	2	2	2	2	2					
-34	-35	-36	-37	-38	-39	-40					
24	20	16	12	8	4	0					

If your speaker path use lineout buffer.

Speaker (lineout buffer):

Total gain (dB)	-1	-2	-3	-4	-5	-6	-7	-8	-9	-10
Analog gain (dB)	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1
Digital gain (dB)	0	-1	-2	-3	-4	-5	-6	-7	-8	-9
Decimal	160	156	152	148	144	140	136	132	128	124
-11	-12	-13	-14	-15	-16	-17	-18	-19	-20	-21	-22
-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1

-10	-11	-12	-13	-14	-15	-16	-17	-18	-19	-20	-21
120	116	112	108	104	100	96	92	88	84	80	76
-23	-24	-25	-26	-27	-28	-29	-30	-31	-32	-33	-34
-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1
-22	-23	-24	-25	-26	-27	-28	-29	-30	-31	-32	-33
72	68	64	60	56	52	48	44	40	36	32	28
-35	-36	-37	-38	-39	-40	-41					
-1	-1	-1	-1	-1	-1	-1					
-34	-35	-36	-37	-38	-39	-40					
24	20	16	12	8	4	0					

2.1.2.3 VOIP Volume Setting

This page is the volume setting for VoIP scenario.

Gain Table can set different gain for following combination.

1) Scene– Default, Application1, Application2, etc
2) Device – Receiver, Speaker, Headset, etc.
3) Volume Index – downlink gain only

There is 2 gain in this scenario.

1) Downlink Gain (0 ~ 255, 4 per step)
   - Consist of digital gain and analog gain, 1dB per step.
   - Digital gain, set to playback stream.
   - Analog gain, set to audio hardware.
   - Downlink total gain mapping is same as Voice section.
2) Uplink Gain (252 ~ 72, 4 per step)
   - analog gain.

Figure 6. Audio EM Mode - VoIP Volume

2.1.2.4 Audio Playback Volume Setting

This page is the volume setting for Audio Playback scenario.

Gain Table can set different gain for following combination.

1) Scene – Default, Application1, Application2, etc
2) Audio stream type - SYSTEM, RING, MUSIC, etc.
3) Device – Receiver, Speaker, Headset, etc.

4) Volume Index – downlink only.

Downlink Gain is consist of digital gain and analog gain,

1) Digital Gain (255 ~ 0, 4 per step)
   - Consist of digital gain and analog gain, 1dB per step.
   - Digital gain, set to playback stream.
     - Each stream type have independent digital gain.
   - Analog gain, set to audio hardware.
     - Ring, Alarm share same analog gain.
     - Notification has its analog gain.
     - Other stream share same analog gain.

Decimal	255	252	248	244	...	8	4	0
dB	0	-1	-2	-3		-62	-63	-64

2) Analog Gain (160 ~ 88, 4 per step)
   - analog gain.
   - Audio buffer / voice buffer / lineout buffer:

Decimal	160	156	152	...	128	...	92	88
dB	8	7	6	...	0	...	-9	-10

   - Speaker Amp:

Decimal	180	176	172	...	148	...	132	128	112
dB	17	16	15	...	9	...	5	4	0

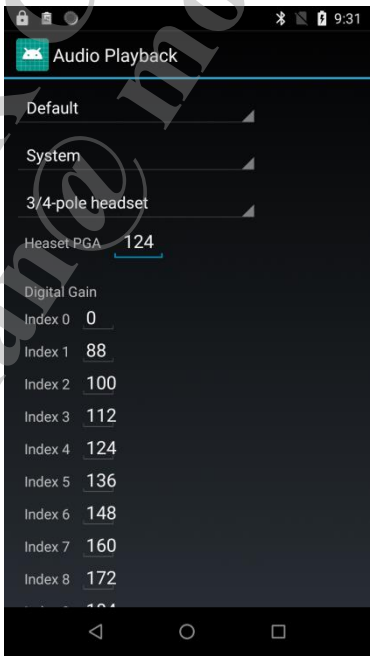


Figure 7. Audio EM Mode - Audio Playback Volume

2.1.2.5 Audio Record Volume Setting

This page is the volume setting for Audio Record scenario.

Gain Table can set different gain for following combination.

1) Scene
   - Default
   - Application1
   - Application2
2) Application
   - Sound Recording
   - Camera Recording
   - ASR improvement
   - Voice recognition
   - Voice Unlock
   - Fast Record
3) Device – Receiver, Speaker, Headset, etc.
   - Handest (phone mic)
   - Headset (headset mic)

There is 1 gain in this scenario.

1) Uplink Gain (252 ~ 72, 4 per step)
   - analog gain.

Decimal	252	248	244	...	76	72
dB	45	44	43	...	1	0

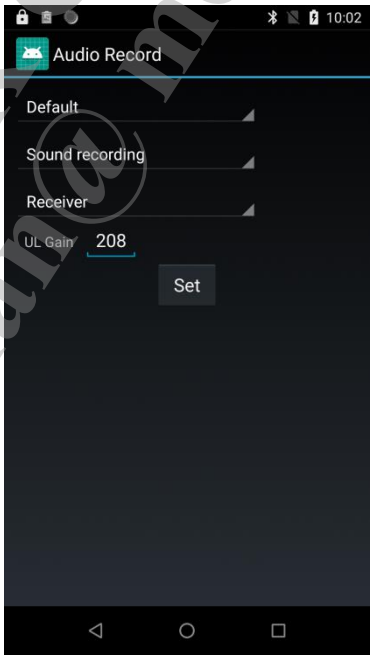


Figure 8. Audio EM Mode -Audio Record Volume

2.1.2.6 Speech Enhancement

1) Choose parameters or mode
2) Choose parameters
3) Setting value

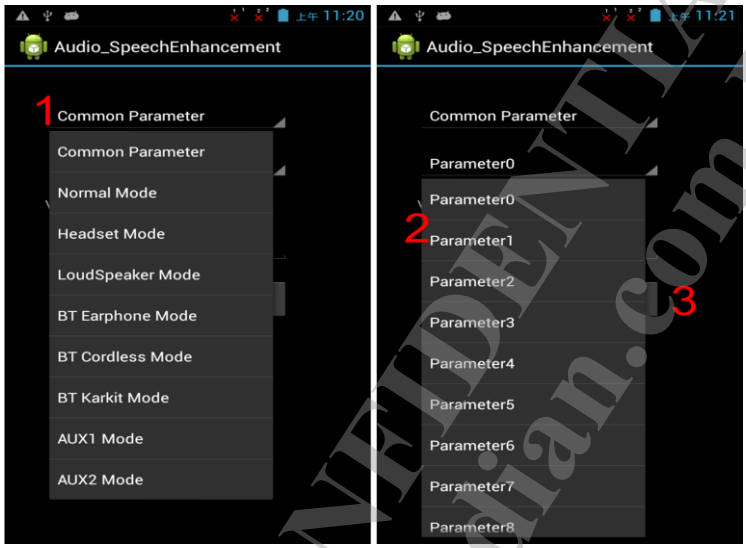


Figure 9. Audio EM Mode – Speech Enhancement I

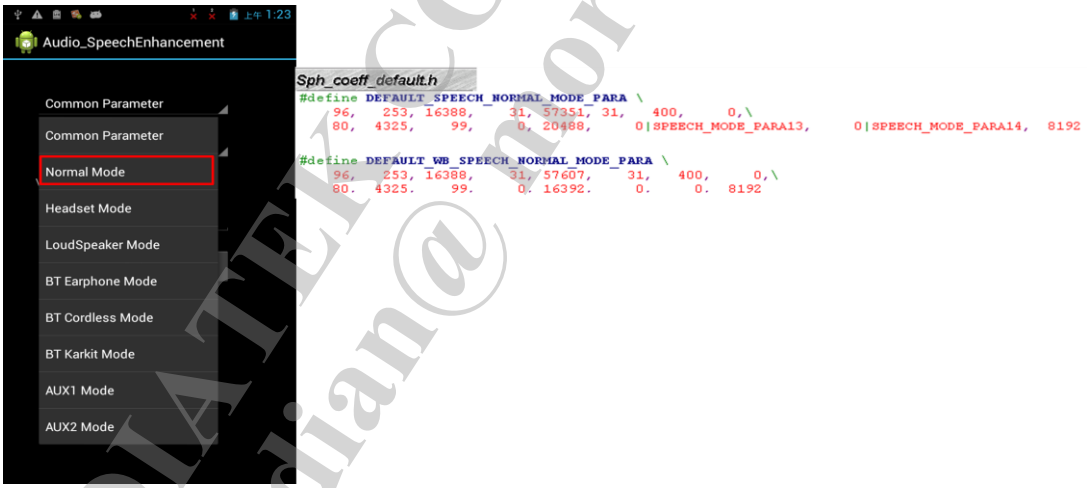


Figure 10. Audio EM Mode – Speech Enhancement II

2.1.2.7 Debug Info

Choose debug parameters and setting value

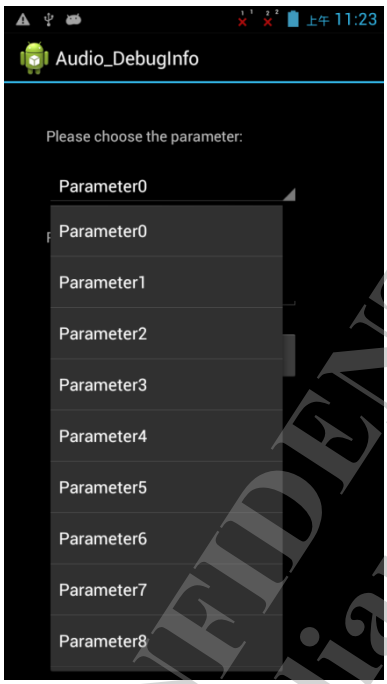


Figure 11. Audio EM Mode - Debug Info

2.1.2.8 Speech Logger

Select required debug information directly. For example, if we need vm log, just enable speech log. If we need the speech debug parameters, just press dump speech debug info.

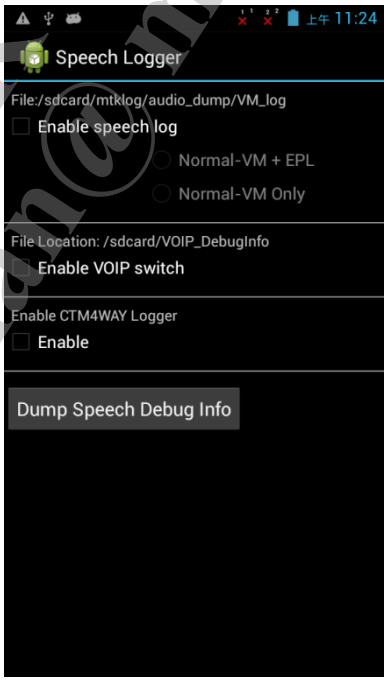


Figure 12. Audio EM Mode - Speech Logger

2.1.2.9 Audio Logger

- Dump PCM stream to storage
  - Audio Stream Output Dump

  To dump pcm data write to audio hardware
  - AudioMixer Buffer Dump

  Dump for Audio flinger mixing data
  - Audio Track Buffer Dump

  Dump for AudioTrack data
  - A2DP Stream Output Dump

  Dump for data write to A2DP
  - Audio Stream Input dump

  Dump for streamin manager and streamin pcm data
- Dump Audio Debug Info
  - Print all audio register and generate sine wave to Downlink
  - When we generate sine wave, we need to do normal playback or analog part need turn on.

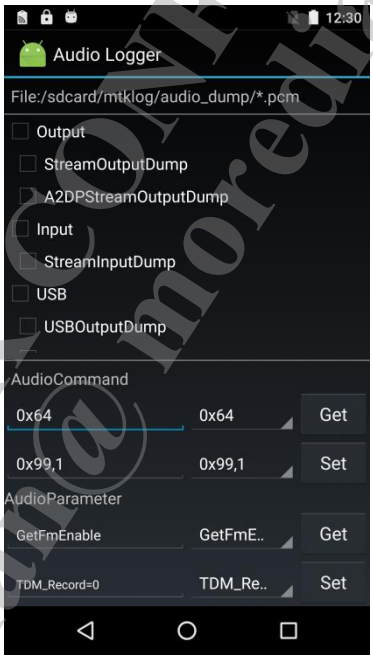


Figure 13. Audio EM Mode - Audio Logger

2.1.3 Volume Level

Maximum Stream Volume Level

Each Stream Type has a maximum volume level, it is defined in

alps/ frameworks/base/services/core/java/com/android/server/audio/AudioService.java

" private static int[] MAX_STREAM_VOLUME = new int[] {

7, // STREAM_VOICE_CALL

15, // STREAM_SYSTEM

15, // STREAM_RING

15, // STREAM_MUSIC
15, // STREAM_ALARM
15, // STREAM_NOTIFICATION
15, // STREAM_BLUETOOTH_SCO
15, // STREAM_SYSTEM_ENFORCED
15, // STREAM_DTMF
15 // STREAM_TTS
}; "

Default Stream Volume Level

Each Stream Type has a default volume level, it is defined in alps/frameworks/base/media/java/android/media/AudioSystem.java

" public static int[] DEFAULT_STREAM_VOLUME = new int[] {
4, // STREAM_VOICE_CALL
15, // STREAM_SYSTEM
8, // STREAM_RING
8, // STREAM_MUSIC
8, // STREAM_ALARM
8, // STREAM_NOTIFICATION
7, // STREAM_BLUETOOTH_SCO
15, // STREAM_SYSTEM_ENFORCED
11, // STREAM_DTMF
11 // STREAM_TTS
}; "

2.1.4 Scene Selection

It's reserved for future expansion.

2.1.5 Power On/Power Off Sound

There are three methods to adjust power on/off volume

- **Method 1**

AudioPolicyManager.cpp:

  - address: alps/frameworks/av/services/audiopolicy/managerdefault
  - Modify the value: #define BOOT_ANIMATION_VOLUME(0.25); the max number allowed is 1

- **Method 2**

BootAnimation.cpp:

  - address: alps/frameworks/base/cmds/bootanimation

- In BootAnimation::threadLoop() function, add mediaplayer->setVolume(leftVolume,rightVolume) before mediastatus = mediaplayer->start().
- Notice that the range is from 0 to 1 and the default value is 1.

➢ **Method 3**

Just replace the original power on/off files to the ones with proper volume

2.2 Gain Configurations and Mappings in Detail

2.2.1 Gain Table Overview

Gain Table is a volume control, a table storing the gain for each scenario.

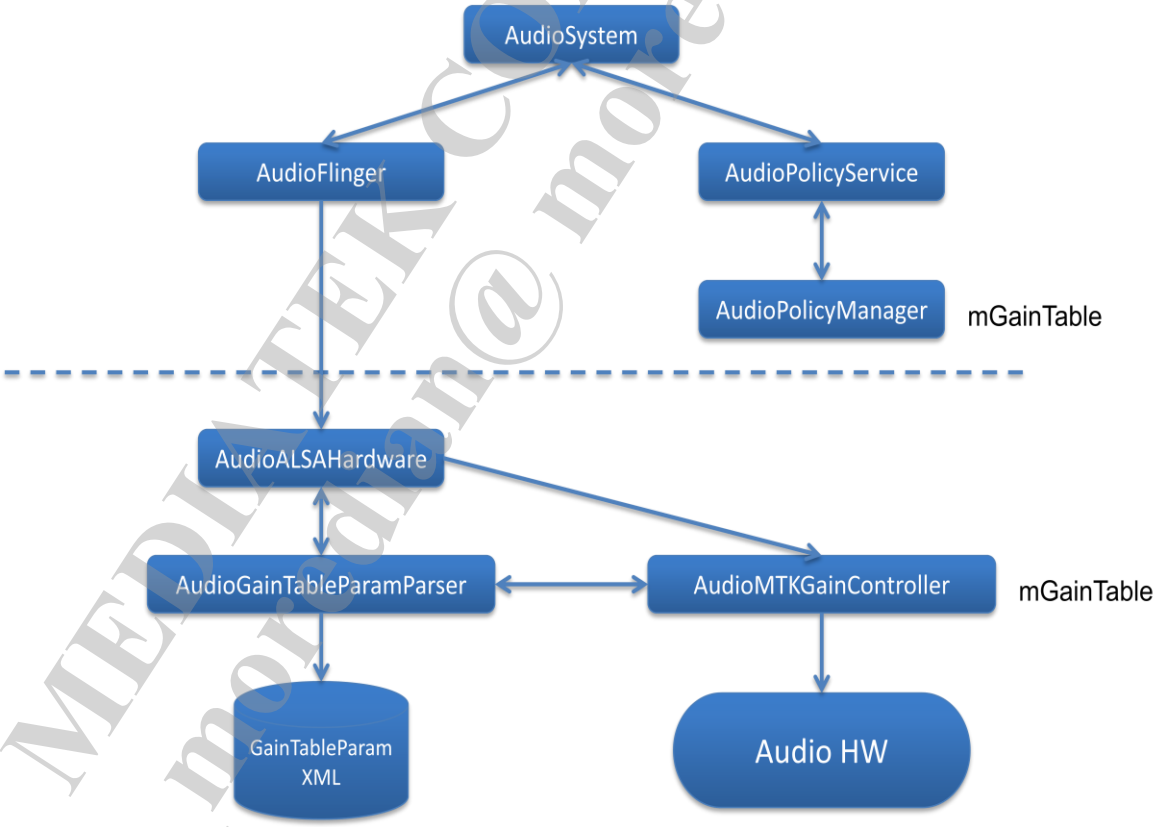
- Music, Speaker, Volume Index 15.
- Phone call, Receiver, Volume Index 6.

Gain Table parameters are **stored in XML.**

Parameters are extracted to audio driver by AudioGainTableParamParser.

AudioPolicyManager will calculate the stream digital gain.

AudioMTKGainController will assign the corresponding hardware PGA gain and the gain in speech gain.



2.2.2 Gain Table Related Files

Audio driver file,

File Name	Path	Description
AudioALSAGainController.h	*vendor\mediatek\proprietary\hardware\audio\common\V3*	*Main Gain Table Controller, to set gain for playback, record, and in call.*
AudioALSAGainController.cpp		
AudioGainTableParam.h		*Gain Table related structure / enum / define.*
AudioGainTableParamParser.h		*Parse Gain Table parameters from xml.* *APIs to let audio driver to get the parameters.*
AudioGainTableParamParser.cpp		

Gain Table Parameters are stored in XML

XML in code base,

- Common: device\mediatek\common\audio_param\
- Platform: device\mediatek\$(PLATFORM)\audio_param (if needed)
  - Ex. device\mediatek\mt6771\audio_param
- Project: device\mediatek\$(PROJECT)\audio_param (if needed)
  - Ex. device\mediatek\k71v1_64_bsp\audio_param
- Project XML > Platform XML > Common XML
  - Project XML will override Platform XML, while Platform XML will override Common XML.

XML on device,

- Default xml: system/etc/audio_param/
- Customized xml (created by tuning tool): sdcard/.audio_param/

File Name	Description
PlaybackVolAna	Hardware PGA Gain for different stream and device.
PlaybackVolDigi	Stream digital gain for different stream and device.
PlaybackVolUI	Volume related visibility in tuning tool Playback page.
RecordVol	Hardware PGA Gain for different recording scenario.
RecordVolUI	Volume related visibility in tuning tool Record page.
SpeechVol	Gain setting for in call scenario.
SpeechVolUI	Volume related visibility in tuning tool Speech page.
VoIPVol	Gain setting for VoIP scenario.
VoIPVolUI	Volume related visibility in tuning tool VoIP page.

RingbackVol	Gain setting for ringback tone
Volume	Common parameters used in tuning tool and audio driver
VolumeGainMap	Total gain mapping to digital and analog gain for Speech and VoIP

2.2.3 Gain Table Parameters Parsing

Gain Table Parameters Structure:

```
struct GainTableUnit {
    unsigned char digital;
    unsigned char analog[NUM_GAIN_ANA_TYPE];
};

struct GainTableSidetoneUnit {
    unsigned char gain;
};

struct GainTableMicUnit {
    unsigned char gain;
};

struct GainTableRingbackToneUnit {
    unsigned char digital;
};

struct GainTableForScene {
    GainTableUnit streamGain[GAIN_STREAM_TYPE_SIZE][NUM_GAIN_DEVICE][GAIN_VOL_INDEX_SIZE];
    GainTableMicUnit micGain[NUM_GAIN_MIC_MODE][NUM_GAIN_DEVICE];
};

struct GainTableForNonScene{
    GainTableUnit speechGain[NUM_GAIN_SPEECH_BAND][NUM_GAIN_SPEECH_NETWORK][NUM_GAIN_DEVICE][GAIN_VOL_INDEX_SIZE];
    GainTableSidetoneUnit sidetoneGain[NUM_GAIN_SPEECH_BAND][NUM_GAIN_SPEECH_NETWORK][NUM_GAIN_DEVICE];
    GainTableMicUnit speechMicGain[NUM_GAIN_SPEECH_BAND][NUM_GAIN_SPEECH_NETWORK][NUM_GAIN_DEVICE];
    GainTableRingbackToneUnit ringbackToneGain[NUM_GAIN_DEVICE][GAIN_VOL_INDEX_SIZE];
};

struct GainTableParam {
    int sceneCount;
    GainTableForScene *sceneGain;
    GainTableForNonScene nonSceneGain;
};
```

GainTableParamParser will parse xml data into GainTableParam:

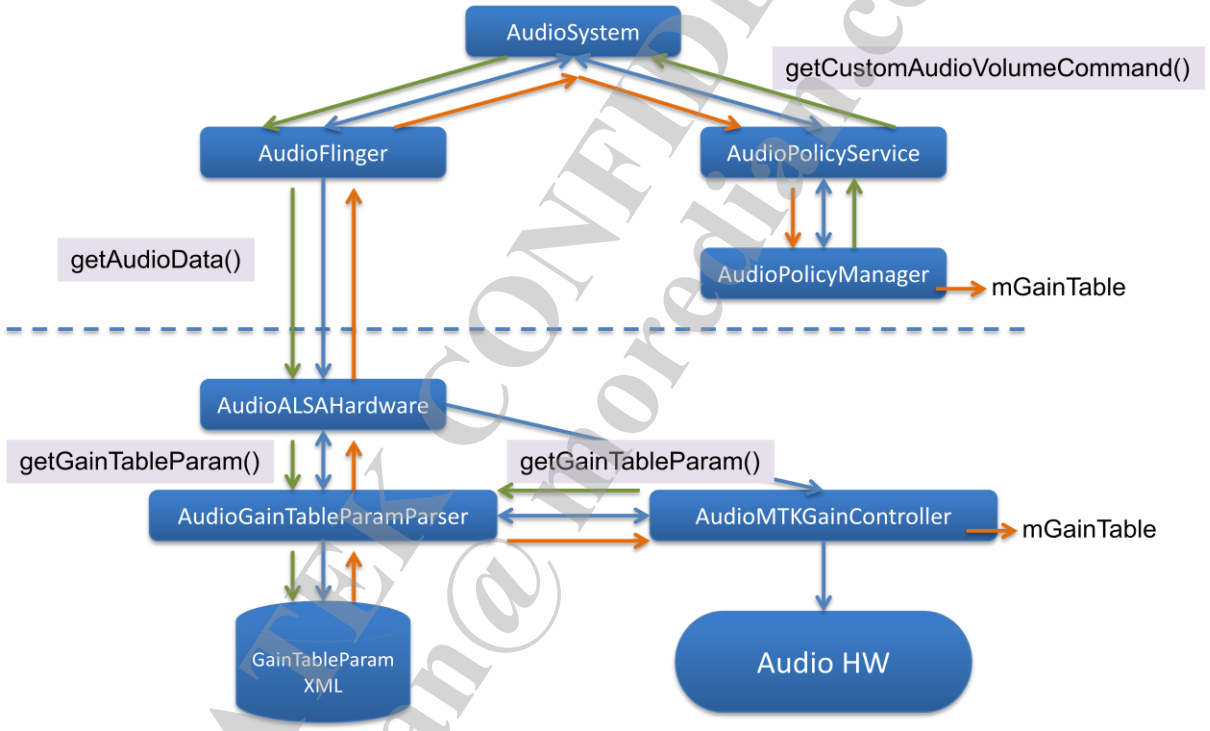
```
status_t GainTableParamParser::getGainTableParam(GainTableParam * _gainTable, std::vector<std::string> *sceneList) {
    ALOGD("%s()", __FUNCTION__);
    clearTableParam(_gainTable, sceneList->size());

    status_t status = NO_ERROR;
    _gainTable->sceneCount = (int)(sceneList->size());
    status |= updatePlaybackDigitalGain(_gainTable, sceneList);
    status |= updatePlaybackAnalogGain(_gainTable, sceneList);
    status |= updateSpeechVol(_gainTable);
    status |= updateRecordVol(_gainTable, sceneList);
    status |= updateVoIPVol(_gainTable, sceneList);
    status |= updateRingbackVol(_gainTable);
```

Two instance of GainTableParam is allocate in,

- AudioPolicyManager
  - Calculate the stream digital gain for playback.
- AudioMTKGainController
  - Playback – assign DL hardware PGA gain
  - Record – assign UL hardware PGA gain and digital gain.
  - Speech downlink– assign DL hardware PGA gain and digital gain to modem DSP
  - Speech uplink – assign UL hardware PGA gain and digital gain to modem DSP
  - Speech sidetone – assign sidtone gain to modem DSP.

Get GainTableParam path:



2.2.4 Gain Table – Playback Scenario

2.2.4.1 Gain Table – Playback Scenario – Digital Gain

Gain Table can set different digital gain for following combination.

- Scene
  - e.g., Default, Application1, Application2, etc.

```
<CategoryType name="Scene" feature_option="VIR_SCENE_CUSTOMIZATION_SUPPORT">
  <Category name="Default"/>
  <Category name="App1"/>
  <Category name="App2"/>
</CategoryType>
```

  In PlaybackVolDigi_ParamUnitDesc.xml

- Audio stream type
  - e.g., SYSTEM, RING, MUSIC, etc.

```
<CategoryType name="Volume type">
  <Category name="System"/>
  <Category name="Ring"/>
  <Category name="Music"/>
  <Category name="Alarm"/>
  <Category name="Notification"/>
  <Category name="Bluetooth_sco"/>
  <Category name="Enforced_Audible"/>
  <Category name="DTMF"/>
  <Category name="TTS"/>
  <Category name="Boot"/>
  <Category name="VIBSPK"/>
  <Category name="Accessibility"/>
```

  In PlaybackVolDigi_ParamUnitDesc.xml

- Primary device
  - e.g., Speaker, Earpiece, Headset, Headphone.

```
<CategoryType name="Profile">
  <Category name="RCV"/>
  <Category name="HS" alias="Headset,3/4-pole Headset,HP"/>
  <Category name="SPK" alias="Speaker"/>
  <Category name="HSSPK" alias="Headset+Speaker"/>
  <Category name="HS5POLE" alias="5-pole Headset"/>
  <Category name="HS5POLE_ANC" alias="5-pole headset+ANC"/>
  <Category name="USB"/>
</CategoryType>
```

  In PlaybackVolDigi_ParamUnitDesc.xml

- Volume index

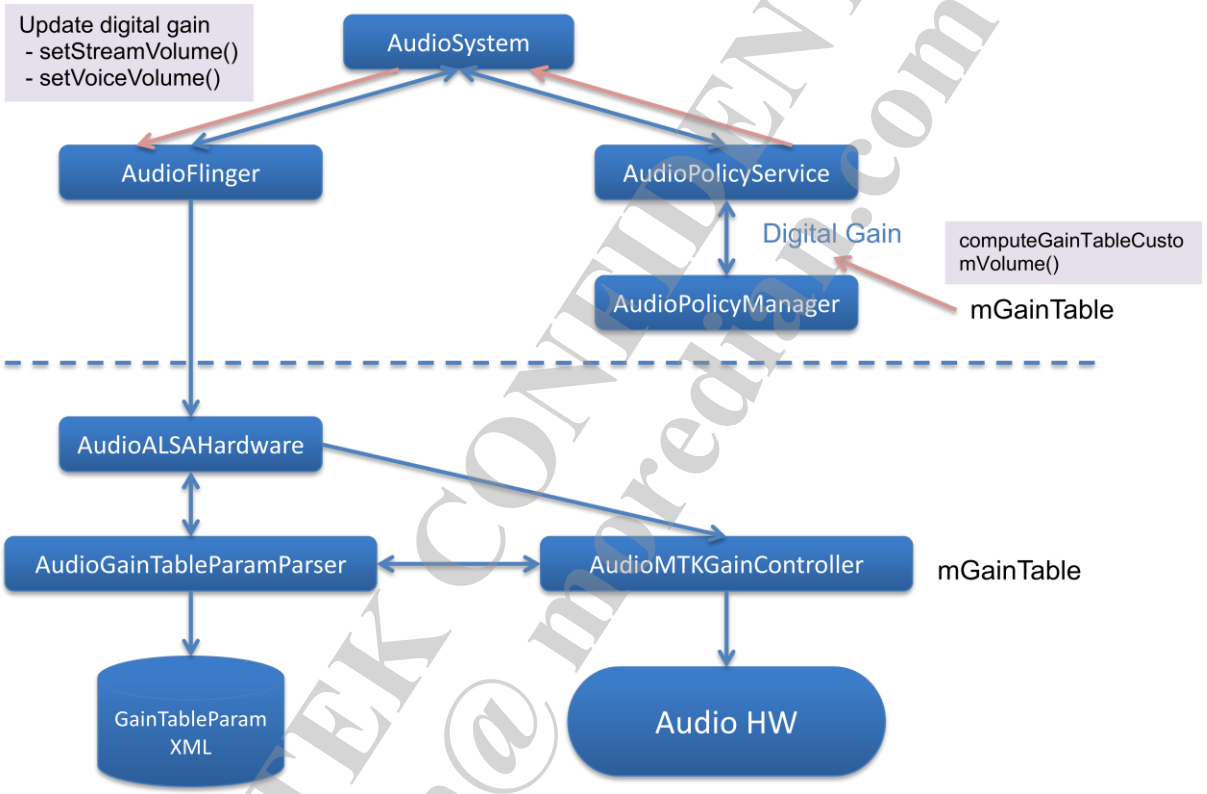
```
<ParamUnit param_id="0">
  <Param name="digital_gain" value="-64,-42,-39,-36,-33,-30,-27,-24,-21,-18,-15,-12,-9,-6,-3,0"/>
</ParamUnit>
```

  In PlaybackVolDigi_AudioParam.xml
  - The values are the digital gain (dB) for volume in dex 0 ~ 15
  - Digital gain is stored in streamGain:

```
struct GainTableForScene {
    GainTableUnit streamGain[GAIN_STREAM_TYPE_SIZE][NUM_GAIN_DEVICE][GAIN_VOL_INDEX_SIZE];
    GainTableMicUnit micGain[NUM_GAIN_MIC_MODE][NUM_GAIN_DEVICE];
};
```

AudioPolicyManager extract digital gain and apply to stream volume.

- AudioPolicyManager::computeGainTableCustomVolume(int stream, int index, audio_devices_t device)



2.2.4.2 Gain Table – Playback Scenario – Analog Gain

Gain Table can set different analog gain for following combination

- Scene
- Audio stream type
  - Ring, Alarm have same analog gain
  - Notification
  - Other stream types have same analog gain
  - PlaybackVolAna_ParamUnitDesc.xml

```
<CategoryType name="Volume type">
  <Category name="Ring_Alarm" alias="Ring,Alarm"/>
  <Category name="Notification"/>
  <Category name="Others" alias="System,Music,Bluetooth_sco,Enforced_Audible,DTMF,TTS,Boot,VIBSPK,Accessibility"/>
```

- Primary device

Same analog gain for all volume index.

The value is stored in register index.

```
<ParamUnit param_id="1">
  <Param name="headset_pga" value="12"/>
  <Param name="speaker_pga" value="10"/>
  <Param name="receiver_pga" value="-1"/>
```

PlaybackVolAna_AudioParam.xml

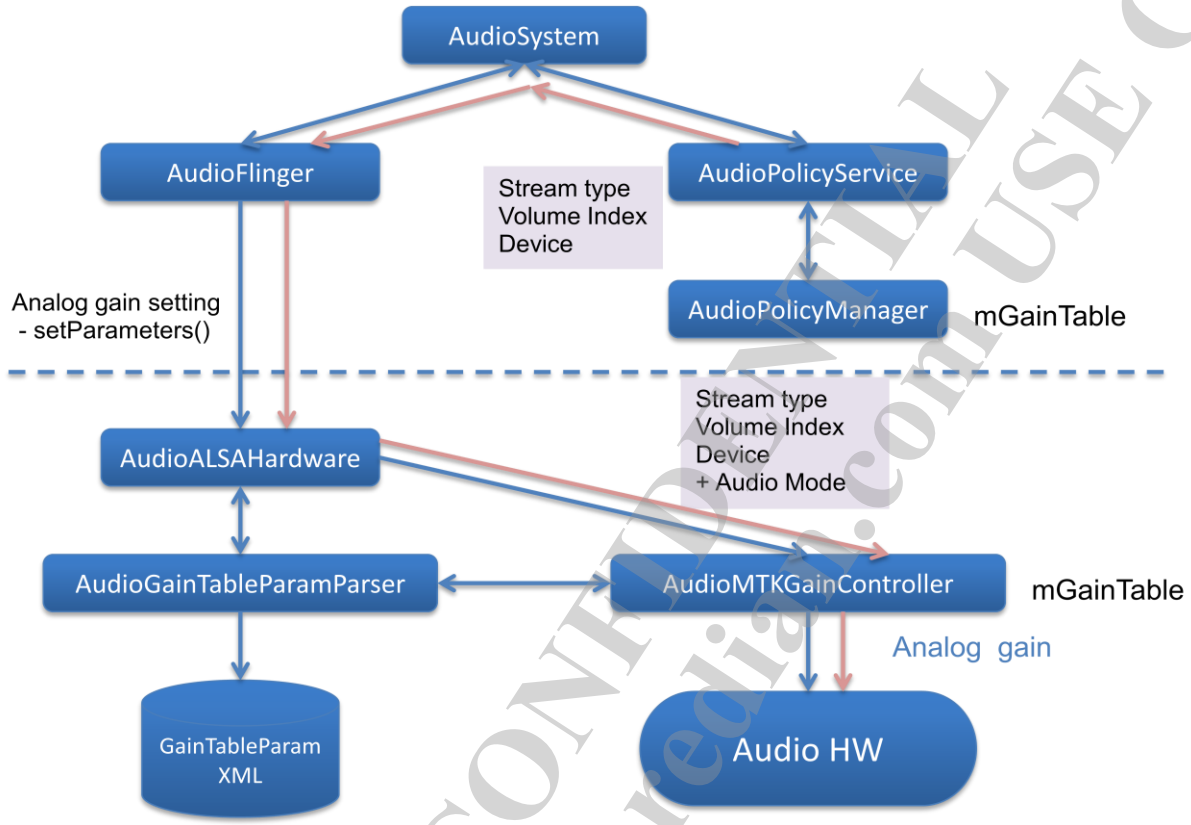
Corresponding dB can be seen in:

```
<Param name="headset_pga" type="short_array"> <!-- save index of register -->
  <Field name="hs_ana_gain" array_index="0" bit="0,15" check_list="0,8dB,1,7dB,2,6dB,3,5dB,4,4dB,5,3dB,6,2dB,7,1dB,8,0dB,9,-1dB,10,-2dB,11,-3dB,12,-4dB,13,-5dB,14,-6dB,15,-
</Param>
<Param name="speaker_pga" type="short_array"> <!-- save index of register -->
  <Field name="spk_ana_gain" array_index="0" bit="0,15" check_list="15,17dB,14,16dB,13,15dB,12,14dB,11,13dB,10,12dB,9,11dB,8,10dB,7,9dB,6,8dB,5,7dB,4,6dB,3,5dB,2,4dB"/>
</Param>
```

PlaybackVolAna_ParamUnitDesc.xml

AudioPolicyManager will set these info to HAL,

- In AudioPolicyManager::checkAndSetGainTableAnalogGain(),
- Stream Type
  - Note, all the stream types share the same analog gain, only one analog gain will be set to hardware. The stream type priority can be seen in AudioPolicyManager::selectGainTableActiveStream().
- Device
- Volume Index



Playback Scenario call flow,

- PolicyManager::checkAndSetVolume()
- PolicyManager::computeVolume()
  - Retrieve digital gain from gain table
- mpClientInterface->setStreamVolume()
- PolicyManager::checkAndSetGainTableAnalogGain(stream, idx, device)
- ↓ stream, idx, device to HAL
- StreamManager::setAnalogVolume(stream, idx, device)
- GainController::setAnalogVolume(stream, idx, device, audio_mode)
- ↓not in phone call
- AudioMTKGainController::setNormalVolume(stream, idx, device, mode)
- ↓ set analog gain

2.2.4.3 Gain Table – Playback Scenario – Master Volume

The master volume is used in tuneGainForMasterVolume(), Gain Table adjust analog gain accordingly.

Note: The master volume range is 0~1.0. But analog gain is 1dB per step, and has limited range.

There is some limitations in here.

2.2.4.4 Gain Table – Playback Scenario – HP Impedance Compensation

HP impedance may varies among different headphones, the volume decrease when the impedance increase and vice versa. To make volume the same, Gain Table compensate this in AudioMTKGainController::tuneGainForHpImpedance();

You can customized the on board resistor, threshold, and degrade gain list, change the value or number of threshold.

```
<Param name="hp_impedance_enable" value="1"/>
<Param name="hp_impedance_onboard_resistor" value="0"/>
<Param name="hp_impedance_default_idx" value="1"/>
<Param name="hp_impedance_threshold_list" value="24,48,96,192"/>
<Param name="hp_impedance_gain_degrade_list" value="3,0,-3,-6,-9"/>
```

Impedance Range (Ω)	0 ~ 24	25 ~ 48	49 ~ 96	97 ~ 192	193 ~
Compesate Gain (dB)	-3	+0	+3	+6	+9

2.2.5 Gain Table – Record Scenario

Gain Table can set different uplink gain for following combination.

- Scene
- Application
  - Sound Recording
  - Camera Recording
  - ASR improvement
  - Voice recognition
  - Voice Unlock
  - Fast Record
- Device
  - Handset (phone mic)
  - Headset (headset mic)

The range parameters is in Volume_AudioParam.xml.

Uplink gain range is 0 ~ 45 dB

```
<Param name="mic_idx_range_max" value="45"/>
<Param name="mic_idx_range_min" value="0"/>
```

Decimal Range is 252 ~ 72

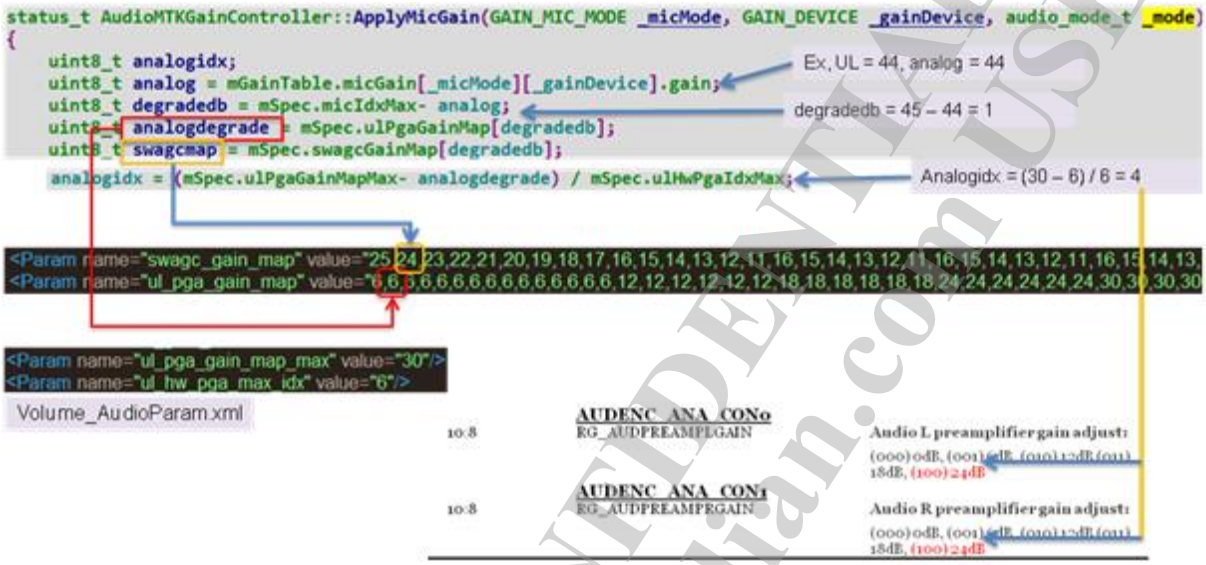
```
<Param name="dec_rec_max" value="252"/>
<Param name="dec_rec_step_per_db" value="4"/>
```

Decimal	252	248	244	...	76	72

dB	45	44	43	…	1	0

UL gain (0~45) is mapped to SWAGC gain and analog gain.

Note: SWAGC is used in phone call only.



2.2.6 Gain Table – VoIP Scenario

Gain Table can set different gain for following combination.

- Bandwidth – Narrow Band, Wide Band
- Device – Receiver, Speaker, Headset, etc.
- Volume Index
  - Downlink Gain only.

There is 3 gain in speech scenario.

- Downlink Gain
  - Consist of digital gain and analog gain, 1dB per step.
  - Digital gain, set to modem DSP.
  - Analog gain, set to audio hardware.
- Uplink Gain
  - Consist of SWAGC gain and analog gain.
- Sidetone Gain
  - Set to modem DSP

Speech Gain Control, call flow:

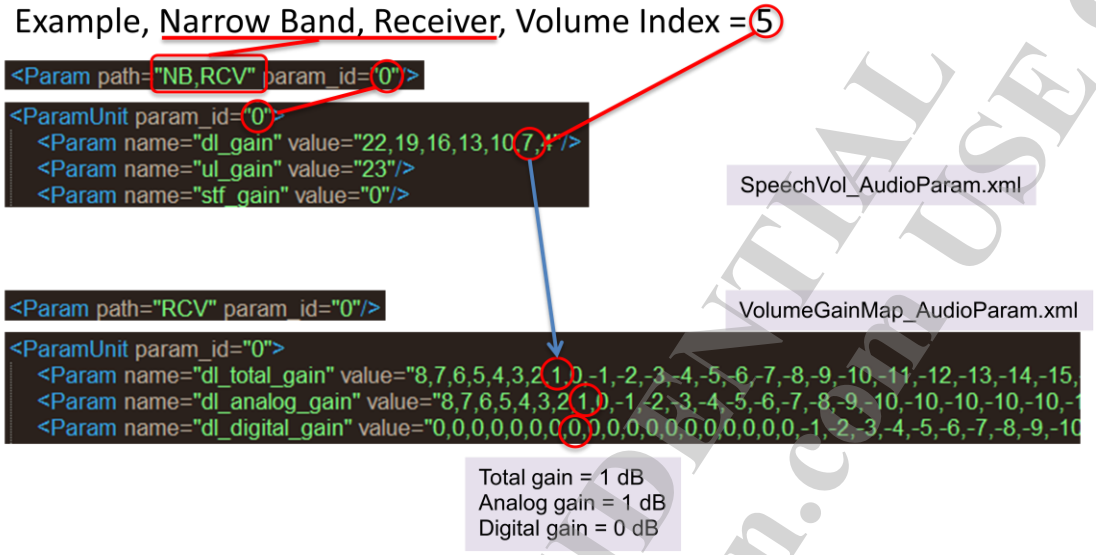
- PolicyManager::checkAndSetGainTableAnalogGain(stream, idx, device)
- ↓ stream, idx, device to HAL
- StreamManager::setAnalogVolume(stream, idx, device)
- GainController::setAnalogVolume(stream, idx, device, audio_mode)
- ↓in phone call
- AudioMTKGainController::setVoiceVolume(idx, device, mode)
  - set analog gain (speaker/headset/receiver)
  - ApplyMdDlGain() – set digital gain
  - ApplyMicGainByDevice() – set uplink gain

ApplySideTone() – set sidetone gain.

2.2.6.1 Gain Table – Speech Scenario – Gain Type

Downlink Gain:

- Example, Narrow Band, Receiver, Volume Index = 5



Uplink Gain

- Uplink gain consist of SWAGC and analog gain.
- Please Reference “Gain Table – Record”

Sidetone Gain

- Sidetone gain range is -10 ~ 20 dB

SpeechVol_ParamUnitDesc.xml

```
<Param name="stf_gain" type="short_array">
  <Field name="stf_gain_field" array_index="0" bit="0,15" check_list="30,20dB,28,18dB,26,16dB,24,14dB,22,12dB,20,10dB,
                                                                      18,8dB,16,6dB,14,4dB,12,2dB,10,0dB,8,-2dB,6,-4dB,
                                                                      4,-6dB,2,-8dB,0,-10dB"/>
```

- Decimal Range is 252 ~ 72
  - <Param name=" dec_stf_max " value=“240"/>
  - <Param name=" dec_stf_step_per_db " value=“8"/>

Decimal	240	224	208	…	48	32	16	0
dB	20	18	16	…	-4	-6	-8	-10

2.2.7 Gain Table – Speech Scenario – Gain Type

Gain Table set different gain for different Device

There is 2 gain in VoIP scenario.

- Downlink Gain
  - Consist of digital gain and analog gain, 1dB per step.
  - Digital gain, set to playback stream
  - Analog gain
  - Please reference “Gain Table – Speech Downlink Gain”
- Uplink Gain
  - analog gain only
  - Please reference “Gain Table – Record”

2.2.8 Registers for Volume Setting

2.2.8.1 PGA Gain Setting

This chapter description for Audio Hardware capability,

Set this register for Audio downlink analog Lineout PGA gain control

248A **ZCD_CON1** **ZCD control register 1** **0F9F**

Bit	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Name					RG_AUDLORGAIN							RG_AUDLOLGAIN				
Type					RW							RW				
Reset					1	1	1	1	1			1	1	1	1	1

Bit(s)	Name	Description
11:7	RG_AUDLORGAIN	00000: +8dB. 00001: +7dB. 10010: -10dB 11111: -40dB(Mute).
4:0	RG_AUDLOLGAIN	00000: +8dB. 00001: +7dB. 10010: -10dB 11111: -40dB(Mute).

Set this register for Audio downlink analog Headphone PGA gain control

248C **ZCD_CON2** **ZCD control register 2** **0F9F**

Bit	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Name					RG_AUDHPRGAIN							RG_AUDHPLGAIN				
Type					RW							RW				
Reset					1	1	1	1	1			1	1	1	1	1

Bit(s)	Name	Description
11:7	RG_AUDHPRGAIN	00000: +8dB. 00001: +7dB. 10010: -10dB 11111: -40dB(Mute).
4:0	RG_AUDHPLGAIN	00000: +8dB. 00001: +7dB. 10010: -10dB 11111: -40dB(Mute).

For Audio downlink analog Headphone PGA gain. We can set the register **AUDDEC_ANA_CON10[2]** to degrade -12dB.

Set this register for Audio downlink analog Handset PGA gain control

248E **ZCD_CON3** **ZCD control register 3** **001F**

Bit	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Name												RG_AUDHSGAIN				
Type												RW				
Reset												1	1	1	1	1

Bit(s)	Name	Description
4:0	RG_AUDHSGAIN	00000: +8dB. 00001: +7dB. 10010: -10dB 11111: -40dB(Mute).

Set this register for Audio downlink analog Speaker PGA gain control

2.2.9 Volume Setting

2.2.9.1 Introduction of Stream Types

In MTK Android system, stream has is total 14 types. User can make an attenuation of each type, can achieve with different stream has different volume strength.

Stream name	Stream value
AUDIO_STREAM_DEFAULT	-1
AUDIO_STREAM_VOICE_CALL	0
AUDIO_STREAM_SYSTEM	1
AUDIO_STREAM_RING	2
AUDIO_STREAM_MUSIC	3
AUDIO_STREAM_ALARM	4
AUDIO_STREAM_NOTIFICATION	5
AUDIO_STREAM_BLUETOOTH_SCO	6
AUDIO_STREAM_ENFORCED_AUDIBLE	7
AUDIO_STREAM_DTMF	8
AUDIO_STREAM_TTS	9
AUDIO_STREAM_BOOT	10
AUDIO_STREAM_VIBSPK	11
AUDIO_STREAM_ACCESSIBILITY	12
AUDIO_STREAM_REROUTING	13
AUDIO_STREAM_PATCH	14

From android design, all streams have it own stream gain setting in digital domain, can Audio Buffer can apply analog gain to strengthen or degrade signal

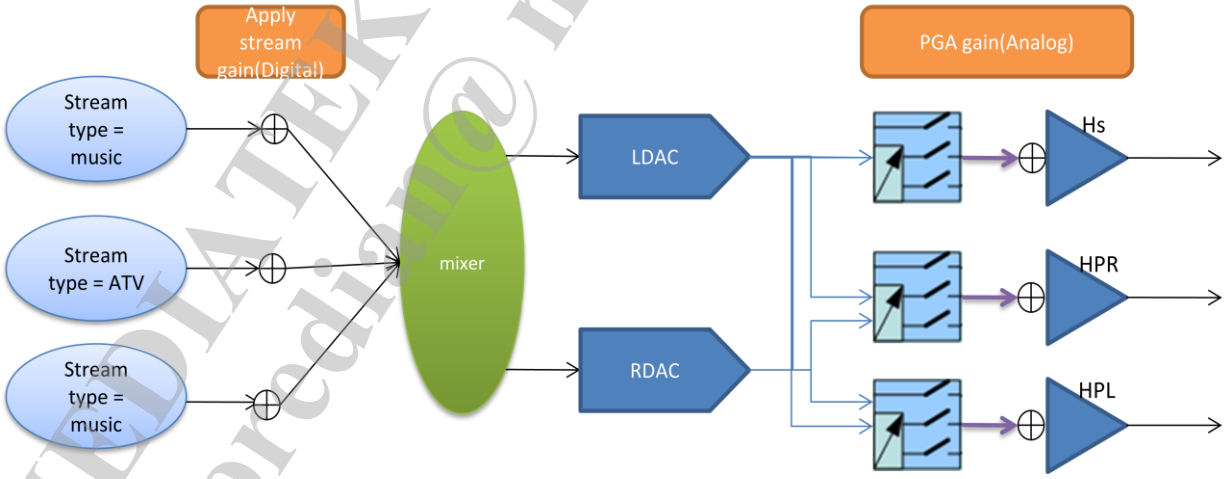


Figure 14. Volume Control Overview

2.2.9.2 Audio Digital Volume Mappings

Here is digital gain mapping table.

dB	Value
0dB	255

-1dB	252
-2dB	248
-3dB	244
	…
	…
	…
-62dB	8
-63dB	4
Mute	0

2.2.10 Speaker Customization

2.2.10.1 PMIC Analog Blocks

6351 has 1 internal amp with left channel support

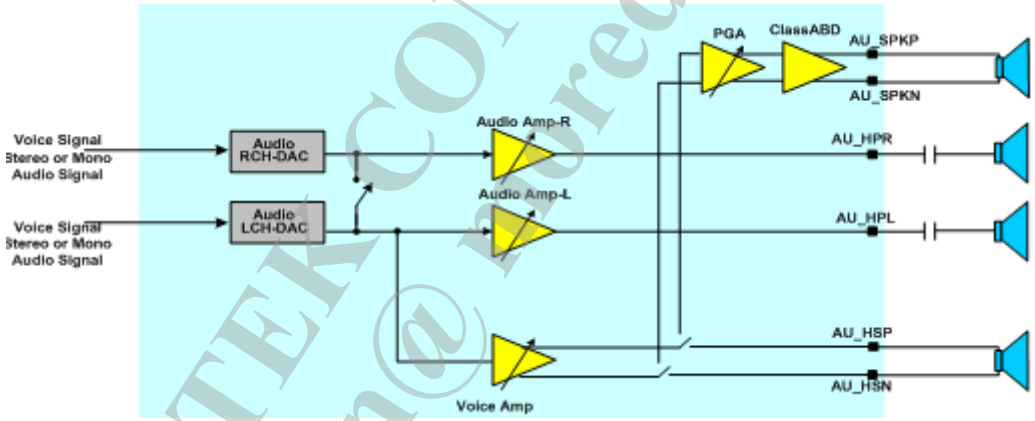


Figure 15. PMIC Analog Blocks

2.2.10.2 Speaker Customization Setting

- Need external speaker amp
  - External amp
    - With lineout buffer
- Kconfig config
  - Config_mtk_speaker=n for external speaker amp
- audio_custom_exp.h
  - #define USING_EXTAMP_LO for solution of external speaker amp connected to Lineout output

Choose CUSTOM_KERNEL_SOUND folder and do speaker implementation

2.2.11 External Analog Device Customization

Use customer alsa codec driver to integration

2.2.12 Terminology

2.2.12.1 Header File

This section will detail descript the meaning of definition in audio_custom_exp.h, below shows where audio_custom_exp.h locates.

audio_Custom_exp.h	*/alps/vendor/mediatek/proprietary/custom/(%proj)/hal/audioflinger/audio* */alps/vendor/mediatek/proprietary/custom/common/hal/audioflinger/audio*	
Audio_Customization_Common.h.	*/alps/vendor/mediatek/proprietary/custom/(%proj)/hal/audioflinger/audio* */alps/vendor/mediatek/proprietary/custom/common/hal/audioflinger/audio*	*Define common audio define*

2.2.12.2 Meaning of Definitions

Name of define	*Description*
DEVICE_MAX_VOLUME	Define maximum unit of volume, unit is dB, this value will depend on audio hardware. Suggest doesn’t adjust it.
DEVICE_MIN_VOLUME	Define minimum unit of volume, unit is dB, this value will depend on audio hardware. Suggest doesn’t adjust it.
DEVICE_VOICE_MAX_VOLUME	Define maximum unit of voice volume, unit is dB, this value will depend on audio hardware. Suggest doesn’t adjust it.
DEVICE_VOICE_MIN_VOLUME	Define minimum unit of voice volume, unit is dB, this value will depend on audio hardware. Suggest doesn’t adjust it.
BOOT_ANIMATION_VOLUME	Adjust boot animation volume. the volume range is from 0 to 1.
ENABLE_AUDIO_COMPENSATION_FILTER	Define this will enable audio compensation filter with speaker mode It can be “pure ACF” or ACF with DRC. Please reference ACF document for more detail.
ENABLE_AUDIO_DRC_SPEAKER	Define this will enable DRC for loudspeaker.

AUDIO_COMPENSATION_FLT_MODE	Define the mode of compensation filter, please reference ACF document.
ENABLE_STEREO_SPEAKER	if define Stereo speaker , speaker output will not do stero to mono, keep in stereo format because stereo output can apply on more than 1 speaker.
ENABLE_HIGH_SAMPLERATE_RECORD	When enable this feature , recording can support for 32K and 48KHz.
ENABLE_AUDIO_SW_STEREO_TO_MONO	Define this will enable SW stereo to mono on LCH & RCH If not define this, HW stereo to mono (only LCH) will be applied.
USE_REFMIC_IN_LOUDSPK	(1)->Use Ref Mic as main mic (0)->Use original main mic.
MagiLoudness_TE_mode	/***** *Define Speech DRC version *bit 0 normal DRC2.0 on=> bits 0 = 1 *bit 1 headset DRC2.0 on=> bits 1 = 1 *bit 2 speaker DRC2.0 on=> bits 2 = 1 *****/

2.2.13 NVRAM Operations

2.2.13.1 NVRAM Operating Flow

NVRAM is a non-volatile storage can let user to store data.

Audio and speech is built in system image, when system boot up, and user call NVRAM module to get specific data ID. if first reference to this ID, NVRAM module will get default value from system image and save to NVRAM.

When second time to reference data ID, NVRAM module will direct get data from NVRAM

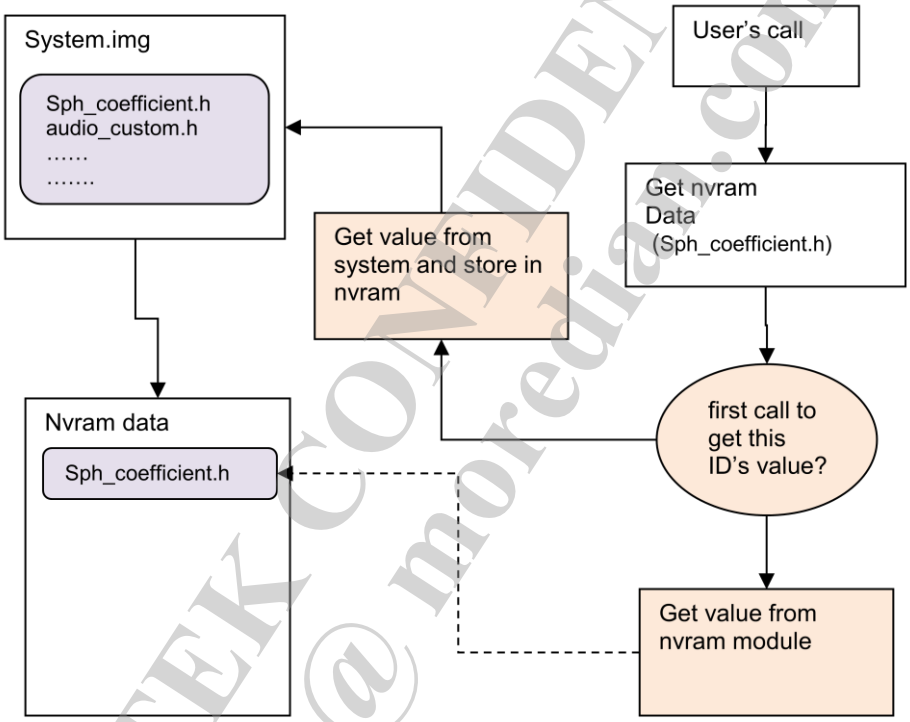


Figure 16. NVRAM Data Operating Flow

2.2.13.2 Audio NVRAM Function Call

Folder	Function	descriton
\mediatek\source\external\audiocustparam AudioCustParam.cpp	GetCustParamFromNV SetCustParamToNV GetAudioGainTableParamFromNV (deprecated) SetAudioGainTableParamToNV (deprecated) GetCustWBParamFromNV	Base on specific LID in Custom_NvRam_LID.h alps\mediatek\custom\($proj)\cgen\inc To get coefficient.

	SetCustWBParamToNV GetMedParamFromNV SetMedParamToNV GetAudioCustomParamFromNV SetAudioCustomParamToNV Read_DualMic_CustomParam_From_NVRAM Write_DualMic_CustomParam_To_NVRAM	
mediatek\source\external\HeadphoneCompensationFilter HeadphoneComFltCustParam.cpp	GetHeadphoneCompFltCustParamFromNV SetHeadphoneCompFltCustParamToNV	
mediatek\source\external\AudioCompensationFilte AudioComFltCustParam.cpp	GetAudioCompFltCustParamFromNV SetAudioCompFltCustParamToNV	

2.2.13.3 Related Files

AP side controls audio path, including normal playback and speech uplink and downlink path. MODEM side works as the slave of AP side. AP side sends command to MODEM side. All speech related configurations and control flow are controlled by AP side. Ex: volume setting, speech on/off, mode setting.

Below we briefly describe the file which need to be customized and the where the folders.

Android		
File Name	***Path***	***Description***
CFG_AUDIO_File.h	*/mediate/custom/common* */cgen/cfgfileinc*	*Audio data structure definition*
CFG_Audio_Default	*/mediate/custom/common* */cgen/cfgdefault*	*Parameters default value*
~~***audio_ver1_volume_custom_default.h***~~	~~*mediatek\custom\common\cgen\inc*~~ ~~*mediate*~~	~~*Audio volume default parameters*~~

2.2.13.4 File Locations

There may be two files with identical name and both place in.
/mediatek/custom/(%proj)/cgen/inc
mediate/custom/common /cgen/cfgfileinc

<path1> (\\alps\mediatek\custom\common\cgen\inc\sph_coeff_default.h)
- for common project customization. (for common phone general behavior)

<path2> (\\alps\mediatek\custom\($proj)\cgen\inc\sph_coeff_default.h)
-for specific project customization. (for specific phone behavior)

Below descript situation when file exist in different folders

Scenario	behavior
Path1 and path2 both has same file name	Use path2 file as parameters
Path1 has file, path2 has no this file.	Use path1 file as parameters
Path1 no file, path 2 has file	Use path2 file as parameters

2.2.13.5 File Location Suggestion

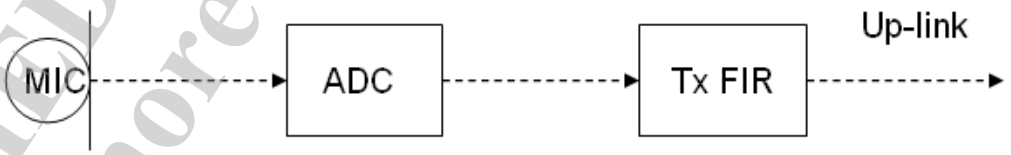
Apply to a specific project, please check in coefficient to alps\mediatek\custom\($proj)\cgen\inc

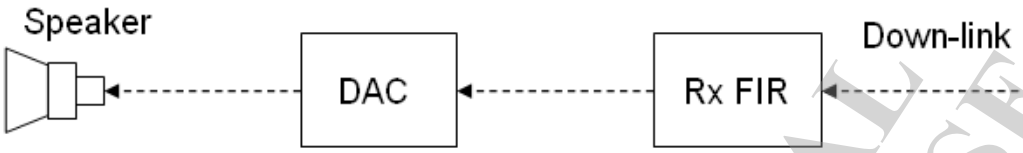
Apply to all projects, please check in coefficient to alps\mediatek\custom\common\cgen\inc\

2.2.14 Speech Setting

2.2.14.1 TX/RX/FIR

Currently, Tx/Rx FIR are stored in MODEM side only. The data structure is the same as feature phone.





- *The way of modifying FIR*
  - *Use Audio Tool*

- *There are several output devices supported in speech mode*
  - *Normal mode*
  - *Earphone mode*
  - *Loudspeaker mode*
  - *BT earphone mode*
  - *BT cordless mode*
  - *….*

Different speech enhancement algorithms are applied to different modes respectively. These algorithms are controlled by speech mode parameters

2.2.14.2 Speech Mode Parameters

Mode Parameters	Meaning
audio_custom_param. speech_mode_para[0][8]	Normal-mode speech enhancement parameters
audio_custom_param. speech_mode_para[1][8]	Earphone-mode speech enhancement parameters
audio_custom_param. speech_mode_para[2][8]	Loudspeaker-mode speech enhancement parameters
audio_custom_param. speech_mode_para[3][8]	BT earphone-mode speech enhancement parameters
audio_custom_param. speech_mode_para[4][8]	BT cordless-mode speech enhancement parameters
audio_custom_param. speech_mode_para[5][8]	Car-kit speech enhancement parameters
audio_custom_param. speech_mode_para[6][8]	Reserved for future use.
audio_custom_param. speech_mode_para[7][8]	Reserved for future use.

3 Audio Effects

Audio Compensation Filter (ACF) is used for compensating the audio quality according to physical case. ACF is applied on audio stream during audio playback. All ACF related configurations and control flow are controlled by AP side. Ex: IIR coefficient, ACF switch and Loudness switch.

Android		
File Name	***Path***	***Related Customizaiton***
aurisys_config.xml	*\vendor\mediatek\proprietary\external\aurisys* *(phone path :) system\vendor\etc*	*Depicts the library's capability and using scenario.*
PlaybackACF_AudioParam.xml	*\device\mediatek\common\audio_param* *(phone path :) system\vendor\etc\audio_param*	*High-pass IIR coefficient* *Band-pass IIR coefficient* *Low-pass IIR coefficient*
audio_custom_exp.h	*\vendor\mediatek\proprietary\custom\(%porj)\hal\audioflinger\audio*	*Turn on/off ACF (**ENABLE_AUDIO_COMPENSATION_FILTER**)*
MtkAudioLoudc.c	*\vendor\mediatek\proprietary\external\aurisys\libaudioloud* *(phone path :) system\vendor\lib\libaudioloudc.so*	*ACF/DRC/HCF aurisys wrapper implementation*
MtkAudioLoudc.h	*\vendor\mediatek\proprietary\external\aurisys\libaudioloud*	*ACF/DRC/HCF aurisys wrapper header file*
AudioCompFltCustParamc.c	*\vendor\mediatek\proprietary\external\AudioCompensationFilter*	*Interact with AudioParamParser ,getting parameters from XML*
AudioCompFltCustParamc.h	*\vendor\mediatek\proprietary\external\AudioCompensationFilter*	*C code implementation header file for param related function.*
AudioCompensationFilterc.h	*\vendor\mediatek\proprietary\external\AudioCompensationFilter*	*C code implementation header file, define audio compensation type and mode.*
libbessound_hd_mtk_vendor.so	*\vendor\mediatek\proprietary\external\bessound_HD*	*ACF/DRC/HCF library*

3.1 ACF Working Flow

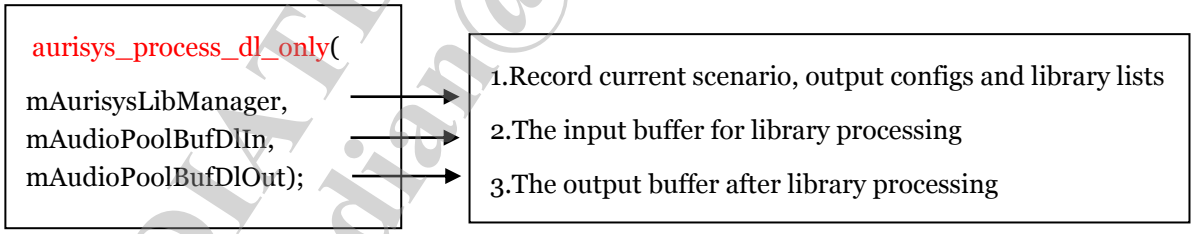
We change the ACF working flow with Aurisys framework. The post process will be triggered and done through the ARSI standard interface. It is the C-based framework thus we modify the files as C format and belows are the changes in our design.

1. In aurisys_config.xml in vendor\mediatek\proprietary\external\aurisys
   - *Add BesSound libarary description*
2. In vendor\mediatek\proprietary\external\aurisys\libaudioloud add
   - *MtkAudioLoudc.c*
   - *MtkAudioLoudc.h*
3. In vendor\mediatek\proprietary\external\AudioCompensationFilter add
   - *AudioCompFltCustParamc.c*
   - *AudioCompFltCustParamc.h*
4. In audio hal, vendor\mediatek\proprietary\hardware\audio, we use

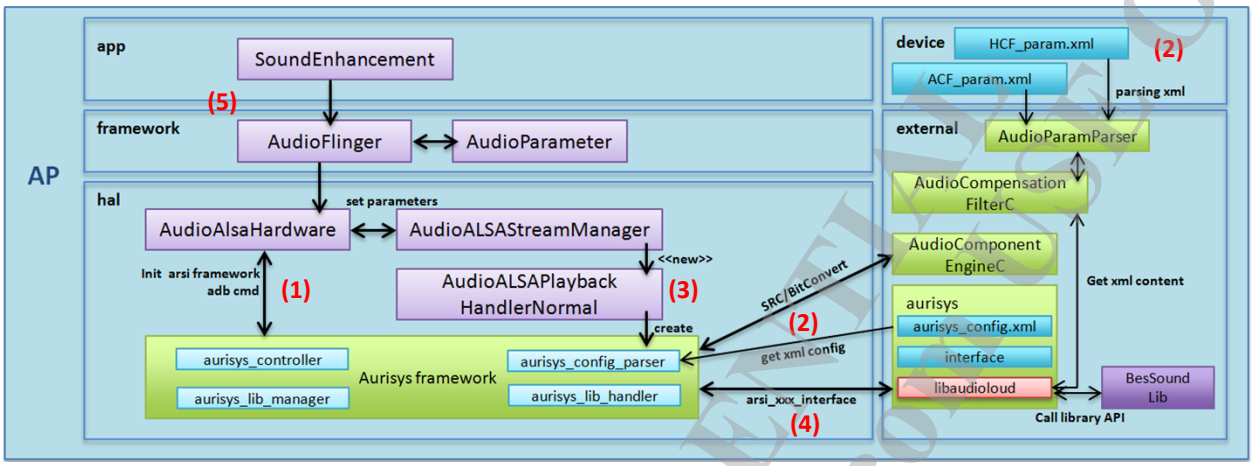
   ***#define MTK_AURISYS_FRAMEWORK_SUPPORT** to identify the new process method.*

   The compile option is set in device\mediatek\{project}\ProjectConfig.mk

At the bootup stage, the ARSI framework initialize and reads the content of aurisys_config.xml, establishing the framework function links and libraries' function links. When first running ACF, the AudioParamParser reads back the paramenters in PlaybackACF_AudioParam.xml and stores in the structure. The processing starts from AudioplaybackHandlerxxx, which indicates the scenario of playback, calling doPostProcessing functions in AudioplaybackHandlerBase.cpp, in this function we check the **MTK_AURISYS_FRAMEWORK_SUPPORT** configuration and decides the processing method. Through the aurisys framework, we use the generic interface **aurisys_process_dl_only** as the process entry.



The working flow is depicted as below, there are five functional blocks. Application UI, Framework, Audio HAL, Device and External hal. The Aurisys Framewok is implemented in Audio Hal and related ARSI general API and aurisys_config.xml are defined in external block.



(1) The aurisys framework is initialized by AudioALSAHardware when audioserver start.

(2) During the initialization stage, the aurisys controller parse aurisys_config.xml , establishing the function callback between library and framework, and dlopen **AudioComponentEngine** libraries (bit converter, samplerate converter.)

For ACF library, we parse the ACF_param.xml and store the parameters into the structure. These parameters set the processing frequency and generate corresponding filter coefficients.

(3) When normal or fast playback starts, we create libmanager and libhandler to handle the processing flow under this scenario. As we can retrieve the stream and hardware info, we store these information and provide to the framework.

If the playback setting is not compatible to the library setting, the Aurisys Framework will trigger samplerate converter or bit converter to do the transformation. This ensure the process can be done without library modification.

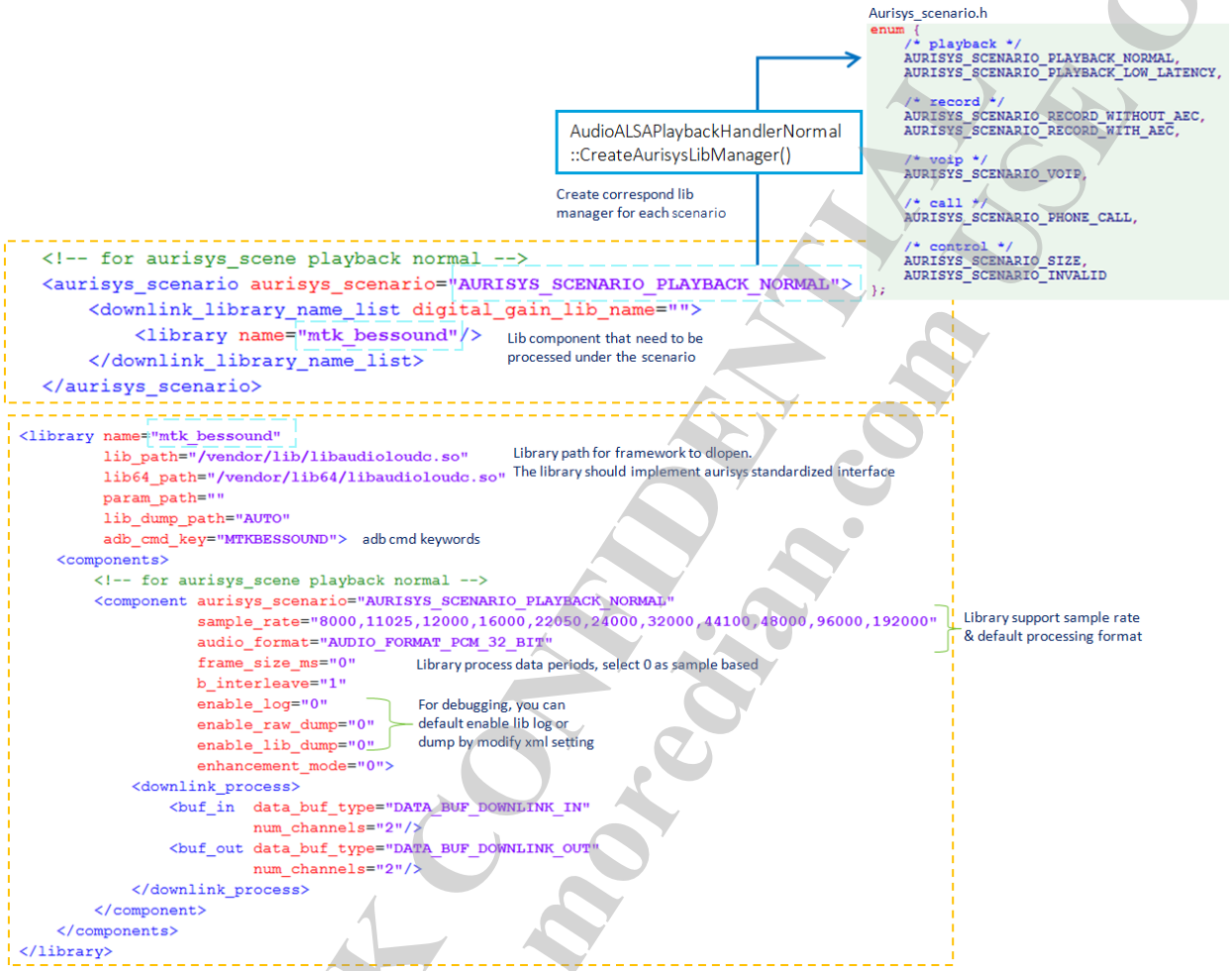
(4) The post-processing is done by calling **aurisys_process_dl_only()** . The framework handles the data transformation and operates with libraries through ARSI generic interface

(5) The application UI can enable function through setparameter path.

There are two key points for the libray porting.

- One is to depicts the library configuration in aurisys_config.xml.
- Anothor one is to implement the wrapper between ARSI interface and library functions.

In aurisys_config.xml, we first indicate the using scenario and list the library component used.We depict the library's attribute, such as file path, library supported samplerate, processing format, frame size, data buffer type (uplink or downlink), channel number, and debug level….etc. These attributes are compared to the input pcm data and indicate the library setting during processing. The user can change the library setting by simply modify the xml file. This enlarge the flexibility and reduce the coding effort.



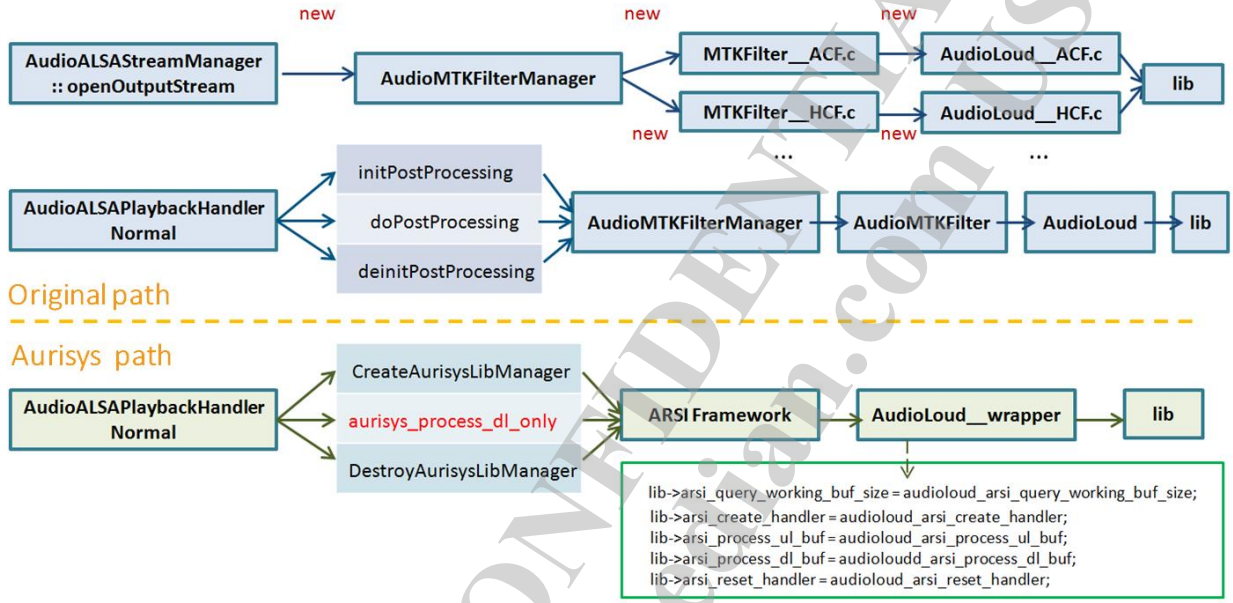
To implement the wrapper between library and ARSI interface, we comparing and mapping their functionality as the table shows.

Aurisys interface	Effect interface	description
audioloud_query_working_buf_size	BLOUD_HD_SetHandle(&mBloudHandle); BLOUD_HD_GetBufferSize	Get pcm format from task config and return total need working buffer size
audioloud_create_handler	mBloudHandle.Open	Assign handler and init lib
audioloud_process_dl_buf	mBloudHandle.Process	Process p_dl_buf_in and generate p_dl_buf_out which data applyies ACF filter.

mapping the interface between ARSI and Effect

Comparing to the original post-processing path, the Aurisys path will not new the instance for every pre-defined scenario at the first loading stage. Instead, it checks the current device info by task config and create corresponding library instance. The calling interfaces changed in PlaybackHandlerNormal and uses the aurisys_process_dl_only() as the unified entry for every post-processing libray.

This means the users do not need to do extra judge or control coding when adding a new library .Once you have done the aurisys_config.xml description and implemented the wrapper. The library components can be sequentially processed.



Comparing the processing path

3.1.1 ACF Setting

The chapter shows the control settings of ACF.

3.1.1.1 Turn On/ Turn Off ACF in HAL

In audio_custom_exp.h, ACF can be enable/disable by the following define.

#define ENABLE_AUDIO_COMPENSATION_FILTER

If the define is available, ACF in HAL is enabled.

If the define is marked, ACF in HAL is disabled.

3.1.1.2 Dynamic turn on/off library log using adb command

Besides setting xml ,user can runtime key in adb debug command to enable/disable library processing log in mobile log.

ON	adb shell "AudioSetParam AURISYS_SET_PARAM,HAL,PLAYBACK_NORMAL,MTKBESSOUND,ENABLE_LOG,1=SET"
OFF	adb shell "AudioSetParam AURISYS_SET_PARAM,HAL,PLAYBACK_NORMAL,MTKBESSOUND,ENABLE_LOG,0=SET"

3.1.1.3 ACF Parameters

In ACF implementation, there're some parameters can be set in algorithm.

The default value of parameters is listed in PlaybackACF_AudioParam.xml.

3.1.1.3.1 IIR

For the audio frequency tuning, there're three IIRs appliled to audio data.

One is High pass filter, one is band pass filter, and the other is low pass filter.

- ***L HPF Coefficients***

```
#define BES_LOUDNESS_ACF_L_HPF_FC      300
#define BES_LOUDNESS_ACF_L_HPF_ORDER   4
```

- ***L BPF Coefficients***

```
#define BES_LOUDNESS_ACF_L_BPF_FC      800,1500,0,0,0,0,0,0
#define BES_LOUDNESS_ACF_L_BPF_BW      1000,1000,0,0,0,0,0,0
#define BES_LOUDNESS_ACF_L_BPF_GAIN    -1533,-1535,0,0,0,0,0,0
```

- ***L LPF Coefficients***

```
#define BES_LOUDNESS_ACF_L_LPF_FC      8000
#define BES_LOUDNESS_ACF_L_LPF_ORDER   1
```

- ***R HPF Coefficients***

```
#define BES_LOUDNESS_ACF_R_HPF_FC      0
#define BES_LOUDNESS_ACF_R_HPF_ORDER   0
```

- ***R BPF Coefficients***

```
#define BES_LOUDNESS_ACF_R_BPF_FC      0,0,0,0,0,0,0,0
#define BES_LOUDNESS_ACF_R_BPF_BW      0,0,0,0,0,0,0,0
#define BES_LOUDNESS_ACF_R_BPF_GAIN    0,0,0,0,0,0,0,0
```

- ***R LPF Coefficients***

```
#define BES_LOUDNESS_ACF_R_LPF_FC      0
```

```
#define BES_LOUDNESS_ACF_R_LPF_ORDER    0
```

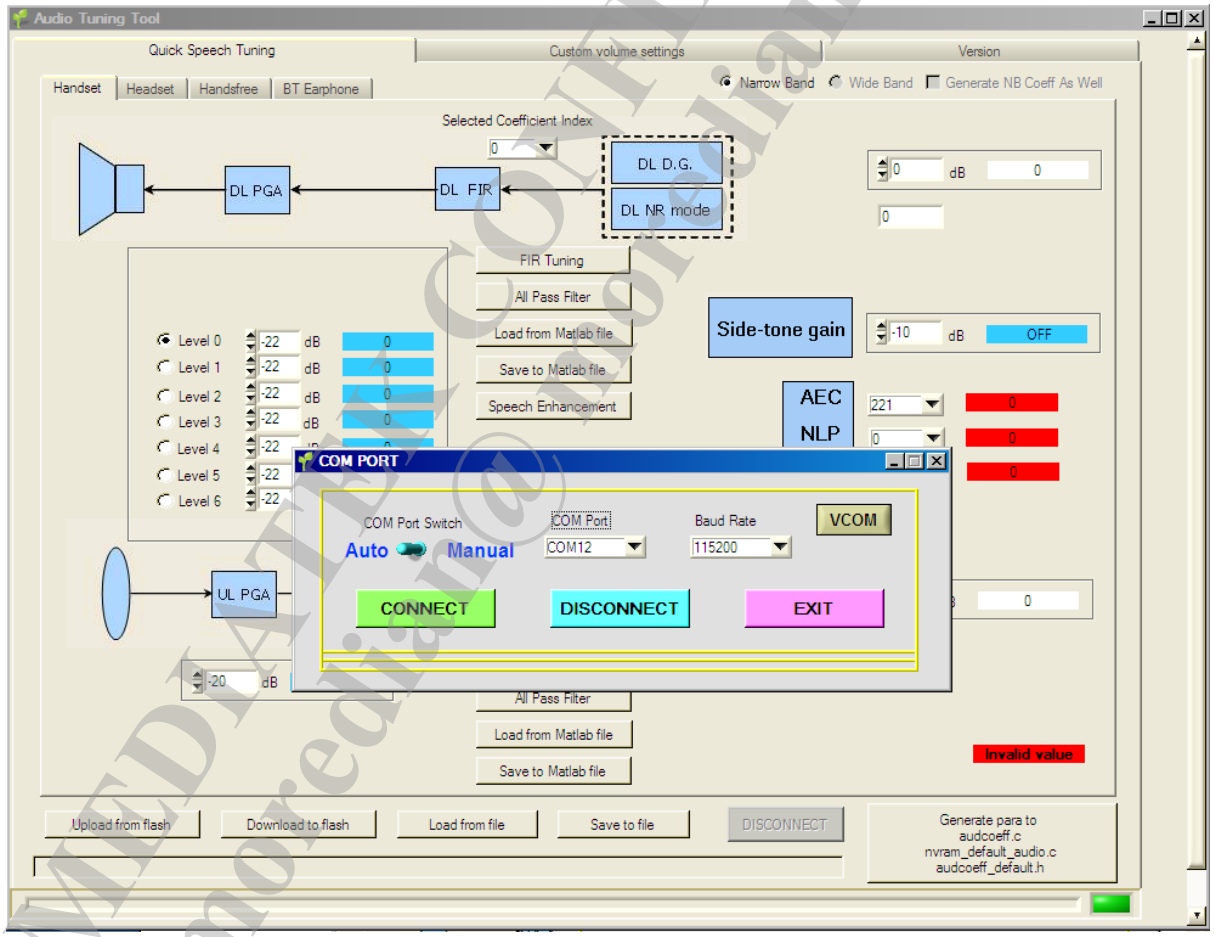
3.1.2 ACF Tuning on Audio Tool

3.1.2.1 Enter Audio Tool

The step of entering:

1. Insert USB Line to target phone
2. Click power-on button on target phone
3. Select Auto for COM Port Switch
4. Select target phone COM Port
5. CONNECT

The figure shows the window of audio tool

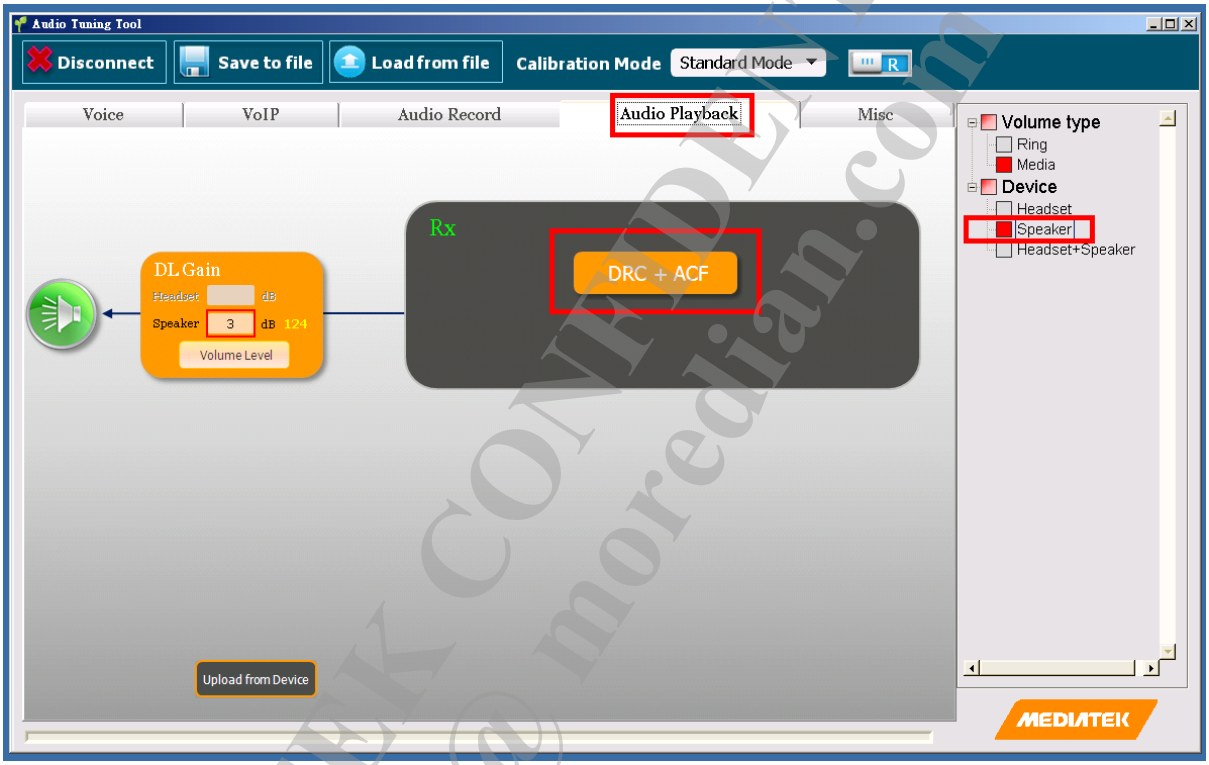


If successfully entering Audio Tool, the audio tool window below will be selected. User should select page "Audio Playback "-> " Speaker" -> "DRC+ACF", then user can see the tuning page.

If the error message occurs, it is fail to entering Audio Tool.

If it is fail to entering Audio Tool, the possible reason may be

1) *Corrupted line*
2) *The version of audio tool and android load don't match.*
3) *Others*

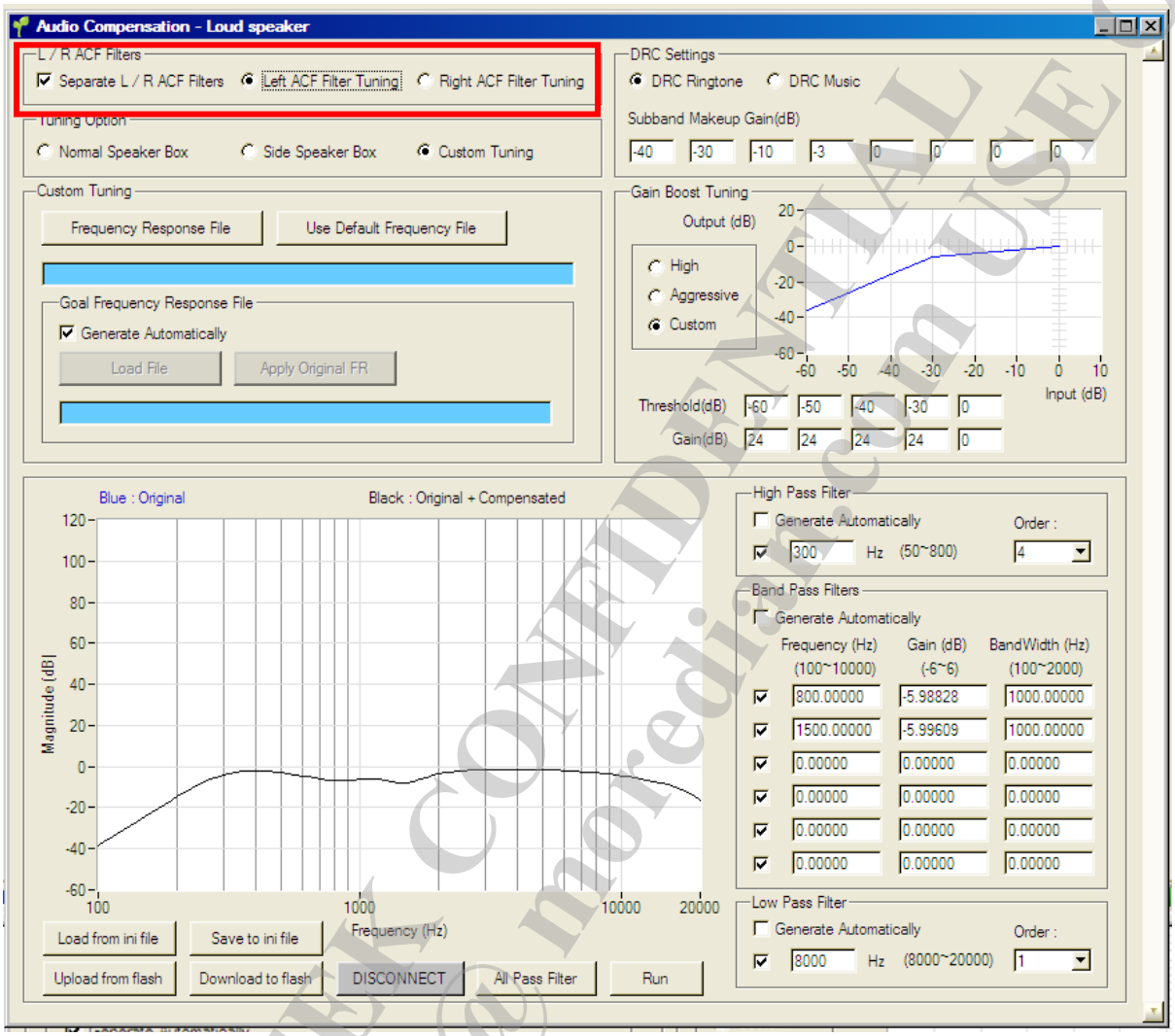


3.1.2.2 L/R ACF Filter

In the "L/R ACF Filter" group, there is one checkbox which can be selected.

- *Separate L/R ACF Filters*

If Separate L/R ACF Filters is selected, the L/R ACF Filters will be shown as bellow; the user can select tuning for L or R ACF Filters.

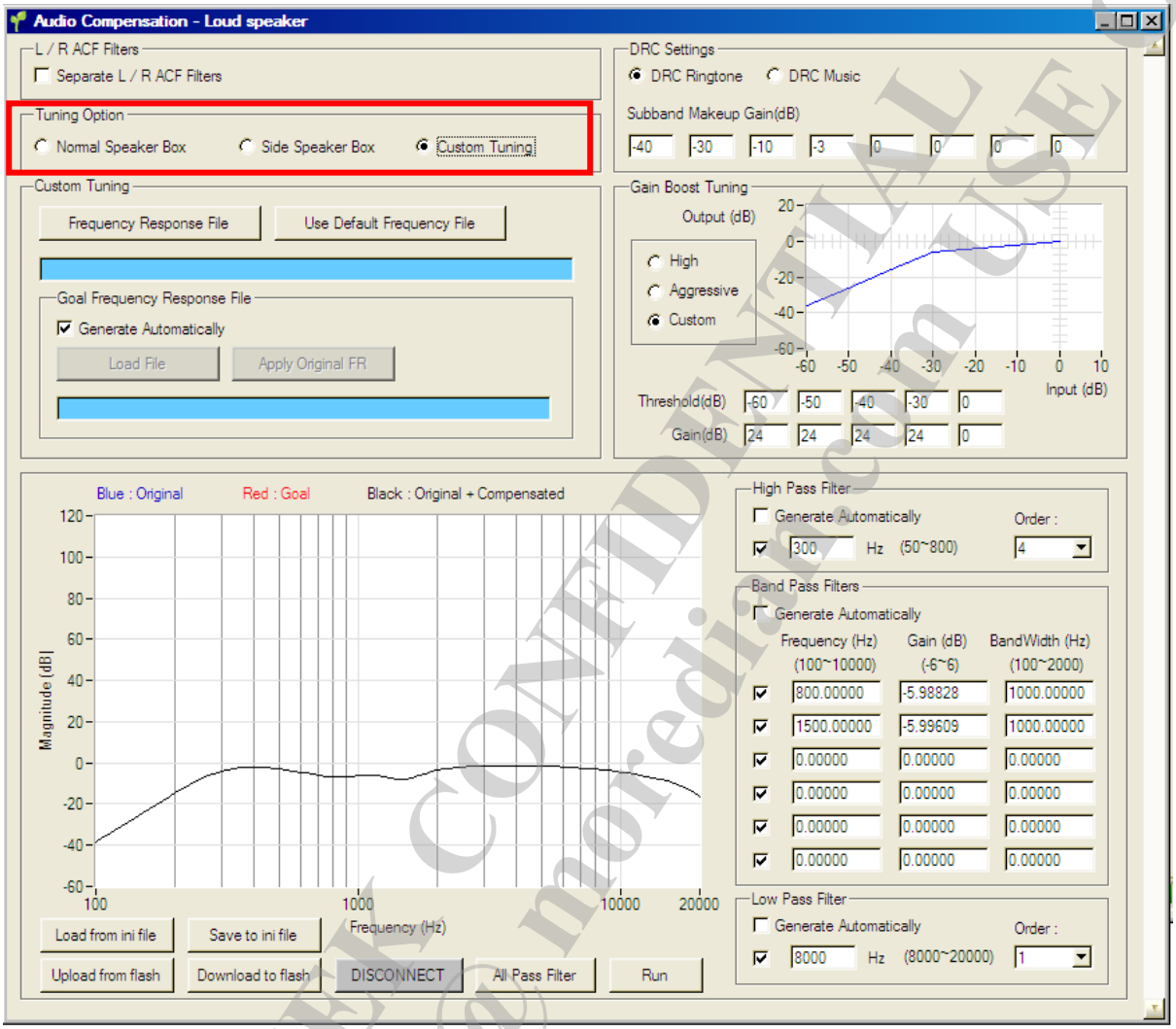


3.1.2.3 Tuning Option

In the "Tuning Option" group, there're 3 type can be selected.

- *Normal Speaker Box*
- *Side Speaker Box*
- *Custom tuning*

If custom tuning is selected, the Custom Tuning Area will be shown.



3.1.2.4 Adjust ACF IIR Coefficients

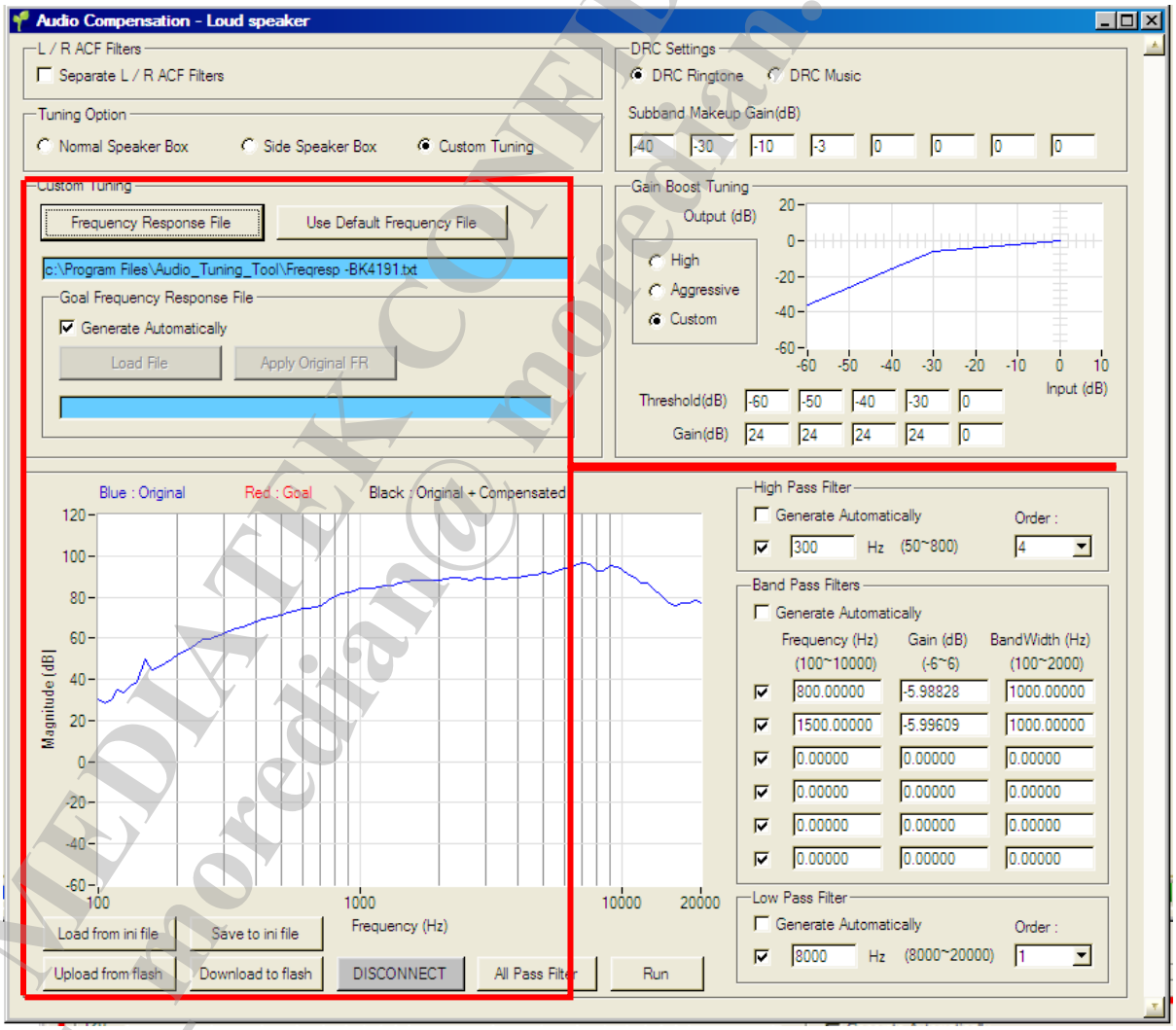
If custom tuning or Custom Tuning II are selected, the ACF IIR Coefficient can be adjusted.

User can adjust ACF IIR by the following buttons and checkboxes.

- *Button usage:*
  - *Frequency Response File*
    - *Load one Frequency Response file*
  - *Goal Frequency Response File*
    - *Load file or generate Frequency Response File by this page*
  - *Run*
    - *Generate and Show Frequency response curve of IIR*
- *Checkbox usage:*

- *High pass filter*
  - *2 options can be selected*
    - *Generate automatically*
    - *Indicate the cut-off frequency of high pass filter*
  -
- *Band pass filter*
  - *2 options can be selected*
    - *Generate automatically*
    - *Indicate the cut-off frequency of band pass filter. There're six bands can be set by user.*
- *Low pass filter*
    - *Generate automatically*

Indicate the cut-off frequency of low pass filter



3.2 Audio Loudness (DRC) Guide

3.2.1 Introduction to Loudness

Loudness (DRC) is an audio post-processing technique to improve the loudness of audio signal and ringtone playback by using dynamic range control and frequency component manipulation

Android		
File Name	**Path**	**Related Customizaiton**
PlaybackDRC_AudioParam.xml	\device\mediatek\common\audio_param\	DRC related coefficient for audio and ringtone
ProjectConfig.mk	device\mediatekprojects\(%proj)\	Is the platform support Loudness MTK_BESLOUDNESS_SUPPORT Is the platform support Loudness processing on HAL layer MTK_BESLOUDNESS_RUN_WITH_HAL (Trun on need to combine with MTK_AURISYS_FRAMEWORK_SUPPORT=yes)
DRC in Audioflinger path		
MtkAudioLoud.cpp	\vendor\mediatek\proprietary\external\audiocomponentengine\	ACF/DRC/HCF implementation
Android.mk	\frameworks\av\services\audioflinger	Turn on DRC apply into AudioFlinger layer
libbessound_hd_mtk.so	\vendor\mediatek\proprietary\external\bessound_HD	ACF/DRC/HCF system partition library
DRC in HAL path		
MtkAudioLoudc.c	\vendor\mediatek\proprietary\external\aurisys\libaudioloud\ (phone path :) system\vendor\lib\libaudioloudc.so	ACF/DRC/HCF aurisys wrapper implementation
MtkAudioLoudc.h	\vendor\mediatek\proprietary\external\aurisys\libaudioloud\	ACF/DRC/HCF aurisys wrapper header file
Android.mk	\vendor\mediatek\proprietary\external\aurisys\libaudioloud\	Check ProjectConfig MTK_BESLOUDNESS_SUPPORT & MTK_BESLOUDNESS_

		RUN_WITH_HAL . *Turn on DRC apply into HAL layer*
libbessound_hd_mtk_vendor.so	*\vendor\mediatek\proprietary\external\bessound_HD*	*ACF/DRC/HCF vendor partition library*

3.2.2 Loudness (DRC) Setting

The chapter shows the control settings of Loudness (DRC). The aurisys framework design had been enabled in newer version chips. Thus there are two method to apply DRC, one is aurisys framework with DRC support in HAL, another is DRC applied in Audioflinger. We depict the two different flow in 3.2.3 chapter.

3.2.2.1 Loudness (DRC) configuration

In ProjectConfig.mk, , Loudness(DRC) can be enable/disable and deciding the processing layer by the following define.

	MTK_BESLOUDNESS_RUN_WITH_HAL = yes	MTK_BESLOUDNESS_RUN_WITH_HAL = no
MTK_BESLOUDNESS_SUPPORT = yes	**Support DRC feature and processing at HAL**	**Support DRC feature and processing at Frameworks**
MTK_BESLOUDNESS_SUPPORT= no	**Not Support DRC feature**	**Not Support DRC feature**

3.2.3 Loudness (DRC) Working Flow

Generally, the Loudness (DRC) parameters are stored in XML.

*In the case, when the first time of Loudness (DRC) processing, the Loudness (DRC) parameters will be applied as the default Loudness (DRC) parameter listed in **PlaybackDRC_AudioParam.xml**.*

Then users can modify some parameters in Audio Tool if needed.

If config **MTK_BESLOUDNESS_SUPPORT = yes** and **MTK_BESLOUDNESS_RUN_WITH_HAL = no**

The framework working flow of Loudness (DRC) will be show as the figure.

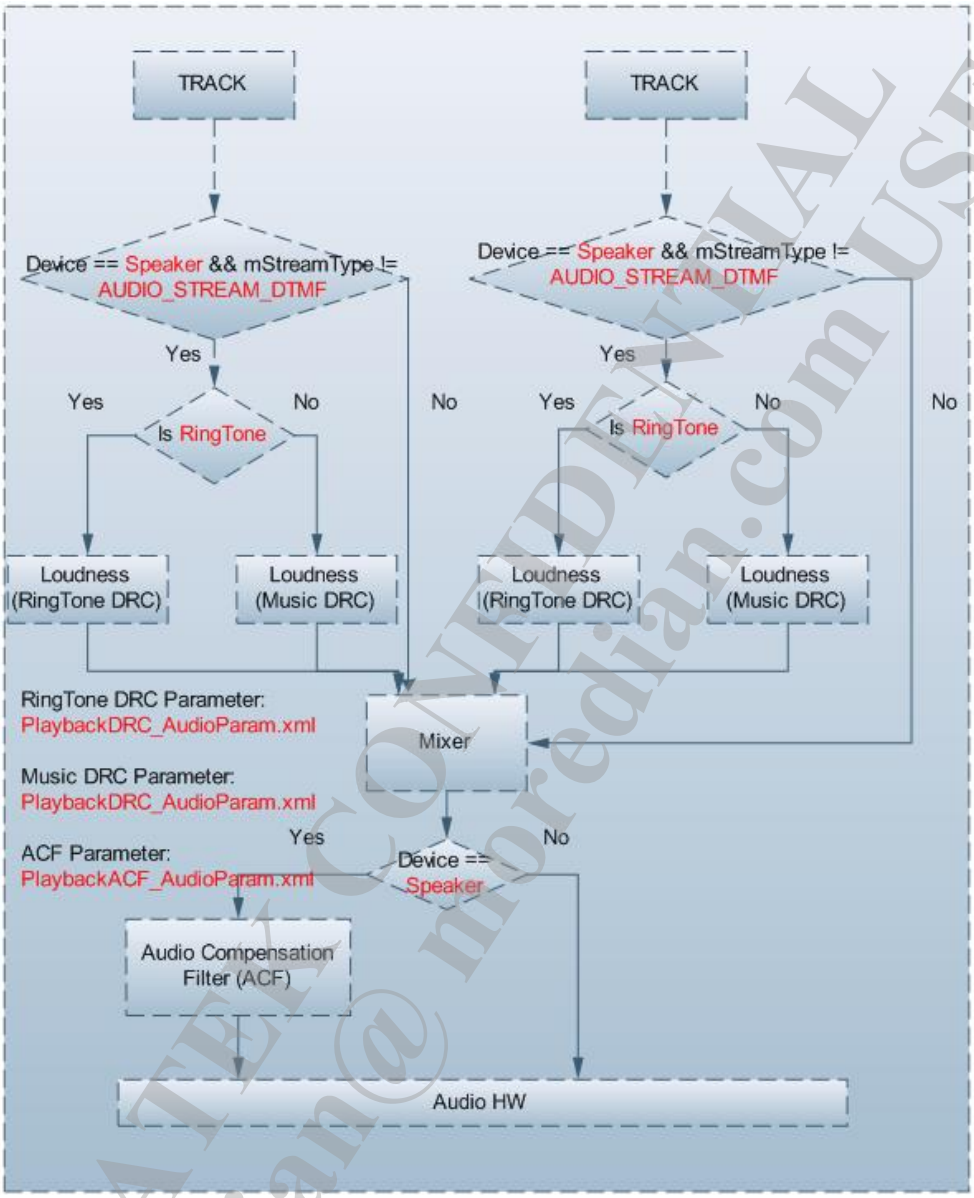


Figure 17. DRC Working at Framework Flow

If config **MTK_BESLOUDNESS_SUPPORT = yes** and **MTK_BESLOUDNESS_RUN_WITH_HAL = yes**

The HAL working flow of Loudness (DRC) will be show as the figure

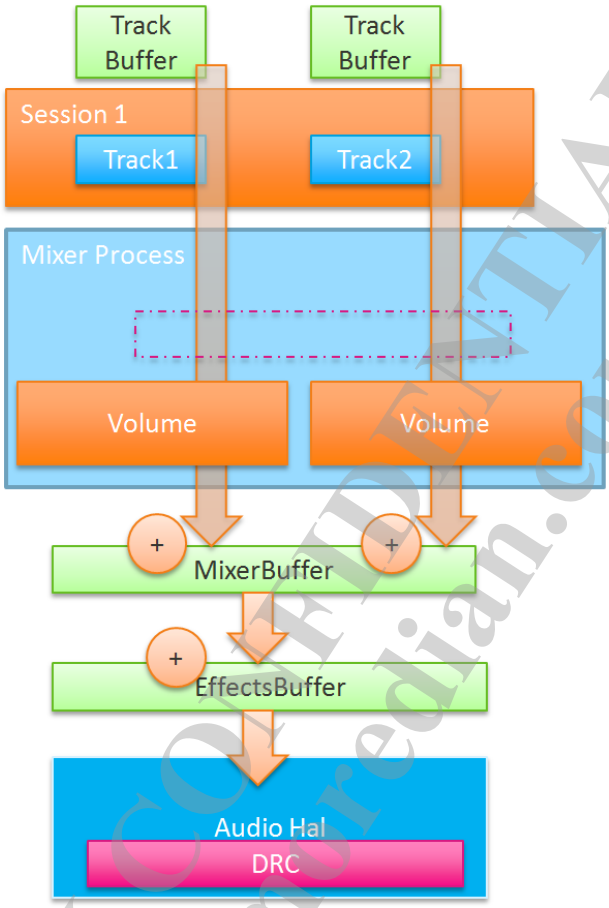


Figure 18. DRC Working at HAL overview

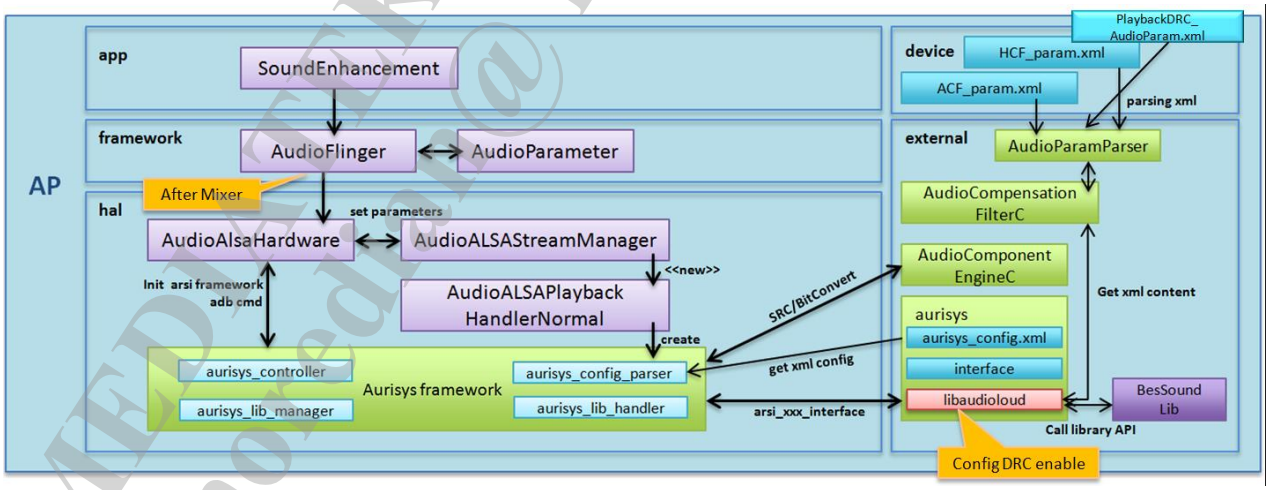


Figure 19. DRC Working at HAL Flow

The advantages for DRC processing in HAL includes :

1. Reduce CPU computing loading, saving power.
2. Make Framework code clean and less migration effort.

But it may cause the volume curve not linear since the DRC applied after volume mix.

3.2.3.1 DRC Parameters

In DRC implementation, there're some parameters can be set in algorithm.
The default value of parameters is listed in PlaybackDRC_AudioParam.xml
And the detail information can be listed in BesLoudness_HD_exp.h

- ***BES_LOUDNESS_MUSICDRC_WS_GAIN_MAX***
  Currently not used.
- ***BES_LOUDNESS_MUSICDRC_WS_GAIN_MIN***

  Currently not used.
- ***BES_LOUDNESS_MUSICDRC_FILTER_FIRST***
  Currently not used.
- ***BES_LOUDNESS_MUSICDRC_NUM_BANDS***

  Number of subbands for multi-band DRC

  Range: 1 ~ 8
- ***BES_LOUDNESS_MUSICDRC_FLT_BANK_ORDER***
  Filter order for filter bank
  Range: 3, 5, 7
- ***BES_LOUDNESS_MUSICDRC_DRC_DELAY***
  Currently not used.
- ***BES_LOUDNESS_MUSICDRC_CROSSOVER_FREQ***
  Subband crossover frequencies, unit: Hz
- ***BES_LOUDNESS_MUSICDRC_SB_MODE***
  Subband mode
  0: makeup gain
  1: subband limiter
  2: bypass
  3: mute
- ***BES_LOUDNESS_MUSICDRC_SB_GAIN***
  Subband gain, Q24.8, unit: dB
- ***BES_LOUDNESS_MUSICDRC_GAIN_MAP_IN***
  DRC curve threshold for input signal, Q24.8, unit: dB
- ***BES_LOUDNESS_MUSICDRC_GAIN_MAP_OUT***
  DRC curve gain, Q24.8, unit: dB
- ***BES_LOUDNESS_MUSICDRC_ATT_TIME***
  Attack time of each subband, unit: 0.1 ms / 6dB
- ***BES_LOUDNESS_MUSICDRC_REL_TIME***
  Release time of each subband, unit: 0.1 ms / 6dB
- ***BES_LOUDNESS_MUSICDRC_HYST_TH***
  Release hysteresis threshold, Q24.8, unit: dB
- ***BES_LOUDNESS_MUSICDRC_LIM_TH***
  Limiter threshold, Q17.15, range: 1 ~ 0x7FFF
- ***BES_LOUDNESS_MUSICDRC_LIM_GN***
  Limiter gain, Q17.15, range: 1 ~ 0x7FFF
- ***BES_LOUDNESS_MUSICDRC_LIM_CONST***
  Limiter recovery constant, unit: step size / sample, range: 1 ~ 100
- ***BES_LOUDNESS_MUSICDRC_LIM_DELAY***
  Currently not used.

The parameters of PlaybackDRC_AudioParam.xml is the same with audio_music_drc_default.h, however they are applied to Ringtone Streams.

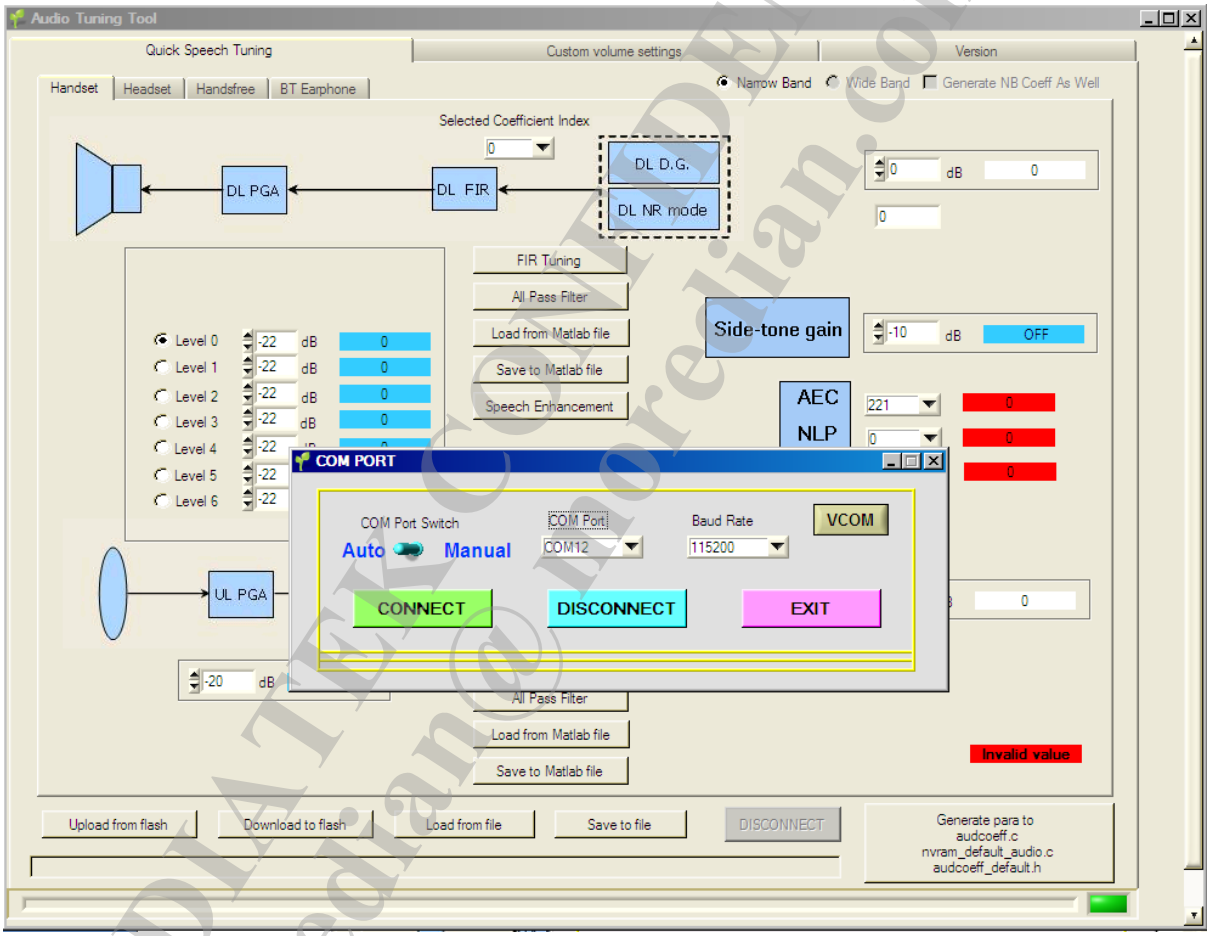
3.2.4 DRC Tuning on Audio Tool

3.2.4.1 Enter Audio Tool

The step of entering:

1. Insert USB Line to target phone
2. Click power-on button on target phone
3. Select Auto for COM Port Switch
4. Select target phone COM Port
5. CONNECT

The figure shows the window of audio tool

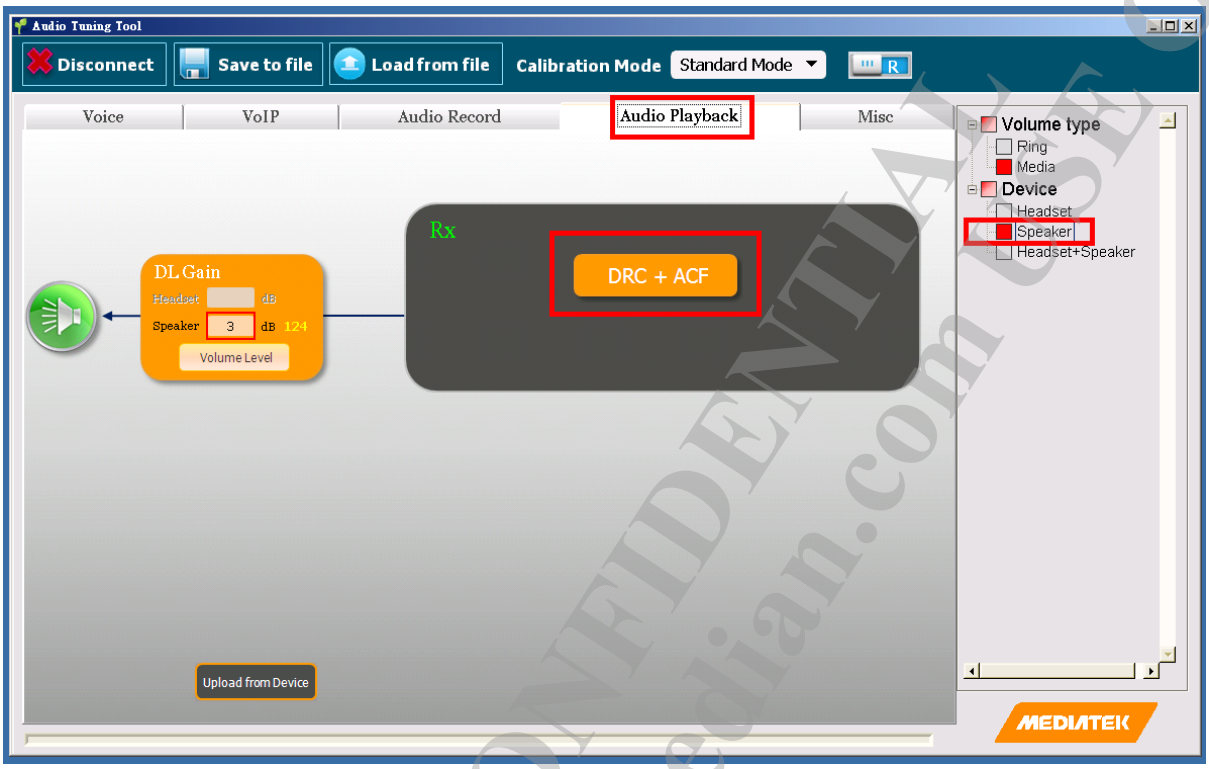


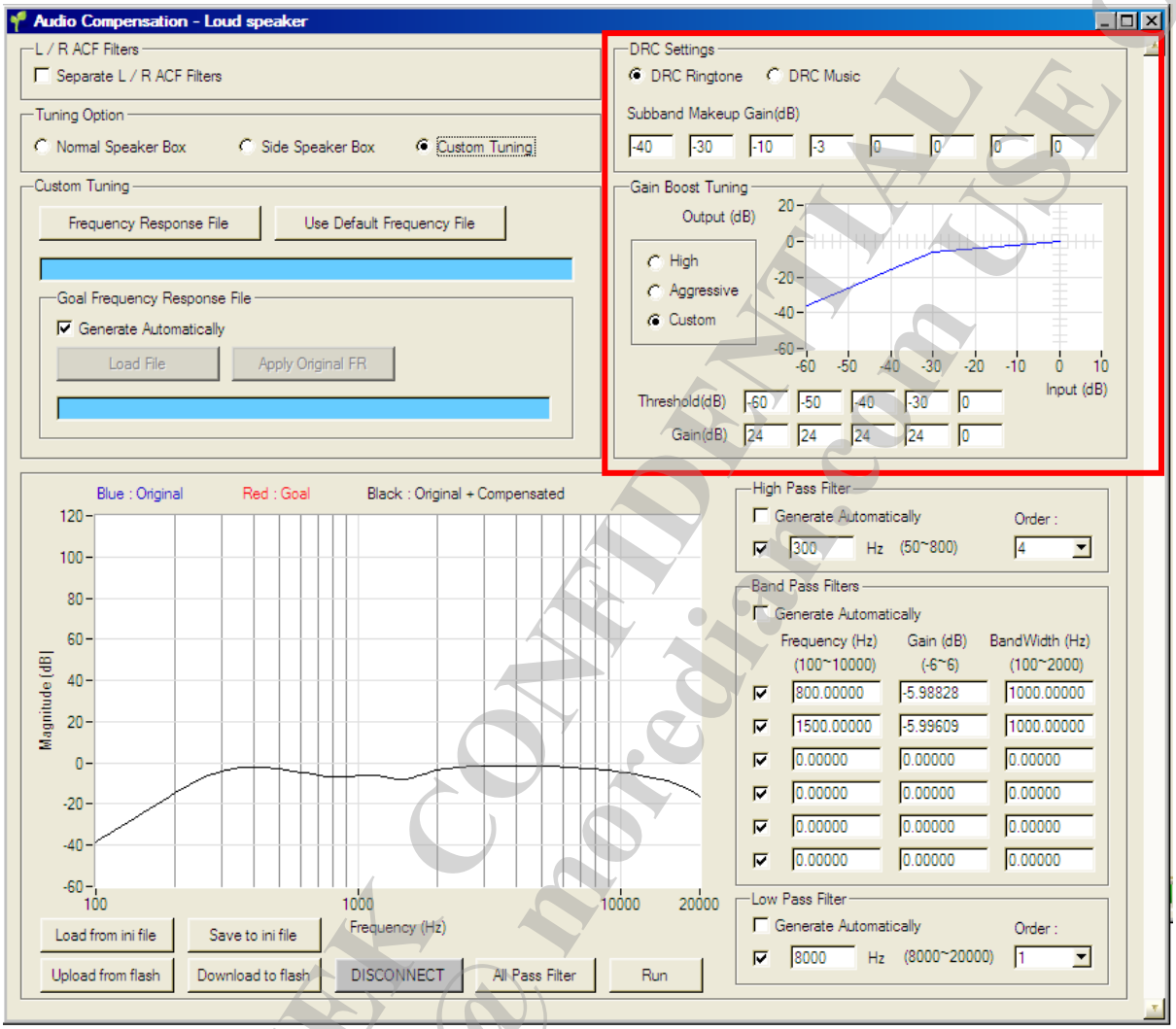
If successfully entering Audio Tool, the audio tool window below will be selected. User should select the page "Audio Playback" -> "Speaker" -> "DRC+ACF", then user can see the tuning page.

If the error message occurs, it is fail to entering Audio Tool.

If it is fail to entering Audio Tool, the possible reason may be

1) *Corrupted line*
2) *The version of audio tool and android load don't match.*
3) *Others*



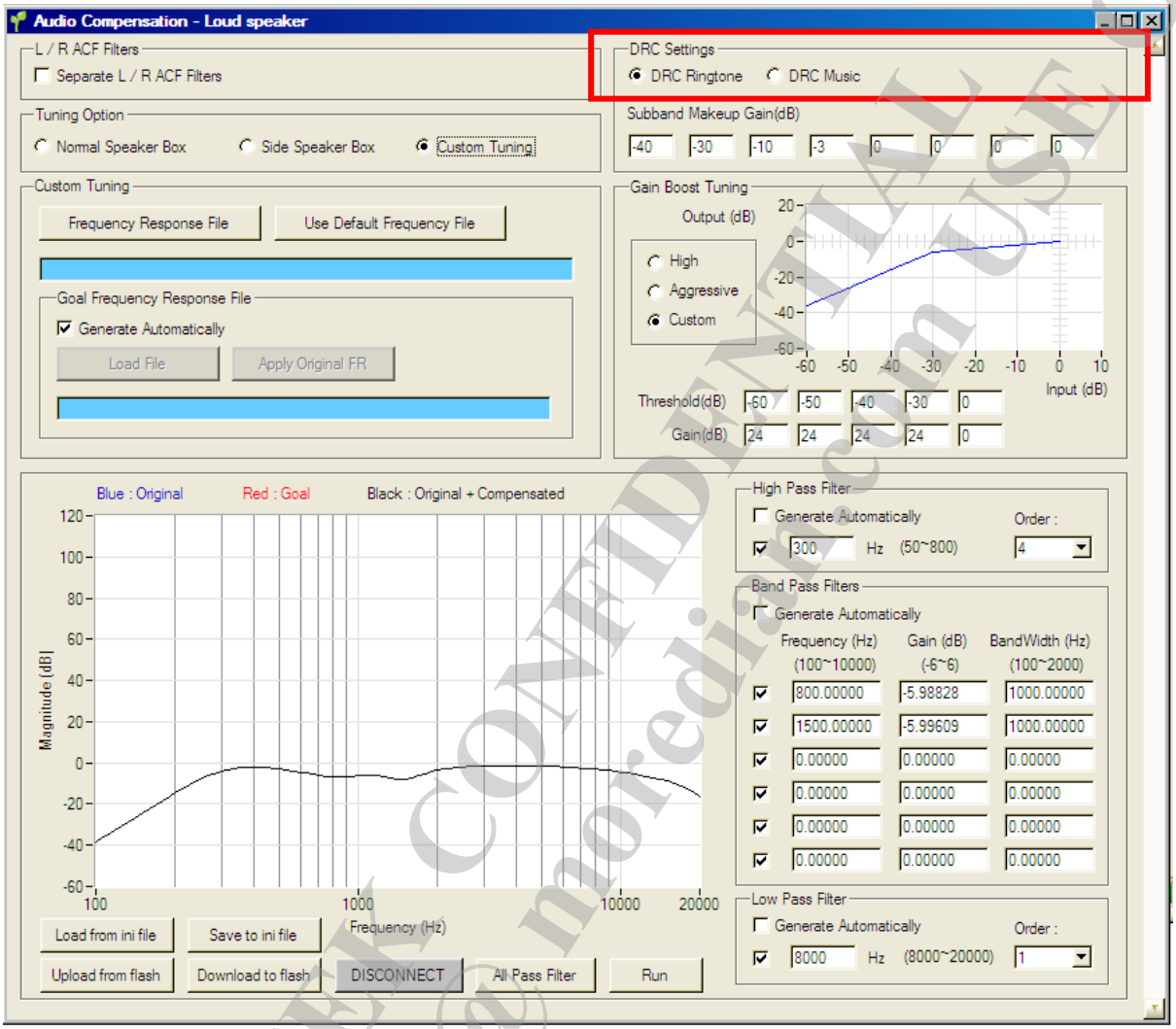


3.2.4.2 DRC Setting Option

In the "DRC Setting" group, there're 2 type can be selected.

- *DRC Ringtone*
- *DRC Music*

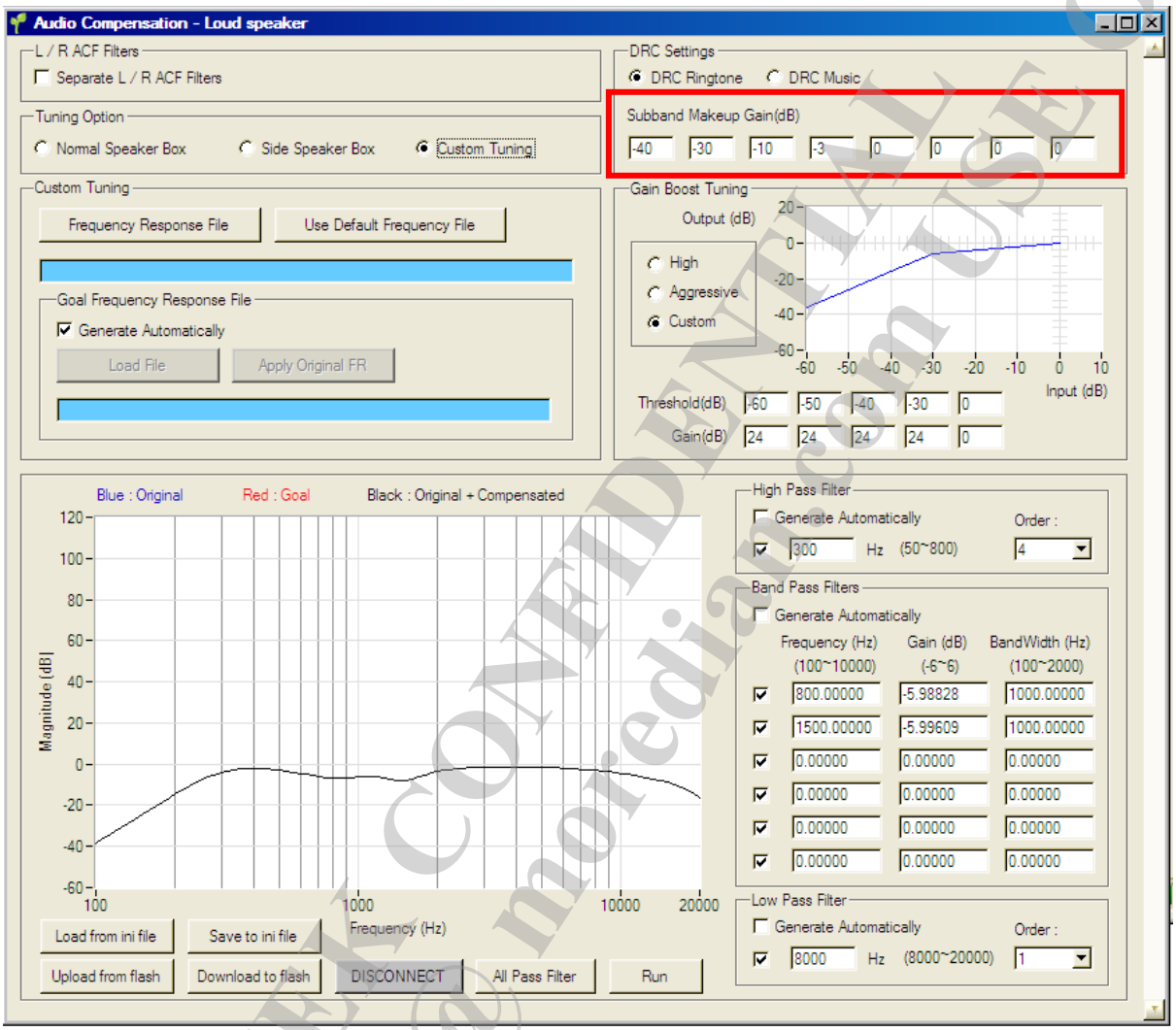
The user can select someone type DRC to tune.



3.2.4.3 Subband Makeup Gain

Set the gain of each subband

There're 8 bands can be used.

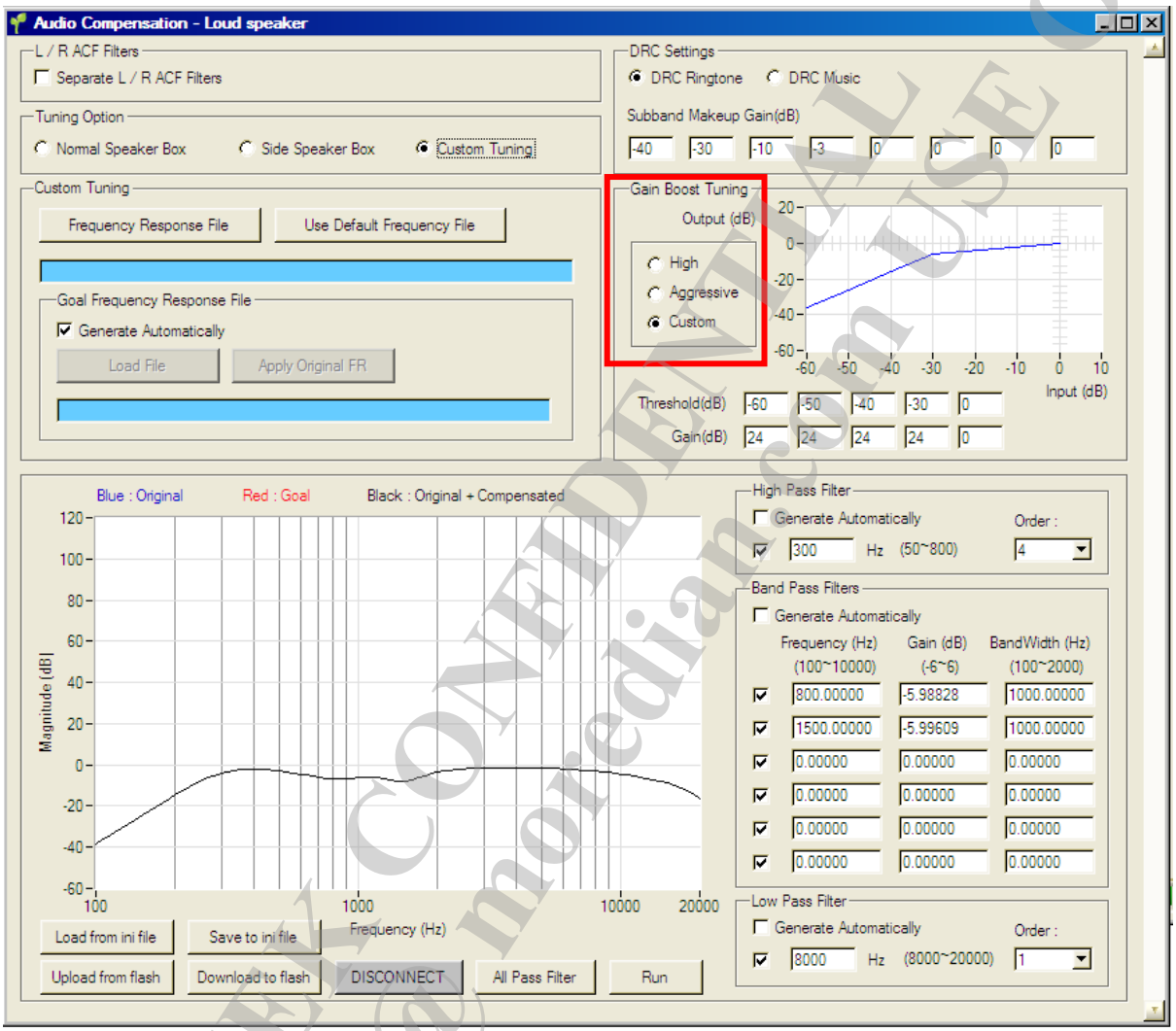


3.2.4.4 Gain-Boost Tuning

Set the Maximum Gain and Minimum Gain

There're 3 types can be used.

- High
- Aggressive
- Custom



4 External HW Devices

4.1 MT6771 I2S Capability Support

I2S capability interface description:

- MTK Audio Interface
  - MTK **specific** interface for MT6358
- I2S0 – – ADC interface
  - Proposed for any audio interface
  - Support **Master**
  - Support **input**
  - Capability up to 192k/24bit.
  - Bit width and sampling rate in should be the same at I2S3.
- I2S1 – DAC interface
  - Support standard I2S in **Master output**
  - Maximum 1 data wire for 2 channel support
  - Capability up to 384k/32bit
- I2S2 – ADC interface
  - Support standard I2S in **Master input**
  - Maximum 2 data wire for 4 channel support
  - Capability up to 192k/24bit
- I2S3 – DAC interface
  - Support standard I2S in **Master output**
  - Maximum 1 data wire for 2 channel support
  - Capability up to 384k/32bit
  - Bit width and sampling rate in should be the same at I2S0.
- I2S5 – DAC interface
  - Support standard I2S in **Master output**
  - Maximum 1 data wire for 2 channel support
  - Capability up to 384k/32bit
- MCLK Support (I2S0/I2S1/I2S2/I2S3/I2S5)
  - Synced MCLK and I2S BCK
  - Capability up to 256fs in 192k mode
  - Capability up to 512fs in 96k mode
  - Capability up to 1024fs in 48k mode

Didn't support 192/384fs related

IP	*Pin Name*
MTK Audio Interface	*AUD_CLK_MOSI*
	AUD_SYNC_MOSI
	AUD_DAT_MOSI0
	AUD_DAT_MOSI1
	AUD_CLK_MISO
	AUD_SYNC_MISO
	AUD_DAT_MISO0
	AUD_DAT_MISO1
I2S0	*I2S0_LRCK*
	I2S0_BCK
	I2S0_MCK
	I2S0_DI
I2S1	*I2S1_LRCK*
	I2S1_BCK
	I2S1_MCK
	I2S1_DO
I2S2	*I2S2_LRCK*
	I2S2_BCK
	I2S2_MCK
	I2S2_DI
	I2S2_DI2
I2S3	*I2S3_LRCK*
	I2S3_BCK
	I2S3_MCK
	I2S3_DO
I2S5	*I2S5_LRCK*
	I2S5_BCK
	I2S5_MCK
	I2S5_DO

4.1.1 Introduction for I2S connection application

Mediatek with a high flexible design between HW function (IP: intellectual property) and Pad.

A relationship between IP and pad can use AUX_FUNC to select Pad connect to which IP

PCM interface:

Pin Name	Aux Func.0	Aux Func.2	Aux Func.3	Aux Func.6
PAD_EINT0	GPIO0	O:PCM0_SYNC		
PAD_EINT1	GPIO1	O*PCM0_CLK		
PAD_EINT2	GPIO2	O:PCM0_DO		
PAD_EINT3	GPIO3	I0:PCM0_DI		
PAD_DPI_D0	GPIO13		O:PCM0_SYNC	
PAD_DPI_D1	GPIO14		O*PCM0_CLK	
PAD_DPI_D2	GPIO15		O:PCM0_DO	
PAD_DPI_D3	GPIO16		I0:PCM0_DI	
PAD_MSDC1_CLK	GPIO29			B0*PCM1_CLK
PAD_MSDC1_DAT3	GPIO30			I0:PCM1_DI
PAD_MSDC1_CMD	GPIO31			B0:PCM1_SYNC
PAD_MSDC1_DAT0	GPIO32			O:PCM1_DO0
PAD_MSDC1_DAT2	GPIO33			O:PCM1_DO2
PAD_MSDC1_DAT1	GPIO34			O:PCM1_DO1

PMIC interface:

Pin Name	Aux Func.0	Aux Func.1	Aux Func.2	Aux Func.3
PAD_AUD_CLK_MOSI	GPIO136	O*AUD_CLK_MOSI	I0*AUD_CLK_MISO	
PAD_AUD_SYNC_MOSI	GPIO137	O*AUD_SYNC_MOSI	I0*AUD_SYNC_MISO	
PAD_AUD_DAT_MOSI0	GPIO138	O*AUD_DAT_MOSI0	I0*AUD_DAT_MISO0	
PAD_AUD_DAT_MOSI1	GPIO139	O*AUD_DAT_MOSI1	I0*AUD_DAT_MISO1	
PAD_AUD_CLK_MISO	GPIO140	I0*AUD_CLK_MISO	O*AUD_CLK_MOSI	
PAD_AUD_SYNC_MISO	GPIO141	I0*AUD_SYNC_MISO	O*AUD_SYNC_MOSI	
PAD_AUD_DAT_MISO0	GPIO142	I0*AUD_DAT_MISO0	O*AUD_DAT_MOSI0	
PAD_AUD_DAT_MISO1	GPIO143	I0*AUD_DAT_MISO1	O*AUD_DAT_MOSI1	

I2S interface:

Pin Name	Aux Func.0	Aux Func.2	Aux Func.6	Aux Func.7
PAD_EINT0	GPIO0		O*I2S3_MCK	
PAD_EINT1	GPIO1		O*I2S3_BCK	
PAD_EINT2	GPIO2		O:I2S3_LRCK	

PAD_EINT3	GPIO3		O:I2S3_DO	
PAD_EINT4	GPIO4	O*I2S0_MCK		
PAD_EINT5	GPIO5	B0*I2S0_BCK		
PAD_EINT6	GPIO6	B0:I2S0_LRCK		
PAD_EINT7	GPIO7	I0:I2S0_DI		
PAD_EINT11	GPIO11			O*I2S5_MCK
PAD_EINT12	GPIO12		I0:I2S2_DI2	O*I2S5_BCK
PAD_DPI_D0	GPIO13		O*I2S0_MCK	
PAD_DPI_D1	GPIO14		B0*I2S0_BCK	
PAD_DPI_D2	GPIO15		B0:I2S0_LRCK	
PAD_DPI_D3	GPIO16		I0:I2S0_DI	
PAD_DPI_D4	GPIO17		O*I2S3_MCK	
PAD_DPI_D5	GPIO18		O*I2S3_BCK	
PAD_DPI_D6	GPIO19		O:I2S3_LRCK	
PAD_DPI_D7	GPIO20		O:I2S3_DO	
PAD_DPI_D8	GPIO21		O*I2S2_MCK	
PAD_DPI_D9	GPIO22		O*I2S2_BCK	
PAD_DPI_D10	GPIO23		O:I2S2_LRCK	
PAD_DPI_D11	GPIO24		I0:I2S2_DI	
PAD_DPI_HSYNC	GPIO25		O*I2S1_MCK	
PAD_DPI_VSYNC	GPIO26		O*I2S1_BCK	
PAD_DPI_DE	GPIO27		O:I2S1_LRCK	
PAD_DPI_CK	GPIO28		O:I2S1_DO	
PAD_INT_SIM2	GPIO46			O:I2S5_LRCK
PAD_INT_SIM1	GPIO47			O:I2S5_DO

Pin Name	Aux Func.0	Aux Func.1	Aux Func.2	Aux Func.3	Aux Func.4
PAD_SPI_MI	GPIO85				O*I2S1_BCK
PAD_SPI_CSB	GPIO86				O:I2S1_LRCK
PAD_SPI_MO	GPIO87				O:I2S1_DO
PAD_SPI_CLK	GPIO88				O*I2S1_MCK
PAD_SRCLKENAI	GPIO89			O*I2S5_BCK	
PAD_PWM_A	GPIO90			O:I2S5_LRCK	
PAD_KPROW1	GPIO91			O:I2S5_DO	
PAD_KPCOL1	GPIO94		I0:I2S2_DI2	O*I2S5_MCK	
PAD_CAM_PDN0	GPIO97		O*I2S2_MCK		
PAD_CAM_PDN1	GPIO98		O*I2S2_BCK		
PAD_CAM_RST0	GPIO101		O:I2S2_LRCK		
PAD_CAM_RST1	GPIO102		I0:I2S2_DI		

PAD_AUD_CLK_MOSI	GPIO136			O*I2S1_MCK	
PAD_AUD_SYNC_MOSI	GPIO137			O*I2S1_BCK	
PAD_AUD_DAT_MOSI0	GPIO138			O:I2S1_LRCK	
PAD_AUD_DAT_MOSI1	GPIO139			O:I2S1_DO	
PAD_AUD_CLK_MISO	GPIO140			O*I2S0_MCK	
PAD_AUD_SYNC_MISO	GPIO141			O*I2S0_BCK	
PAD_AUD_DAT_MISO0	GPIO142			O:I2S0_LRCK	
PAD_AUD_DAT_MISO1	GPIO143			I0:I2S0_DI	
PAD_I2S1_BCK	GPIO170	O*I2S1_BCK	O*I2S3_BCK		O*I2S5_BCK
PAD_I2S1_LRCK	GPIO171	O:I2S1_LRCK	O:I2S3_LRCK		O*I2S5_LRCK
PAD_I2S1_DO	GPIO172	O:I2S1_DO	O:I2S3_DO		O:I2S5_DO
PAD_I2S1_MCK	GPIO173	O*I2S1_MCK	O*I2S3_MCK		O*I2S5_MCK
PAD_I2S2_DI	GPIO174	I0:I2S2_DI	I0:I2S0_DI		I0:I2S2_DI2

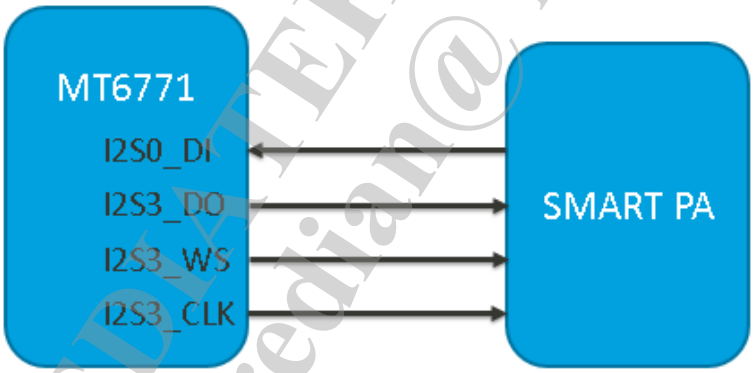
Figure 20. I2S GPIO AUX function

4.1.2 I2S Application

Set	Direction	Function	Note
I2S0	Input	Ext. DSP Smart PA	1. ADC input available to use
I2S1	Output	Ext. DAC MHL	1. MHL or ext. DAC available to use 2. Ext. DAC and MHL while using the same set of shared
I2S2	Input	Ext. ADC	1. MATV available to use 2. PinMux 2 places (MHL/SPI)
I2S3	Output	Ext. DSP Smart PA	1. DAC output available to use 2. PinMux 2 places (MHL/SPI)
PCM1	Bi-direction	Ext. MD	1. PinMux RF1_BSI 2. Setting and DT (MT6176) for the exclusive

	Pin	Scenario - 1	Scenario - 2	Scenario - 3
I2S0_ADC input	4	Smart PA - echo path	Audience - for D/L in	Smart PA - echo path
I2S1_DAC output	4	External DAC - 2ch DAC or 2ch MHL	Audience - for U/L in	External codec - Downlink
I2S2_ADC_input	5	External ADC	Audience - for U/L out	External codec - Uplink
I2S3_DAC output	4	Smart PA - output	Audience - for D/L out	Smart PA - output
I2S5_DAC output	4	3rd Party DAC/PA - output	Audience - for D/L out	
Master PCM1	4	External Modem (C2K)	External Modem (C2K)	External Modem (C2K)
Marge IF	4	MT6625	MT6625	MT6625
MTK proprietary Audio IF	3	MT6358	MT6358	x

For smart PA share pin application, please use I2S3 CLK/WS/DO and I2S0 DI.



4.1.3 Recommend Part List

- DAC
- HP buffer
- PLL

4.2 FM Integration Guide

4.2.1 Overview

There is an important peripheral for ALPS audio: FM Radio (FM RX). In this document, we will discuss the integration of this feature.

In this section we will discuss some background knowledge. In the following sections, we will discuss the reference design for each chip.

There is only one type for FM audio: digital I2S data path, but no analog line-in path anymore.

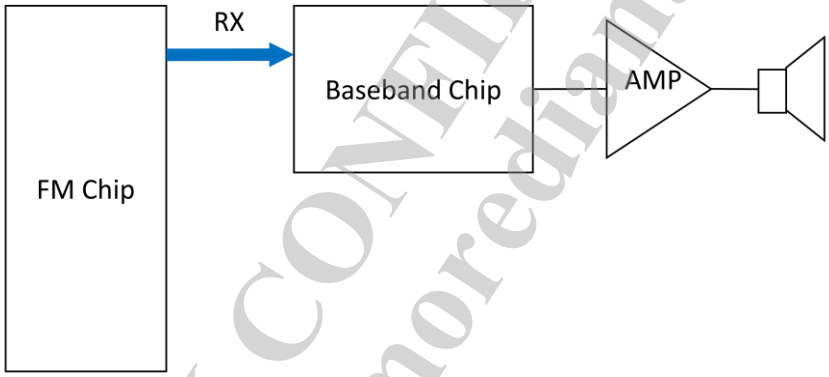


Figure 21. FM Hardware Architecture

In the figure, FM RX support I2S path only. There is no line-in path

4.2.2 FM Radio Description

FM Radio (FM RX) is a popular audio feature in smart phone devices. In this chapter, we will describe characteristics of FM RX, includes data path, audio stream type, sampling rate conversion and volume control method.

4.2.2.1 Audio Stream Playback via RX

When FM Radio Application is opened, users can turn on FM Radio and search the radio channel. Because the headphone line is seen as antenna, there ' re warning message pop upping if headphone not detected by accdet driver. However, user can skip the warning message and continue enable FM Radio by short antenna support. (MTK_FM_SHORT_ANTENNA_SUPPORT=yes in Project Configuration setting)

The audio stream type of FM RX is FM

4.2.2.2 Audio Sampling Rate Conversion via FM RX

For MT6771 + MT6631, the sampling rate of audio stream from FM is fixed at 32000 Hz. However, the sampling rate of playing and recording FM stream is 44100Hz. Hence, we need to apply hardware sampling rate conversion from 32000 Hz to 44100 Hz.

For M*T6771* + MT6630, the sampling rate of audio stream from FM Chip can be configured as 32000/44100/48000 Hz, and the sampling rate of playing and recording FM stream is 44100Hz. Hence, we just configure FM chip as 44100 Hz and don't need to apply software sampling rate conversion in AudioFlinger.

4.2.2.3 FM RX Data Path – Digital (I2S Mode)

There're two digital paths implemented in host chip: Direct I2S Data path and In-direct I2S Data path.

- **Direct I2S Data Path**

  This kind of data flow is applied on headphone, headset, and speaker device for the purpose of low power playback of FM RX. Please refer to the following figure for I2S data path.

  Since there is a high-pass filter in FM DSP to remove DC value, we can just use direct mode to play FM radio in speaker mode.

  Playback path (blue): Player configures Audio HW only. Audio digital data plays directly through digital hardware.

  Record path (Red): Recorder gets PCM data I2S. After encoding, it is stored into SD card.

  Audio data is transferred from MT6625 by I2S interface to audio hardware output device directly.

  If there's other audio stream played at the same time, the audio stream will be mixed with FM data at audio digital hardware side.

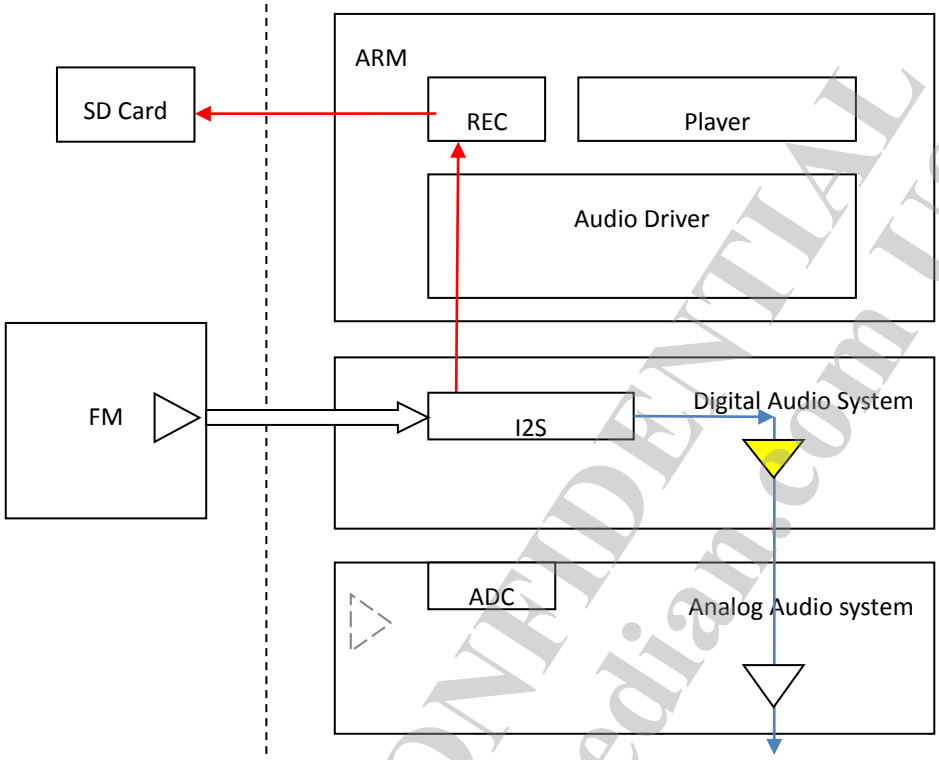


Figure 22. FM Direct I2S Data Path

- **Direct I2S Gain Setting**

  Playback Path related: [FM Internal Gain], [Audio Digital Hw Gain], [Analog PGA Gain]

  End User tunes: [Audio Digital Hw Gain] Only

  Platform customization tunes: [FM Internal Gain], [Analog PGA Gain]

  Record Path: [FM Internal Gain] only.

  End User cannot do any fine tune. (This is reasonable, if playback gain equals to zero, we can still record sound)

- **In-Direct I2S Data Path**

  This kind of data flow is applied on any device except headphone, headset, and speaker device, like WFD, USB audio and BT A2DP. We use in-direct I2S data path for FM RX playback for the purpose of audio quality.

  Please refer to the following figure for I2S data path.

  Playback path (blue): Player plays an important role. It gets PCM data from I2S and plays the data just like typical file playback.

  Record path (Red): Recorder gets PCM data I2S. After encoding, it is stored into SD card.

  Audio data is transferred from FM chip by I2S interface to host chip. Processor will mix all available audio streams. After mixed and re-sampled, audio data will be played by audio hardware.

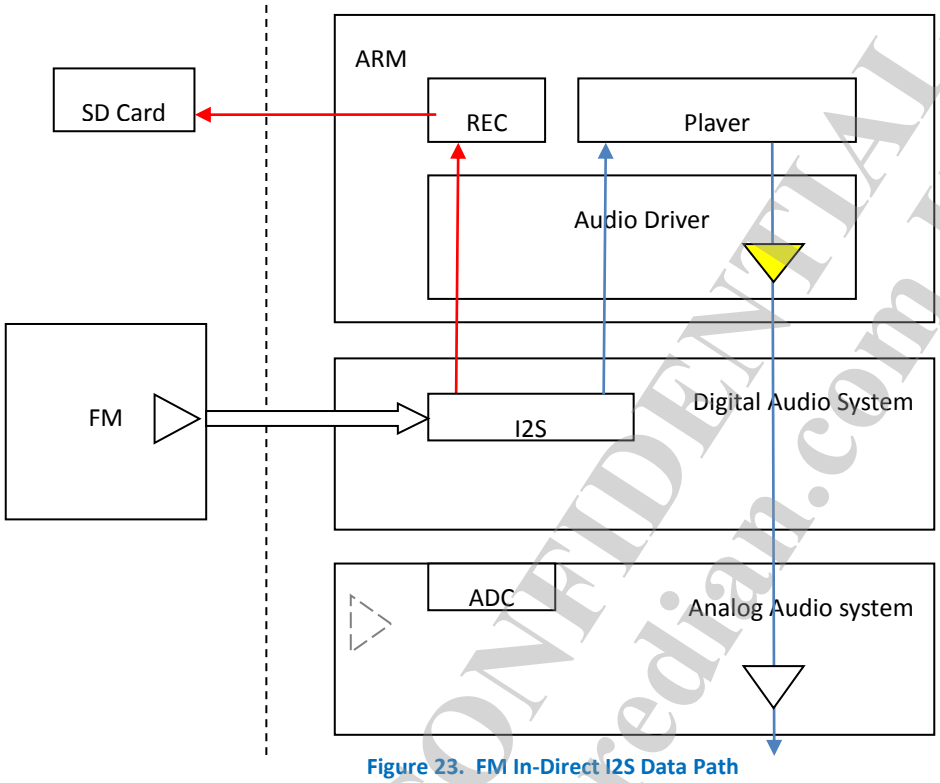


Figure 23. FM In-Direct I2S Data Path

4.2.2.4 In-Direct I2S Gain Setting

Playback Path related: [FM Internal Gain], [Audio Digital Stream Gain], [Analog PGA Gain]

End User tunes: [Audio Digital Stream Gain] Only

Platform customization tunes: [FM Internal Gain], [Analog PGA Gain]

Record Path: [FM Internal Gain] only

End User cannot do any fine tune. (This is reasonable, if playback gain equals to zero, we can still record sound)

4.2.2.5 I2S Volume Control

The volume is controlled by degrading digital gain of FM Stream in Audio Driver. For Direct I2S Data Path, Audio Digital HW Gain is controlled to be degraded. For In-Direct I2S Data Path, Audio Digital Stream Gain is controlled to be degraded.

4.2.2.6 Audio Channel

The audio stream from FM Chip to host chip is stereo channel.

4.2.2.7 Power Mode

Direct I2S Data Path consumes less power than In-Direct I2S Data Path while playback because audio data not runs

4.2.3 Audio Stream Playback via FM Tx

We don't support FM Transmitter (FM TX) for current solution.

4.2.4 Project Configuration

FM related configurations and control flow are controlled by AP side.

File Name	*Path*	*Related Customizaiton*
ProjectConfig.mk	\device\mediatek\(%porj)\	*Configure Digital Data Path for FM*

FM related settings:

File Name	*value*	*Note*
MTK_FM_SUPPORT	yes or no	
MTK_FM_CHIP	MT6625_FM MT6630_FM	FM Chip Name
MTK_FMRADIO_APP	yes or no	
MTK_FM_RX_SUPPORT	yes or no	
MTK_FM_TX_SUPPORT	yes or no	Disable for MT6625 Enable for MT6630
MTK_FM_RX_AUDIO	*FM_DIGITAL_INPUT*	
MTK_FM_TX_AUDIO	FM_DIGITAL_OUTPUT	
MTK_FM_SHORT_ANTENNA_SUPPORT	*yes or no*	
MTK_BT_FM_OVER_BT_VIA_CONTROLLER	yes or no	Default enable
MTK_FM_RECORDING_SUPPORT	yes or no	

4.3 BT Control

For MT6771 and MT6631 BT, audio driver direct access BT firmware to read/write BT audio raw data, and CVSD and mSBC encoder/decoder are implemented on AP-side and MD-side processors.

For MT6771 and MT6630 BT, merge interface is a proprietary interface used to transfer pcm and I2S interface concurrently.

4.3.1 Software Architecture

MT6631 BT

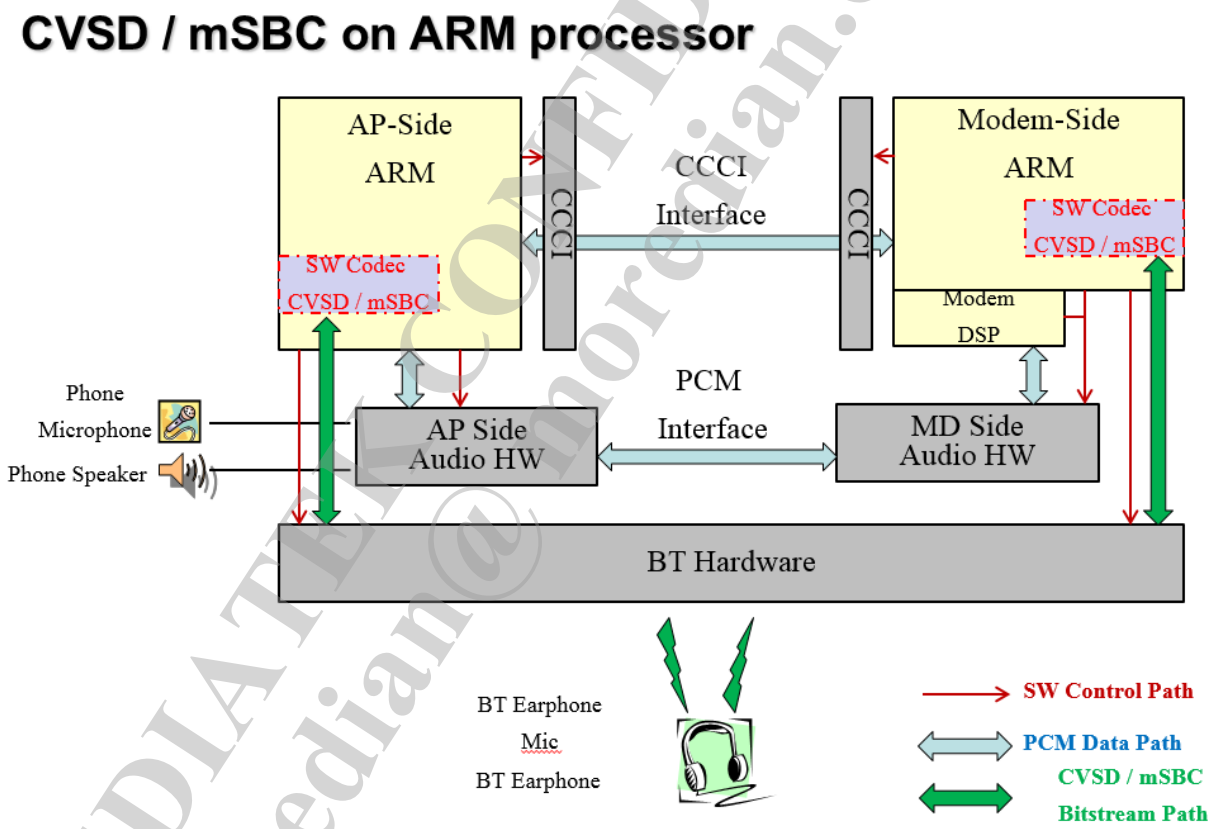


Figure 24. Audio Block Diagram with BT 6631

MT6630 BT

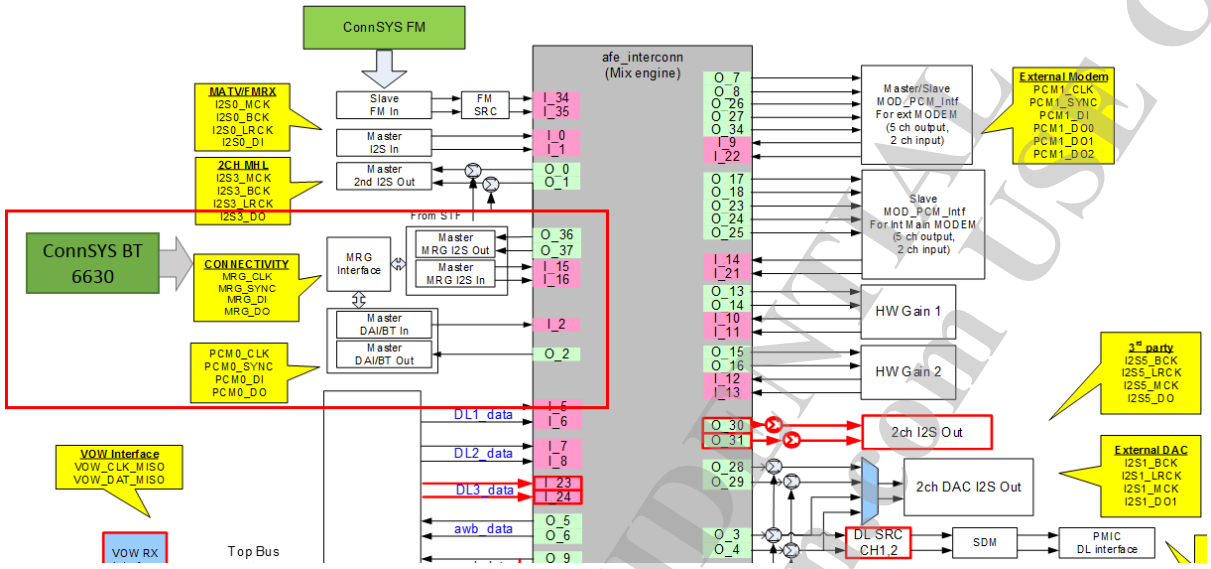


Figure 25. Audio Block Diagram with BT 6630

4.3.2 BT Codec

- BT Framework using setParameter to Audio HAL
- NB : “bt_wbs=off”
  - Sample rate = 8kHz
  - 使用 CVSD encode/decode
- WB: “bt_wbs=on”
  - Sample rate = 16k Hz
  - 使用 MSBC encode/decode

4.3.3 2G/3G/4G Phone Call with BT earphone (MD ARM)

4.3.3.1 BT ConnSYS MT6631

➢ **Uplink Data Path**

[MT6631]BT earphone (BT Mic) → BT HW → MD CVSD decoder → DSP/Network

➢ **Downlink Data Path**

[MT6631]Network/DSP→ MD CVSD encoder → BT HW → BT earphone

- Uplink Data Path →
  - BT earphone (BT Mic) → BT HW → MD-ARM (CVSD Decoder) → DSP/Network
- Downlink Data Path →
  - Network/DSP→ MD-ARM (CVSD Encoder) → BT HW → BT earphone
- Voice Record →
  - Audio Encoder & File Writer are in AP side
- Background Sound →
  - Modem is busy. AP side should make sound.

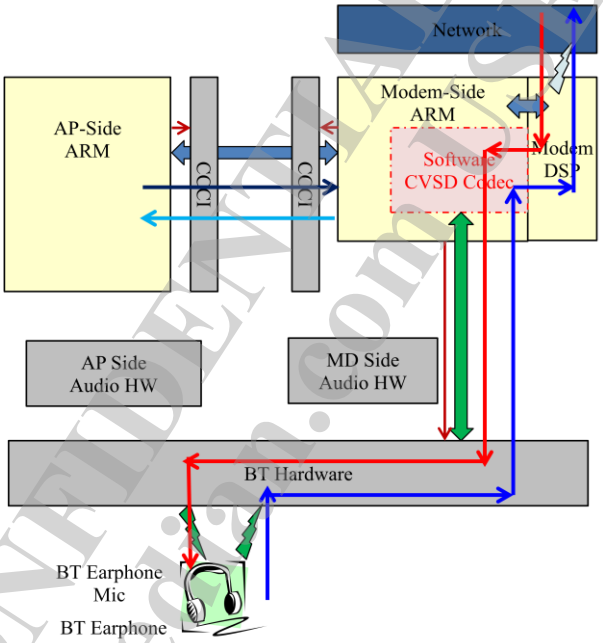


Figure 26. MD ARM 2G/3G/4G Phone Call with BT (MT6631)

4.3.3.2 BT ConnSYS MT6630

➢ **Uplink Data Path**

[MT6630]BT earphone (BT Mic) → BT HW → Merge Interface (PCM) → DSP/Network

➢ **Downlink Data Path**

[MT6630]Network/DSP→ Merge Interface (PCM) → BT HW → BT earphone

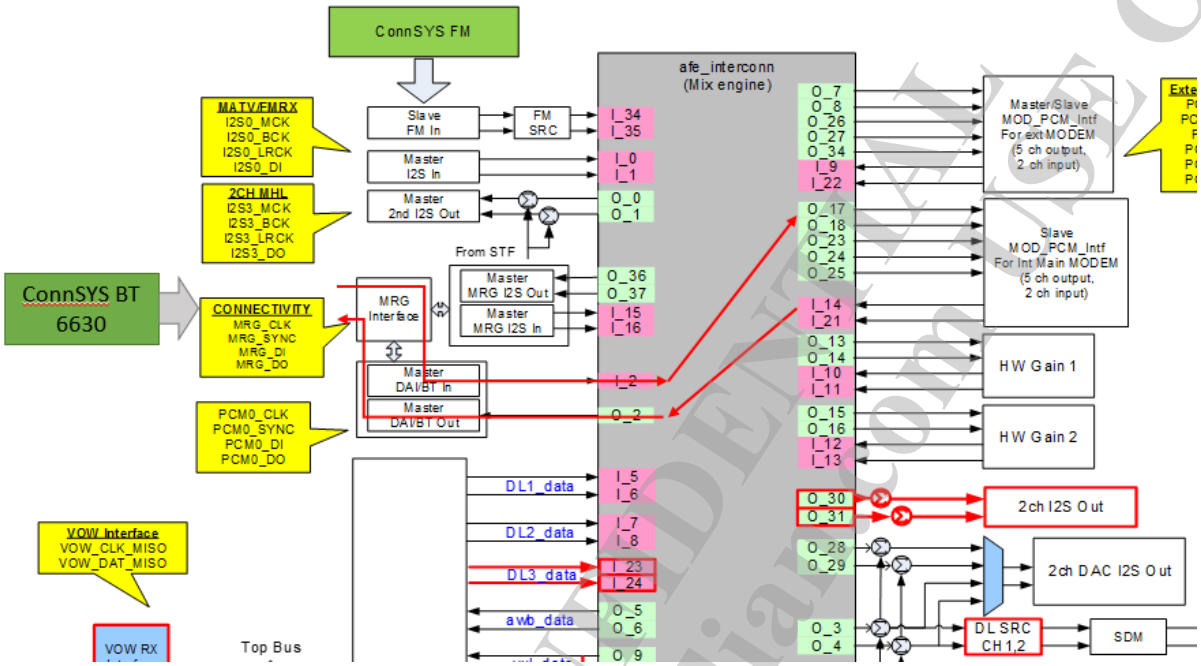


Figure 27. MD ARM 2G/3G/4G Phone Call with BT (MT6630)

4.3.4 VoIP Call with BT earphone (AP ARM)

4.3.4.1 BT ConnSYS MT6631

- **Uplink Data Path**

  [MT6631]BT earphone (BT Mic) → BT HW → AP CVSD decoder → DSP/Network

- **Downlink Data Path**

  [MT6631]Network/DSP→ AP CVSD encoder → BT HW → BT earphone

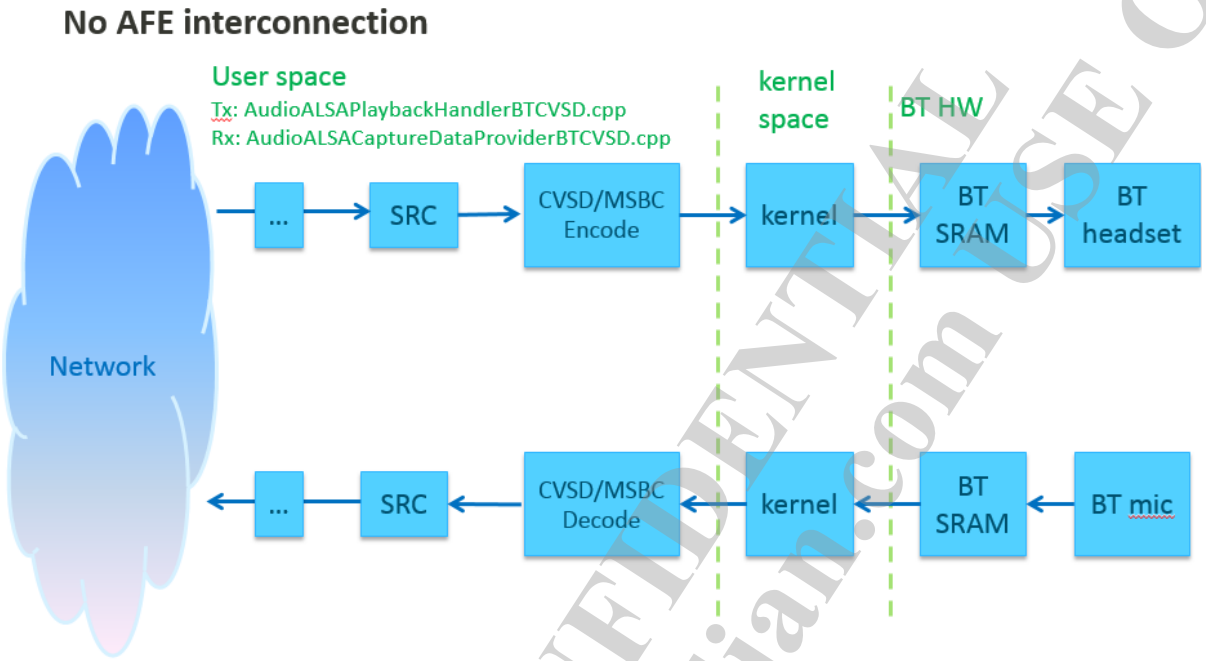


Figure 28. AP ARM VOIP Call with BT (MT6631)

- **Downlink Data Path**
- BT TX control flow
  - AudioALSAPlaybackHandlerBTCVSD::write

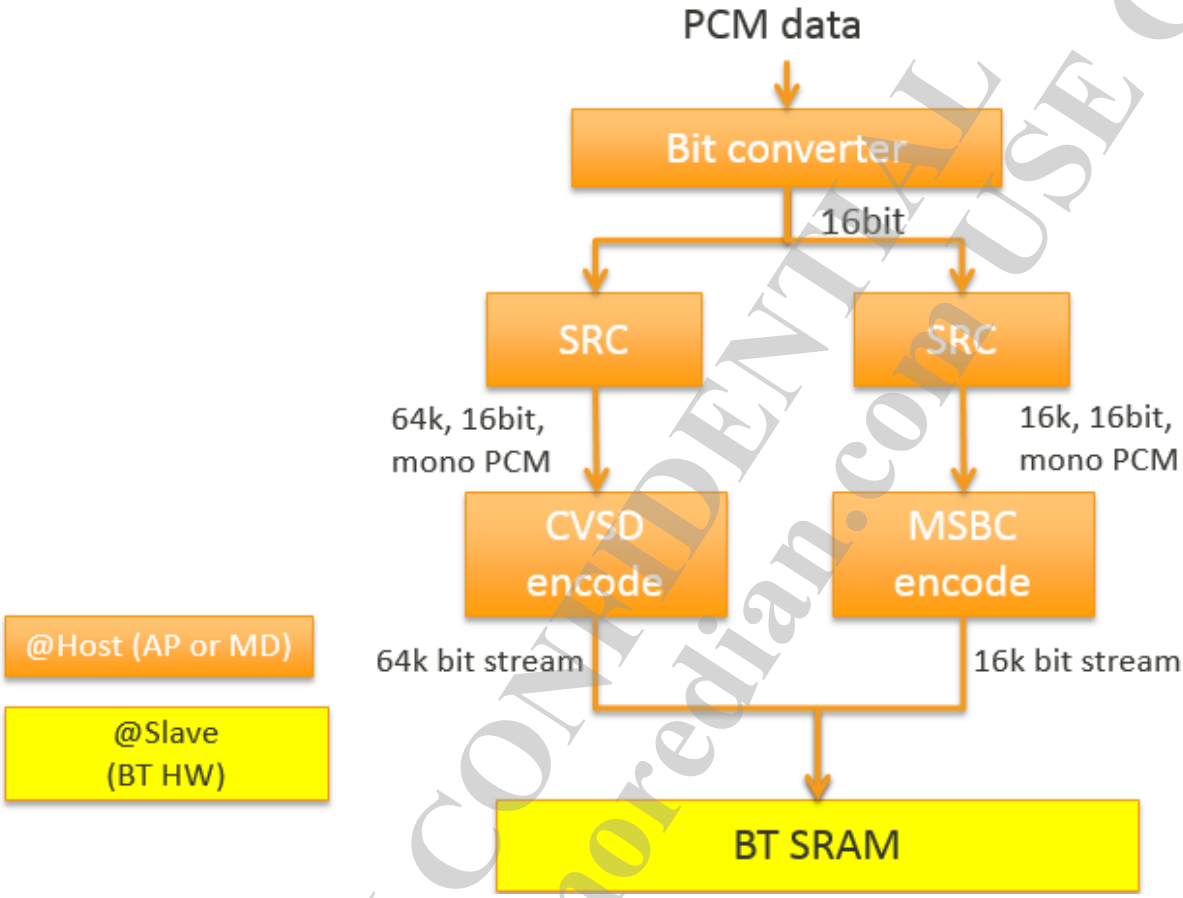


Figure 29. VoIP BT downlink control flow (BT 6631)

- **Uplink Data Path**
- BT RX control flow
  - AudioALSACaptureDataProviderBTCVSD::readDataFromBTCVSD

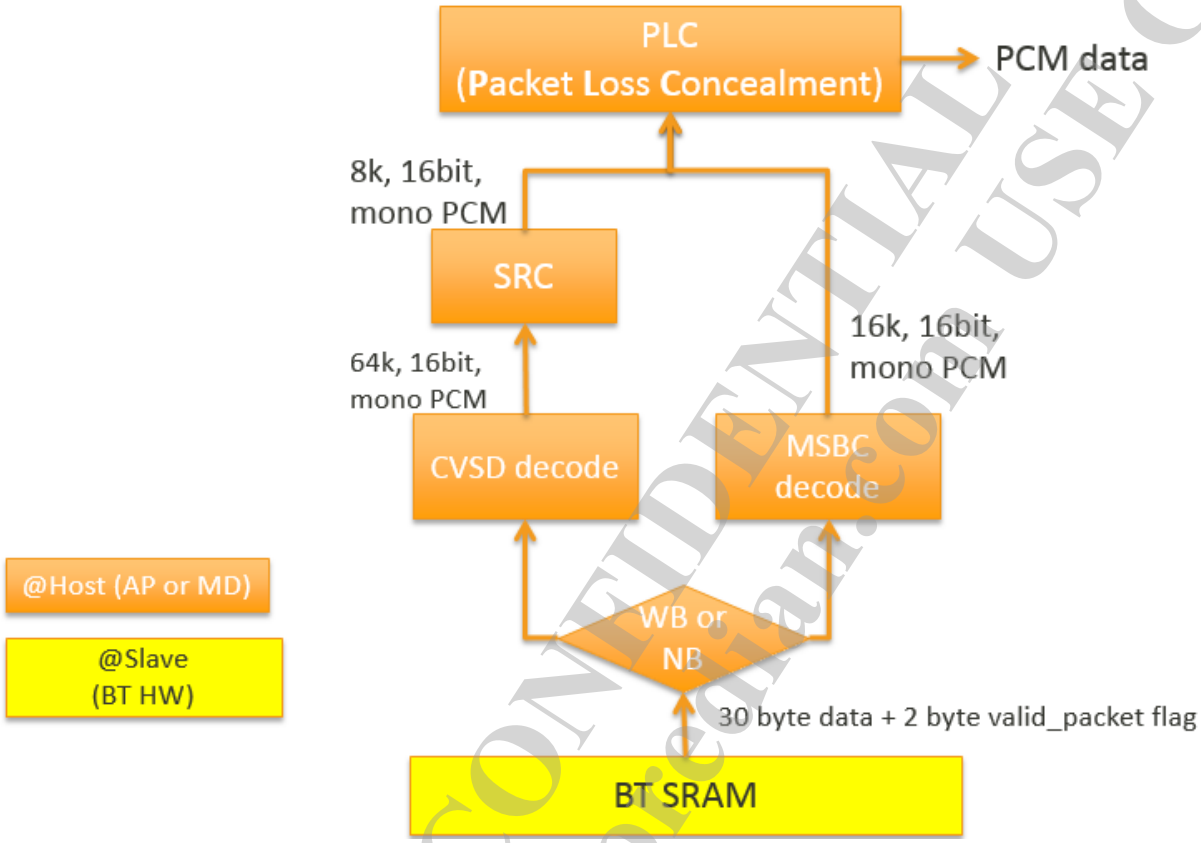


Figure 30. VoIP BT uplink control flow (BT 6631)

4.3.4.2 BT ConnSYS MT6630

- **Uplink Data Path**

  [MT6630]BT earphone (BT Mic) → BT HW → AP-ARM (Merge interface) → Network

- **Downlink Data Path**

  [MT6630]Network → AP-ARM (Merge interface) → BT HW → BT earphone

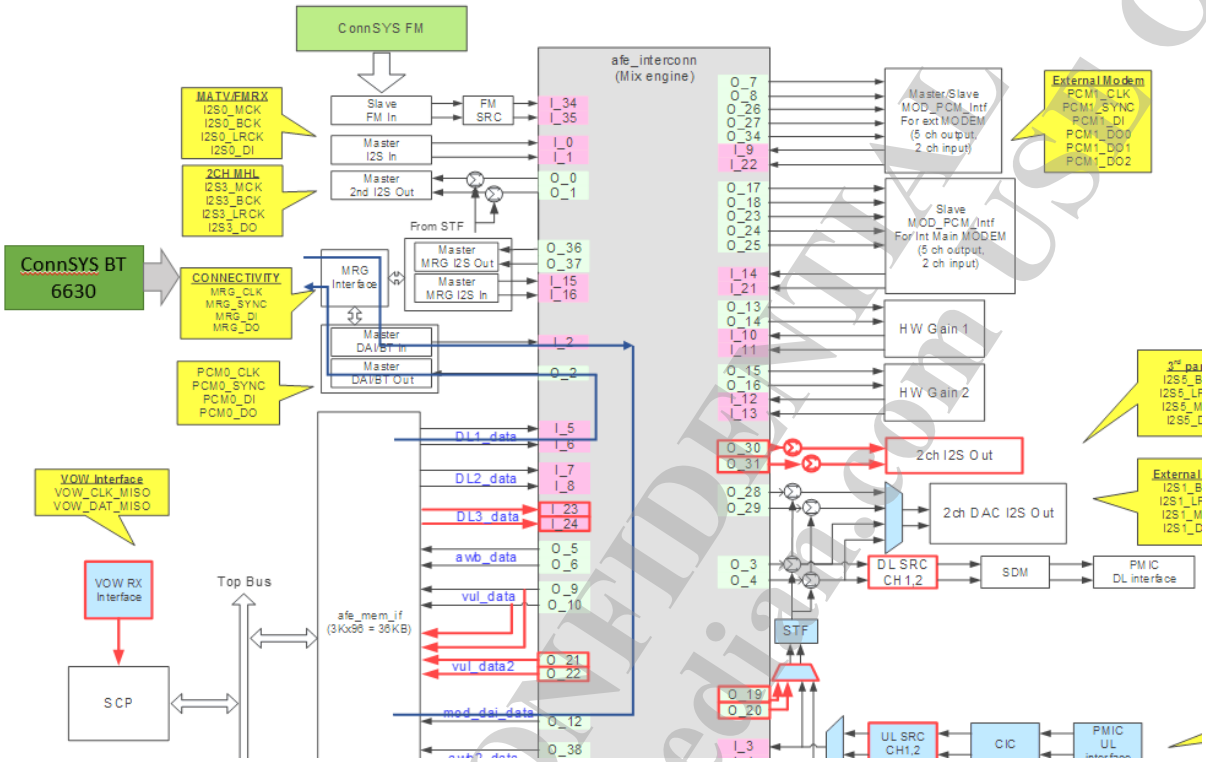


Figure 31. AP ARM VOIP Call with BT (MT6630)

4.3.5 BT ALSA(Advanced Linux Sound Architecture) in kernel

- **introduction**

  Modify original BTCVSD kernel driver to ALSA, which meet kernel standard.

- **Kernel config**

  # CONFIG_MTK_BTCVSD is not set
  CONFIG_SND_SOC_MTK_BTCVSD=y

- HAL feature option

  LOCAL_CFLAGS += -DMTK_SUPPORT_BTCVSD_ALSA

- **Platform driver name**

"BTCVSD_RX" for uplink; "BTCVSD_TX" for downlink.

5 Audio Test Suite

5.1 Audio Loopback

5.1.1 Loopback Introduction

5.1.1.1 Overview

Loopback refers to the routing of data flows back to the originating devices without modifications. It is used to check whether the I/O devices, audio hardware, or modem/DSP works or not. The method to check the devices is to open the loopback function, and then blow/speak to the microphone. Finally the sound comes out from the receiver, earphone, or speaker.

5.1.1.2 Loopback Functions

There are two main types of loopback functions:

- **AFE Loopback**

  The sound only goes through audio HW
- **Acoustic Loopback**

  The sound not only goes through audio HW but also goes through Modem and DSP

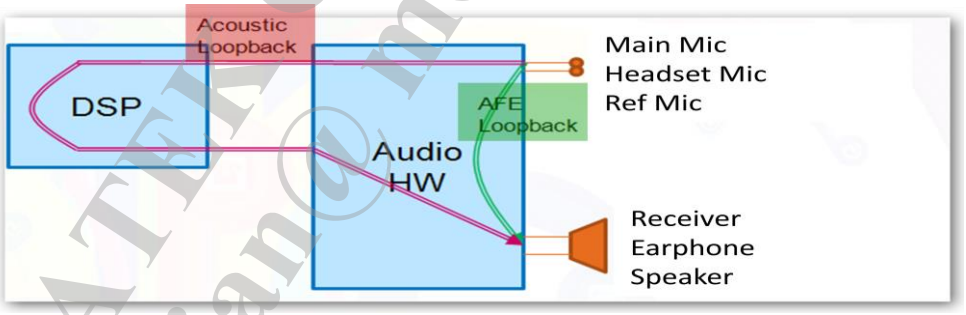


Figure 32. Audio Loopback Block Diagram

Main Types	Loopback Types	Input Source	Output Device	Speech Enhance-ment	DMNR
AFE Loopback Go through Audio HW only	AP_MAIN_MIC_AFE_LOOPBACK	**Main mic**	**We can assign any one of:** 1. **Receiver** 2. **Earphone** 3. **Speaker** **If not assigned, then:** If earphone plugged in: use **Earphone** else: use **Receiver**	Off	Off
	AP_HEADSET_MIC_AFE_LOOPBACK	**Headset mic**			
	AP_REF_MIC_AFE_LOOPBACK	**Ref mic**			
	AP_3RD_MIC_AFE_LOOPBACK	**3rd mic**			
Acoustic Loopback Go into Modem & DSP	MD_MAIN_MIC_ACOUSTIC_LOOPBACK	**Main mic**		On	Off
	MD_HEADSET_MIC_ACOUSTIC_LOOPBACK	**Headset mic**			
	MD_REF_MIC_ACOUSTIC_LOOPBACK	**Ref mic**			
	MD_3RD_MIC_ACOUSTIC_LOOPBACK	**3rd mic**			
	MD_DUAL_MIC_ACOUSTIC_LOOPBACK_WITHOUT_DMNR	**Dual mic**			*Off*
	MD_DUAL_MIC_ACOUSTIC_LOOPBACK_WITH_DMNR	**Dual mic**			*On*

Notice that:

1) In acoustic loopback, the data will be applied with speech enhancement in DSP and then pass to DL path.
2) To use ref mic and dual mic, we should set the MTK_DUAL_MIC_SUPPORT as yes in project configurations.

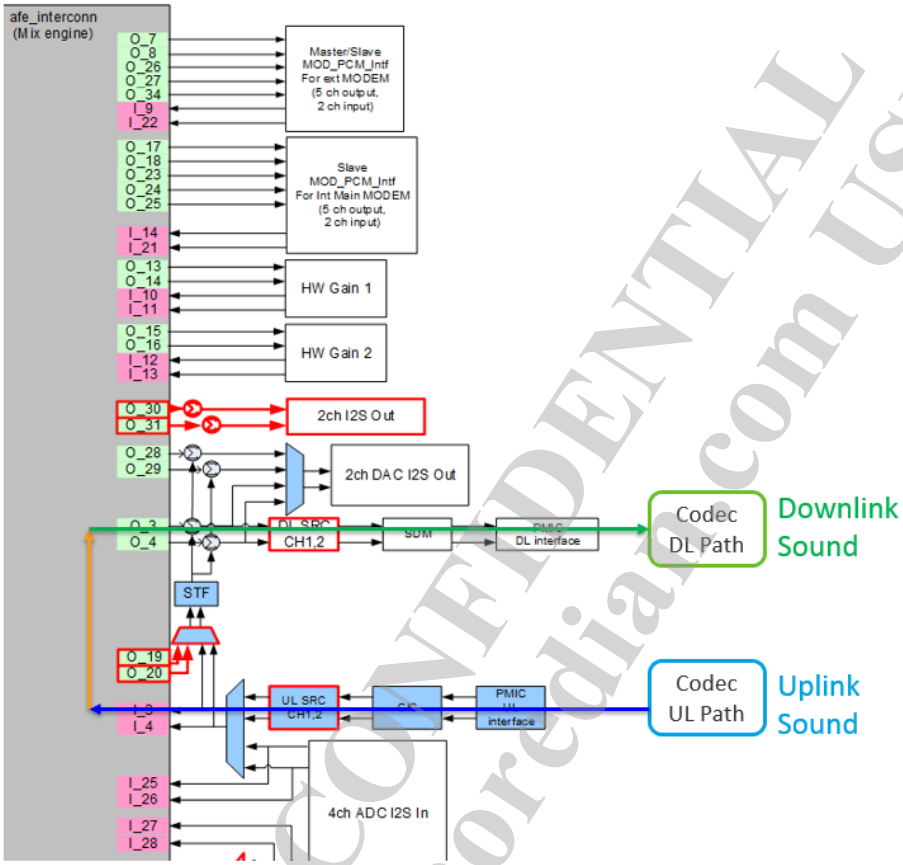


Figure 33. Data Flow of AFE loopback

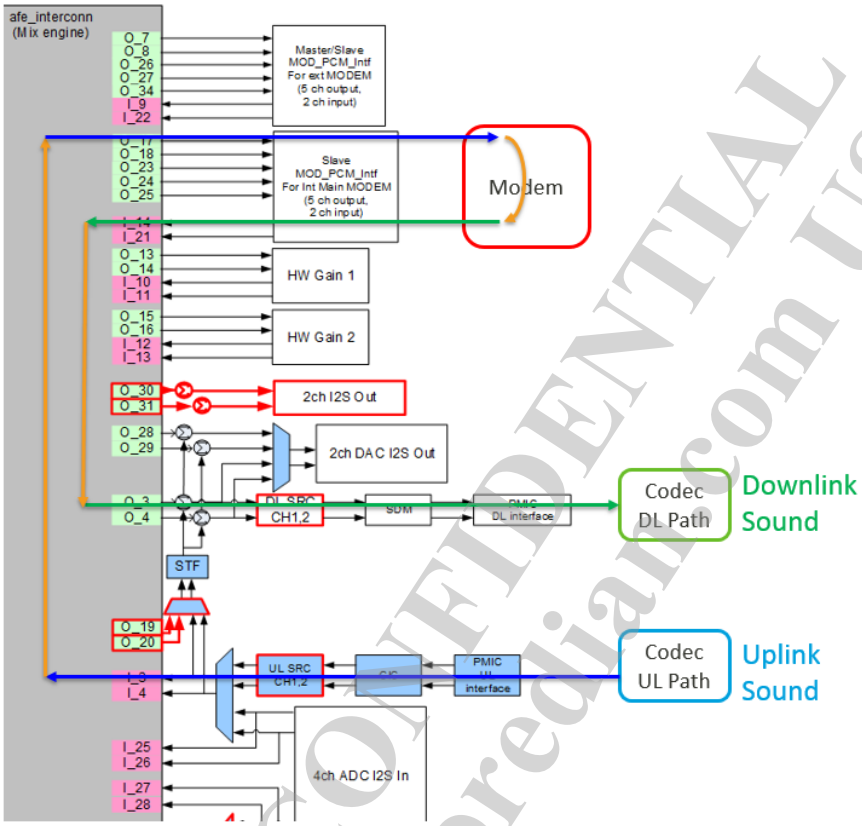


Figure 34. Data Flow of Acoustic Loopback

5.1.2 How to Turn on/off Loopback Function

There are two methods to turn on/off loopback function, one is by set parameters function, the other is MM command.

5.1.2.1 Set Parameters

We can simply turn on/off loopback functions via AudioSystem.setParameters("key=value"), where the key is "SET_LOOPBACK_TYPE". Besides, the value of type is *"Type"* or *"Type, OutputDevice"*. The *Type* is the loopback types, and the *OutputDevice* is used to specify a output device, such as speaker, receiver, and earphone.

```
enumloopback_t {
    NO_LOOPBACK                                 = 0,

    // AFE Loopback
    AP_MAIN_MIC_AFE_LOOPBACK                    = 1,
    AP_HEADSET_MIC_AFE_LOOPBACK                 = 2,
    AP_REF_MIC_AFE_LOOPBACK                     = 3,
AP_3RD_MIC_AFE_LOOPBACK                     = 4,

    // Acoustic Loopback
    MD_MAIN_MIC_ACOUSTIC_LOOPBACK               = 21,
    MD_HEADSET_MIC_ACOUSTIC_LOOPBACK            = 22,
    MD_DUAL_MIC_ACOUSTIC_LOOPBACK_WITHOUT_DMNR  = 23,
    MD_DUAL_MIC_ACOUSTIC_LOOPBACK_WITH_DMNR     = 24,
    MD_REF_MIC_ACOUSTIC_LOOPBACK                = 25,
    MD_3RD_MIC_ACOUSTIC_LOOPBACK                = 26,
};
```

Turn off current loopback

AP side control digital/analog registers itself to achieve AFE loopback functions. Without MD, we could simply justify whether audio hw & i/o devices are fine or not first.

AP side control digital/analog registers itself and let the data of sound go through modem side to achieve Acoustic loopback functions. We can use these types of loopbackfunctions to verify modem and speech enhancement function.

```
enumloopback_output_device_t {
    LOOPBACK_OUTPUT_RECEIVER = 1,
    LOOPBACK_OUTPUT_EARPHONE = 2,
    LOOPBACK_OUTPUT_SPEAKER  = 3,
};
```

Assign the output device in Loopback test

Figure 35. Enumerations of Value/OutputDevice of SetParameters()

```
enumloopback_t {
    NO_LOOPBACK                                 = 0,

    // AFE Loopback
    AP_MAIN_MIC_AFE_LOOPBACK                    = 1,
    AP_HEADSET_MIC_AFE_LOOPBACK                 = 2,
    AP_REF_MIC_AFE_LOOPBACK                     = 3,
AP_3RD_MIC_AFE_LOOPBACK                     = 4,

    // Acoustic Loopback
    MD_MAIN_MIC_ACOUSTIC_LOOPBACK               = 21,
    MD_HEADSET_MIC_ACOUSTIC_LOOPBACK            = 22,
    MD_DUAL_MIC_ACOUSTIC_LOOPBACK_WITHOUT_DMNR  = 23,
    MD_DUAL_MIC_ACOUSTIC_LOOPBACK_WITH_DMNR     = 24,
    MD_REF_MIC_ACOUSTIC_LOOPBACK                = 25,
    MD_3RD_MIC_ACOUSTIC_LOOPBACK                = 26,
};
```

The output device :
If earphone plugged in:
use **Earphone**
else:
use **Receiver**

```
// Enable Main Mic AFE Loopback
AudioSystem.setParameters("SET_LOOPBACK_TYPE=1");
// Testing ..
// DisableCurrentLoopback Function
AudioSystem.setParameters("SET_LOOPBACK_TYPE=0");
```

Figure 36. Example of Not assigning OutputDevice by setParameters()

```
enumloopback_t {
    NO_LOOPBACK                                 = 0,

    // AFE Loopback
    AP_MAIN_MIC_AFE_LOOPBACK                    = 1,
    AP_HEADSET_MIC_AFE_LOOPBACK                 = 2,
    AP_REF_MIC_AFE_LOOPBACK                     = 3,
AP_3RD_MIC_AFE_LOOPBACK                     = 4,

    // Acoustic Loopback
    MD_MAIN_MIC_ACOUSTIC_LOOPBACK               = 21,
    MD_HEADSET_MIC_ACOUSTIC_LOOPBACK            = 22,
    MD_DUAL_MIC_ACOUSTIC_LOOPBACK_WITHOUT_DMNR  = 23,
    MD_DUAL_MIC_ACOUSTIC_LOOPBACK_WITH_DMNR     = 24,
    MD_REF_MIC_ACOUSTIC_LOOPBACK                = 25,
MD_3RD_MIC_ACOUSTIC_LOOPBACK                = 26,
};
```

```
enumloopback_output_device_t {
    LOOPBACK_OUTPUT_RECEIVER = 1,
    LOOPBACK_OUTPUT_EARPHONE = 2,
    LOOPBACK_OUTPUT_SPEAKER  = 3,
};
```

```
// Enable Main Mic AFE Loopback
AudioSystem.setParameters("SET_LOOPBACK_TYPE=3,3");
// Testing …
// DisableCurrentLoopback Function
AudioSystem.setParameters("SET_LOOPBACK_TYPE=0");
```

Figure 37. Example of assigning OutputDevice by setParameters()

The figures above depict the examples of assigning a specific output device and not assigning the output device. If we don't assign an output device, the output device will be receiver by default unless an earphone is plugged. If an earphone is plugged, the output device will be the earphone.

5.1.2.2 MM Command Handler

We can also MM Command Handler to turn on/off loopback functions. The MM Command Handler will eventually call *setParameters()* function to turn on/off loopback function.

To use MM Command Handler, we need to enable MM Command first. There are two steps:

1) *adb shell setprop persist.sys.usb.config mtp,adb,acm;*

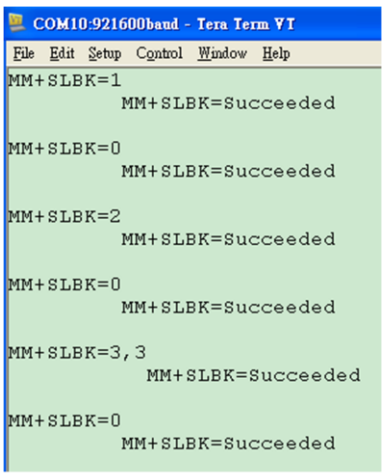
   This is a one time command; we don't need to do it again even we reboot the device. We need to set the command again only after we download flash to the devices.

2) *adb shell start audio-daemon*

   We need to set this command each time when we reboot the device.

After enabling the MM Command, what we need to do is open a hyper-terminal or a tera-term or other similar application. We can then use the application to send commands to the device in order to do loopback test.

The following figure is the example of turning on/off loopback test through MM Command Handler. The command format is "*MM+SLBK=value, device*", where MM is MM command; SLBK means "*Set Loopback Type*"; value means the type of loopback; device means the specified output device. Again, if we only set the type of loopback, the default device is receiver, and if there is an earphone has been plugged in, the default device is earphone. If the value is zero, for example "*MM+SLBK=0*", the loopback test will be turned off.




```
enumloopback_t {
    NO_LOOPBACK                                 = 0,

    // AFE Loopback
    AP_MAIN_MIC_AFE_LOOPBACK                    = 1,
    AP_HEADSET_MIC_AFE_LOOPBACK                 = 2,
AP_REF_MIC_AFE_LOOPBACK                     = 3,
   AP_3RD_MIC_AFE_LOOPBACK                      = 4,

    // Acoustic Loopback
    MD_MAIN_MIC_ACOUSTIC_LOOPBACK               = 21,
    MD_HEADSET_MIC_ACOUSTIC_LOOPBACK            = 22,
    MD_DUAL_MIC_ACOUSTIC_LOOPBACK_WITHOUT_DMNR  = 23,
    MD_DUAL_MIC_ACOUSTIC_LOOPBACK_WITH_DMNR     = 24,
    MD_REF_MIC_ACOUSTIC_LOOPBACK                = 25,
MD_3RD_MIC_ACOUSTIC_LOOPBACK                = 26,
};
```

```
enumloopback_output_device_t {
    LOOPBACK_OUTPUT_RECEIVER = 1,
    LOOPBACK_OUTPUT_EARPHONE = 2,
    LOOPBACK_OUTPUT_SPEAKER  = 3,
};
```

Figure 38. Example of loopback test by MM Command Handler

5.1.3 The Effect of Each Loopback Function

Loopback Types	Note
AP_MAIN_MIC_AFE_LOOPBACK	Blow to **main mic**, should hear sound from output source
AP_HEADSET_MIC_AFE_LOOPBACK	Blow to **headset mic**, should hear sound from output source
AP_REF_MIC_AFE_LOOPBACK	Blow to **ref mic**, should hear sound from output source
AP_3RD_MIC_AFE_LOOPBACK	Blow to **3rd mic**, should hear sound from output source
MD_MAIN_MIC_ACOUSTIC_LOOPBACK	*Speak* to **main mic**, should hear sound from output source with slow delay
MD_HEADSET_MIC_ACOUSTIC_LOOPBACK	*Speak* to **headset mic**, should hear sound from output source with slow delay
MD_REF_MIC_ACOUSTIC_LOOPBACK	*Speak* to **ref mic**, should hear sound from output source with slow delay
MD_3RD_MIC_ACOUSTIC_LOOPBACK	*Speak* to **3rd mic**, should hear sound from output source with slow delay
MD_DUAL_MIC_ACOUSTIC_LOOPBACK_*WITHOUT_DMNR*	*Speak* to main mic, should hear sound **(without distortion)** from output source with some delay.

MD_DUAL_MIC_ACOUSTIC_LOOPBACK_*WITH_DMNR*	*Speak* to **main mic**, should hear sound **(with distortion)** from output source with some delay. **When we turn on DMNR, the sound must be similar to the one without DMNR.**

5.1.4 Adjust Acoustic Loopback Delay Time

We can simply adjust acoustic loopback delay time via AudioSystem.setParameters("key=value") where the key is "SET_LOOPBACK_MODEM_DELAY_FRAMES" and the value is delay frame number (1 frame = 20ms). The default delay frame number is 12 frames (that is, 240 ms). The max delay frame number allowed is 32 and the minimum one is 0. Notice that we need to assign the delay frame number before acoustic loopback is enabled.

```
// Assign delay frame for modem loopback
AudioSystem.setParameters("SET_LOOPBACK_MODEM_DELAY_FRAMES=32");

// Enable Main Mic Acoustic Loopback with speaker out
AudioSystem.setParameters("SET_LOOPBACK_TYPE=21,3");

// Testing …

// DisableCurrent Loopback Function
AudioSystem.setParameters("SET_LOOPBACK_TYPE=0");
```

Figure 39. Example: Adjust Acoustic Loopback Delay Time

5.1.5 Limitations

There are some limitations in loopback test:

- **Setup Devices**

  The I/O devices must be ready before starting loopback and we should not change the device during loopback test. For example, if we want the sound comes out from earphone, we need to plug in first before starting loopback; if we want to set HEADSET_MIC_XXX_LOOPBACK, we need to plug in a headset mic first; and vice versa. Besides, please don't use reference mic or dual mic when the phone only support single mic.

- **During Testing**

  Not to make a phone call, record, play music and anything related to audio behavior during Loopback testing.

- **Flight Mode**

  Not to use Acoustic Loopback in flight mode since the modem side doesn't work

- Output Sound

If the output sound is not large enough, trying to use speaker instead of the original device. For example, the original scenario is "ref mic + receiver (if no earphone is plugged in)", the program is *AudioSystem.setParameters("SET_LOOPBACK_TYPE=3")*. We can change the scenario to "ref mic + speaker", and the program becomes *AudioSystem.setParameters("SET_LOOPBACK_TYPE=3,**3**")*.

5.2 Factory Mode

We can press power key and volume-down key simultaneously when booting-up the device. Then we can enter the factory mode. The following figure is the factory mode menu.

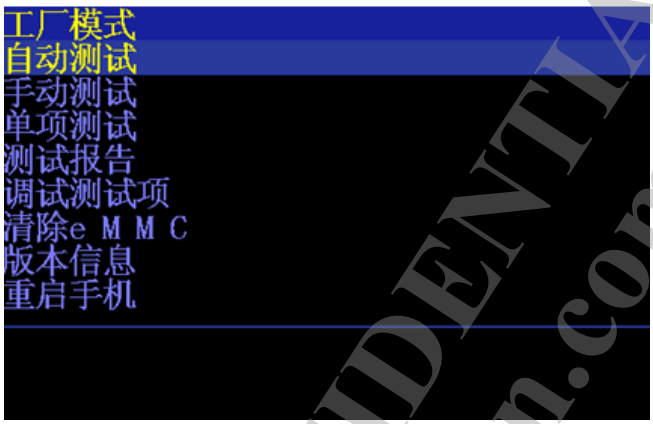


Figure 40. Factory Mode Menu

There three kinds of factory mode test: auto test, manual test, and item test, we’ll have descriptions in the following sections.

5.2.1 Auto Test

When we select the auto test item, the following test items will be performed atomically and the results will be shown in the test report. The auto test items are "Touch Panel Auto Test", "eMMC", "Memory Card", "Signaling Test", "RTC", "Loopback-PhoneMic_SpeakerLR", "Receiver", "Main Camera", "Sub Camera", "GPS", "FM Radio", "Bluetooth", “Wi-Fi", and "Battery & Charger"

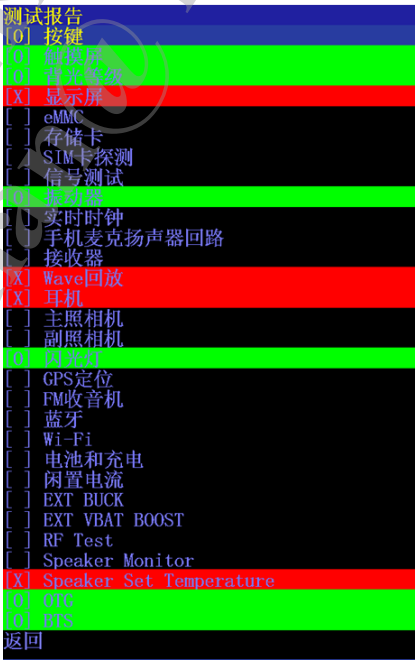


Figure 41. Factory Mode AutoTest Menu

5.2.2 Manual Test

The manual test is similar to the auto test. The test cases in manual test will be executed one by one. The only difference is that the user needs to have a feedback to each test item. For example, in “Key test”, the user needs to press the volume key and power key.

After all predefined test cases are executed; the test results will be shown in the test report, too.

5.2.3 Item Test

We can also verify a specified test item only by using item test. The steps are to enter item test and then select the test item you want. We’ll introduce audio related test items here.

Figure 42. Factory Mode Item Test Menu

- **Loopback-PhoneMic_SpeakerLR**

  This item is used to check if speaker and phone mic work. We can hear a 1 kHz tone from speaker when performing this test. The result will be judged automatically and shown in test report.

- **Receiver**

  This item is similar to the previous one. The only difference is the 1 kHz tone is played from the receiver. The result will also be judged automatically.

- **Earphone**

  This item is similar to the previous one. The only difference is the 1 kHz tone is played from the headset earphone. The result will also be judged automatically.

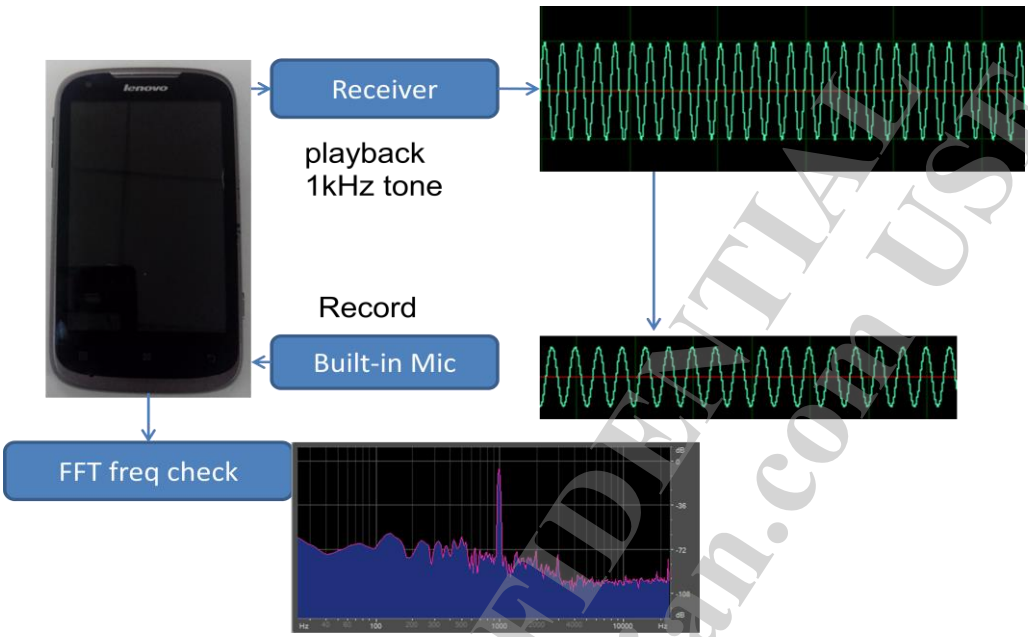


Figure71. Receiver test work flow

- **WavePlayback**

  This is a wave loopback test, and the user can hear a 3 kHz tone sound from speaker. The user need to judge if it is passed or not.

Figure 72. Factory Mode WavePlayback Item Test Menu

- **FM Radio**

  This item is used to check if the FM works. We need to plug-in an earphone first. We can hear the FM sound from earphone shortly. The result will be automatically judged.

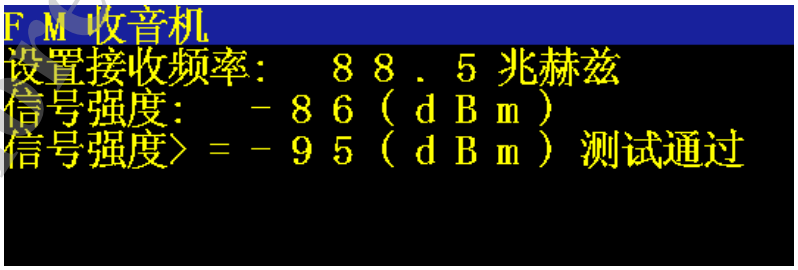


Figure 43. Factory Mode FM Radio Item Test Menu

5.2.4 Test Report

In the test report, if the item is fail, the background color will be red and shows [X]; otherwise, the background color will becomes green and shows [O]. If there is no background color, it means the item has not been tested.

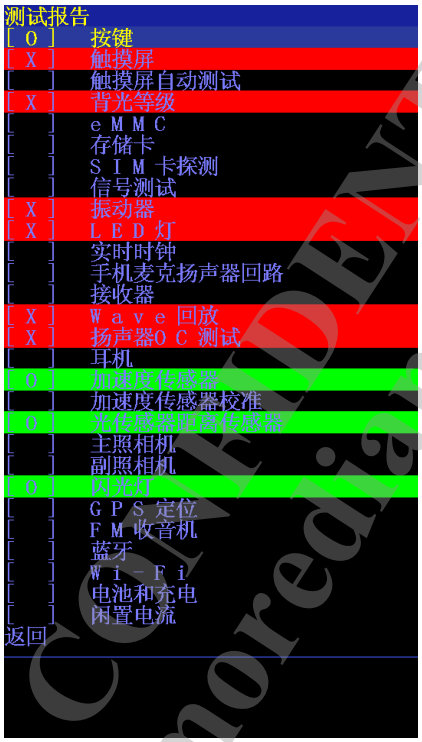


Figure 44. Factory Mode Test Report Menu

6 Audio Frameworks

6.1 Introduction

In the document, we will show the audio architecture in native layer, including AudioFlinger and Mediatek HAL (Hardware Abstraction Layer).

6.1.1 Overview

We show the architecture in the following figure.

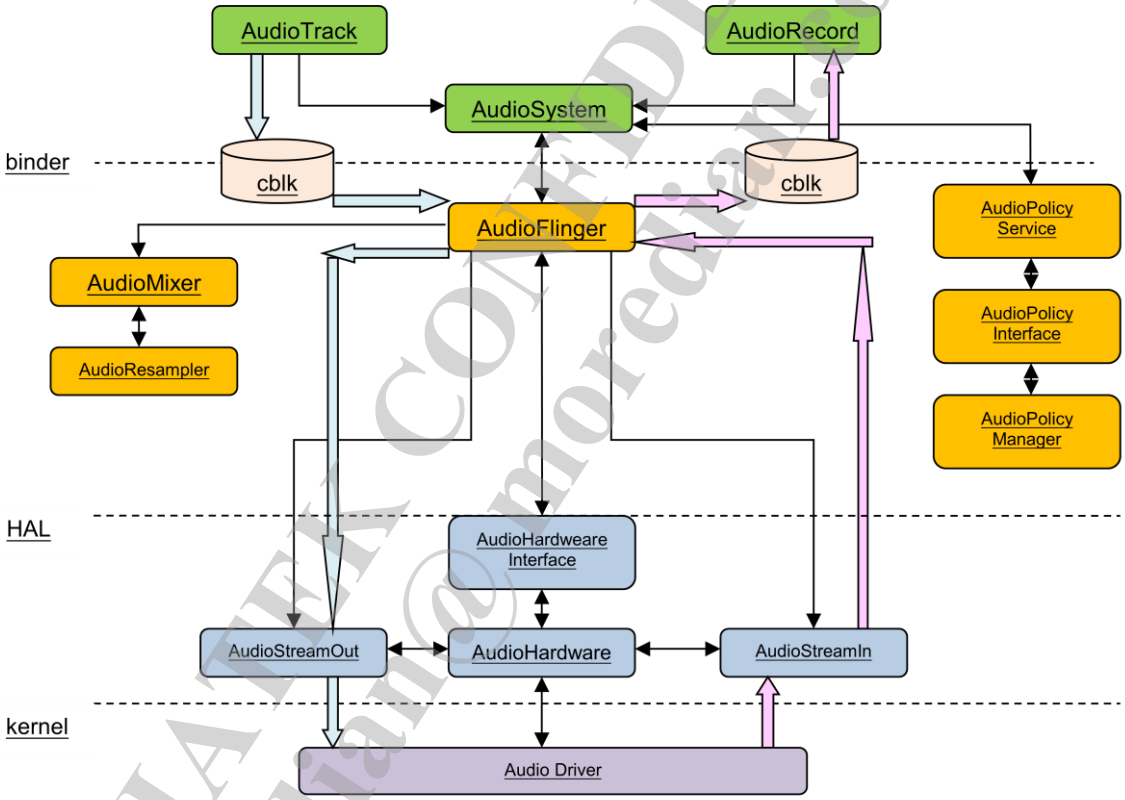


Figure 45. Audio Software Architecture

6.1.2 Functionality

In the section, we introduce the functionality of each block in Figure 1.

Block Name	*Description*
AudioTrack	*Collect and handle the PCM data for playback.*
AudioRecord	*Collect and handle the PCM data for recording.*
AudioSystem	*Provide interface and handle audio operation.*
AudioFlinger	*Handle the PCM data for playback and recording, including signal processing (pre- and post- processing).*

AudioMixer	*Mix the PCM data for playback.*
AudioResampler	*Convert to the specific sampling rate; then AudioMixer can mix data.*
AudioPolicyService	*Provide the interface and service to set/get device and volume information.*
AudioPolicyInterface	*Provide the interface that AudioPolicyManager should be implemented.*
AudioPolicyManager	*Implement the audio policy, such as device routing and volume calculation.*
cblk	*Cblk is the abbreviation of "control block". It is the sheared circular buffer between AudioFlinger and AudioTrack/AudioRecord.*
AudioHardwareInterface	*Provide hardware interface to AudioFlinger for read/write data and config/query.*
AudioStreamIn	*Handle the stream for recording. Read PCM data from driver.*
AudioStreamOut	*Handle the stream for playback. Write PCM data to driver.*
AudioHardware	*In user space, communicate with audio driver in kernel space.*
ALSA Driver	*Call ALSA API to control hardware in kernel space.*

6.1.3 Mediatek HAL

Because there are several differences between each chip and version, we take MT6771 L1 as example to show the common part of MediaTek HAL.

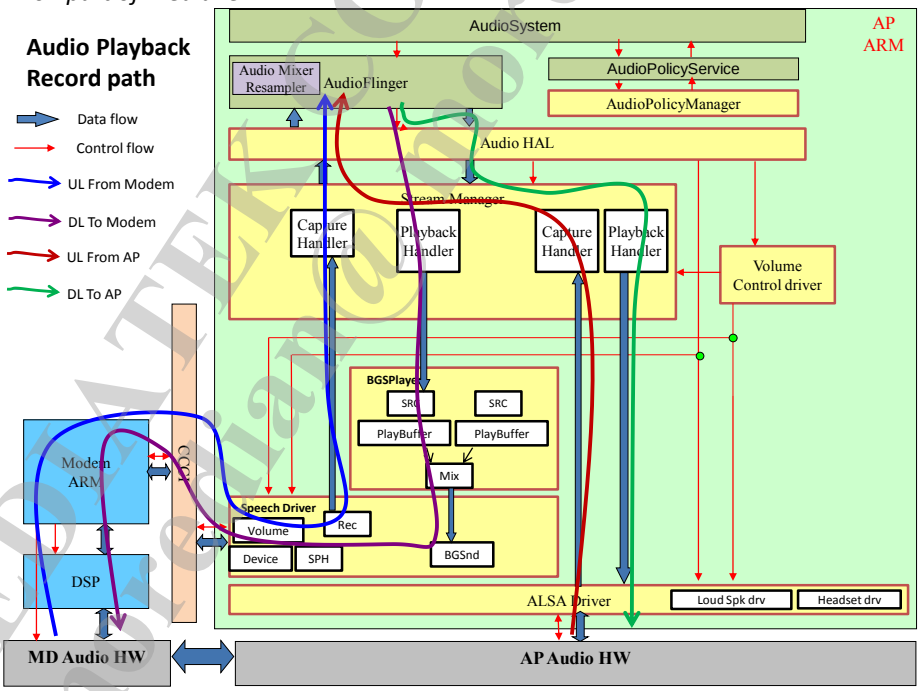


Figure 46. Audio Software Data Flow

The functionality of each major block is following.

Block Name	*Description*

StreamManager	*Manage input and output stream, which is related to AudioStreamIn and AudioStreamOut.*
BGSPlayer	*A player to handle the playback during speech call.*
SpeechDriver	*The speech driver implements all speech related functionality. For example, turn on/off speech, background sound, voice recording, and modem DSP ditital gain.*
CCCI	*CCCI is the abbreviation of "Cross Core Communication Interface". It is an interface to communicate between modem side and AP side.*
Modem ARM	*We put modem side speech driver on it.*
DSP	*It implements speech codec, and communicates with modem.*
Modem Audio HW	*These are hardware related to control Audio Front End.*

6.2 Binder Service

AudioFlinger and AudioPolicyService are two services registered to binder. They are service providers, which inherit from BinderService. While AudioSystem is service users, AudioSystem uses the service of AudioFlinger and AudioPolicyService through binder. The following figure shows the service architecture.

MTK_BT_FM_OVER_BT_VIA_CONTROLLER	yes or no	Default enable

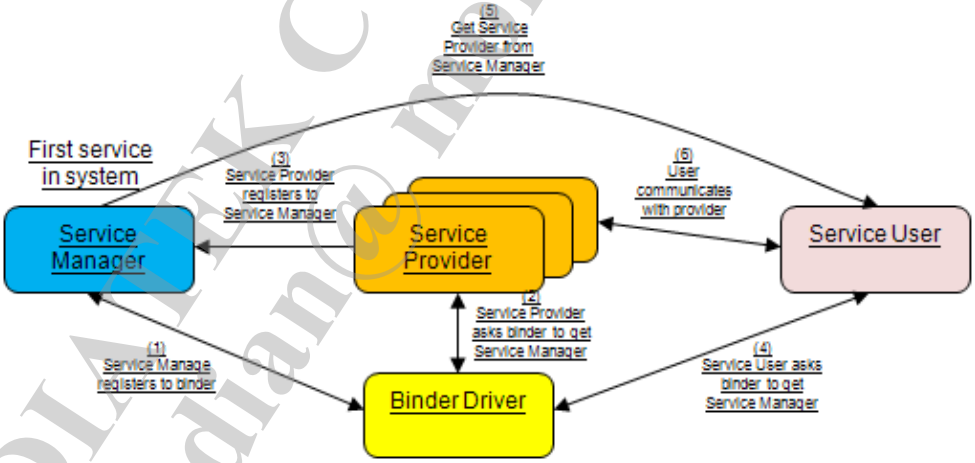


Figure 47. Binder Architecture

6.2.1 Instantiate

We introduce how AudioFlinger registers to binder service, and how AudioSystem gets the service of AudioFlinger in the following sentences. For AudioPolicy, the operation is similar.
The method to register service for AudioFlinger:

(1) *In Main_mediaserver.cpp (Android6.0)/Main_audioserver.cpp (Android7.0), it instantiates AudioFlinger.*
AudioFlinger::instantiate();

(2) *The instantiate function is BinderService.h. It adds service to ServiceManager by name.*
static void instantiate() { publish(); }

```
    static status_t publish(bool allowIsolated = false) {
        sp<IServiceManager> sm(defaultServiceManager());
        return sm->addService(String16(SERVICE::getServiceName()), new SERVICE(), allowIsolated);
    }
```

(3) The getServiceName function is in AudioFlinger.h

```
    static const char* getServiceName() { return "media.audio_flinger"; }
```

The method to get AudioFlinger Service by AudioSystem:
In AudioSystem.cpp, it obtains the ServiceManager at first, and gets the service by using the name of AudioFlinger.

```
    sp<IServiceManager> sm = defaultServiceManager();
    sp<IBinder> binder;
    do {
        binder = sm->getService(String16("media.audio_flinger"));
        if (binder != 0)
            break;
        ALOGW("AudioFlinger not published, waiting...");
        usleep(500000); // 0.5 s

    } while (true);
```

6.2.2 Command Transact

For the Bn- and Bp- prefix API, it means:
Bn means native. It is implemented in service provider.
Bp means proxy. It is referenced in service user.
We show the notation in the following figure.

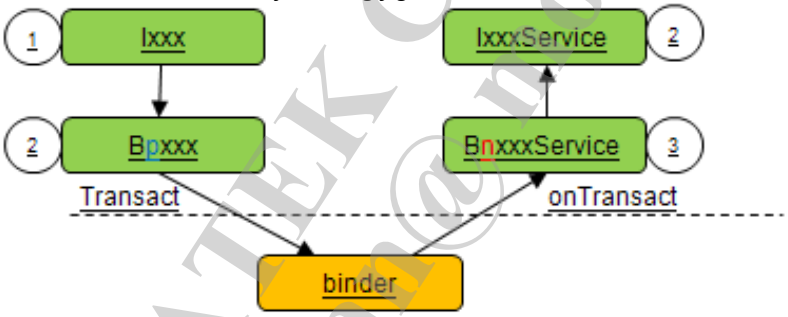


Figure 48. Bn and Bp API

We take the command, newAudioSessionId, between AudioSystem and AudioFlinger as example.

(1) In AudioSystem.cpp, AudioSystem calls af->neAudioSessionId(), where af is IAudioFlinger.

```
int AudioSystem::newAudioSessionId() {
    const sp<IAudioFlinger>& af = AudioSystem::get_audio_flinger();
    if (af == 0) return 0;
```

(2) In IAudioFlinger.cpp, it converts the command into another format for transmission.

```
virtual int newAudioSessionId()
{
    Parcel data, reply;
    data.writeInterfaceToken(IAudioFlinger::getInterfaceDescriptor());
    status_t status = remote()->transact(NEW_AUDIO_SESSION_ID, data, &reply);
    int id = 0;
    if (status == NO_ERROR) {
```

(3) In IAudioFlinger.cpp, the transferred command is converted to original format.

```
status_t BnAudioFlinger::onTransact(
    uint32_t code, const Parcel& data, Parcel* reply, uint32_t flags)
{
    switch (code) {
...
        case NEW_AUDIO_SESSION_ID: {
            CHECK_INTERFACE(IAudioFlinger, data, reply);
```

(4) In audioFlinger.cpp, it implements the newAudioSession

```
int AudioFlinger::newAudioSessionId()
{
    return nextUniqueId();
```

The interface between user (AudioSystem / AudioTrack / AudioRecord) and provider (AudioFlinger / AudioPolicyService) is defined under the folder:
\alps\frameworks\av\include\media
The implementation is under the folder:
\alps\frameworks\av\media\libmedia

Although the interface for binder and the interface of internal header file are two individual files, the contain of interface is equivalent. The interface will be described in each section.

Header File	Implementation File	Service Provider	Service User	Section
IAudioFlinger.h	IAudioFlinger.cpp	AudioFlinger	AudioSystem	4.6
IAudioPolicyService.h	IAudioPolicyService.cpp	AudioPolicyService	AudioSystem	3.5
IAudioTrack.h	IAudioTrack.cpp	AudioFlinger::TrackHandle	AudioTrack	5.2
IAudioRecord.h	IAudioRecord.cpp	AudioFlinger::RecordHandle	AudioRecord	5.3

6.3 Audio Policy

AudioPolicy decides the routing path and volume. It only makes decision, but doesn't control hardware. The notification and hardware control is done by AudioFlinger.

6.3.1 Thread

There are three threads -- one is TonePlaybackThread, one is ApmCommandThread, and the other is OutputCommandThread.

Thread Name	Trigger	Description
TonePlaybackThread	(1) Time (2) mWaitWorkCV (command)	Handle tone related commands.
ApmCommandThread	(1) Time (2) mWaitWorkCV (command)	Handle volume and set_parameter commands.
OutputCommandThread	(1) Time (2) mWaitWorkCV (command)	Handle output stop and open.

TonePlaybackThread handles tone related commands, such as start / stop tone. The thread is needed to generate tone in specific situation. The scenario is as following:

If the alarm is ringing during speech call, the alarm will be replaced by tone to notify user.

ApmCommandThread handles the volume and set_parameter commands. There is a queue/vector to store the command and delay time. When a new command is coming, it sorts the command queue according time. If there is similar command already in queue, the rear operation (in time sequence) will be removed / invalidated. The reason to remove the rear command is that the old command is out-of-date, and is replaced by the new one.

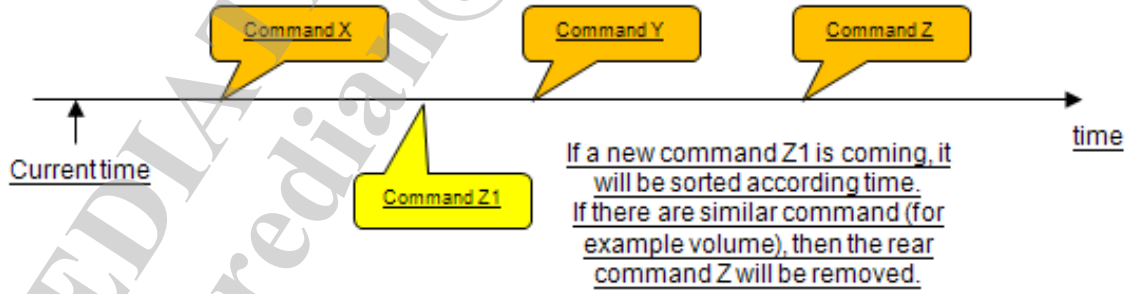


Figure 49. Command Queue Example

6.3.2 Routing

*For each kind of sound, there is a corresponding **stream type**. In order to simplify the routing decision and abstract the manager, the stream type is grouped and handled by a **strategy**. Then AudioPolicy decides **device** by the strategy.*

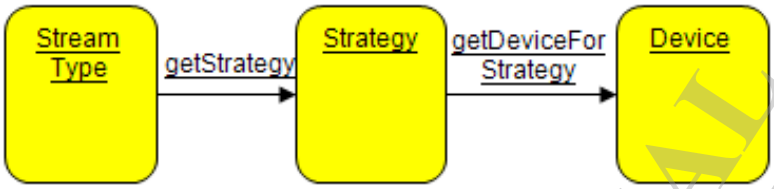


Figure 50. The conversion from stream type to device

The mapping from stream type to strategy is listed in the following table.

Stream Type	Stream Type Value	Strategy
VOICE_CALL	0	STRATEGY_PHONE
BLUETOOTH_SCO	6	
RING	2	STRATEGY_SONIFICATION
ALARM	4	
NOTIFICATION	5	STRATEGY_SONIFICATION_RESPECTFUL
DTMF	8	STRATEGY_DTMF
SYSTEM	1	STRATEGY_MEDIA
VIBSPK	11	
MUSIC	3	
BOOT	10	
Default	-1	
TTS	9	STRATEGY_TRANSMITTED_THROUGH_SPEAKER
ACCESSIBILITY	12	STRATEGY_ACCESSIBILITY
REROUTING	13	STRATEGY_REROUTING
ENFORCED_AUDIBLE	7	STRATEGY_ENFORCED_AUDIBLE

The decision rule in getDeviceForStrategy is mainly as following:

(1) According to strategy and phone status, dispatch to the rule of Phone or Media.

(2) If ForceUse is set, use the specified device.

If ForceUse isn't set, choose a device from the available devices according priority.

We list the device priority of STRATEGY_MEDIA as following.

Priority	Device
1	AUDIO_DEVICE_OUT_REMOTE_SUBMIX
2	DEVICE_OUT_BLUETOOTH_A2DP
3	DEVICE_OUT_BLUETOOTH_A2DP_HEADPHONES
4	DEVICE_OUT_BLUETOOTH_A2DP_SPEAKER

5	DEVICE_OUT_WIRED_HEADPHONE
6	DEVICE_OUT_WIRED_HEADSET
7	AUDIO_DEVICE_OUT_USB_ACCESSORY
8	AUDIO_DEVICE_OUT_USB_DEVICE
9	DEVICE_OUT_DGTL_DOCK_HEADSET
10	DEVICE_OUT_AUX_DIGITAL
11	DEVICE_OUT_ANLG_DOCK_HEADSET
12	DEVICE_OUT_SPEAKER

6.3.3 Volume

The volume value is related to output device. The decision rule is as following:

- Get the output device according to phone state

  If ForceUse is set, use the specified device.

  If ForceUse isn't set, choose the available device according to priority.
- According to the device and stream type, use the volume curve to compute the gain.

Phone State	ForceUse is set	Force isn't set
MODE_NORMAL	Find forced device for FOR_MEDIA	Audio_Find_Normal_Output_Device
MODE_RINGTONE	Find forced device for FOR_MEDIA	Audio_Find_Ringtone_Output_Device
MODE_IN_CALL	Find forced device for FOR_COMMUNICATION	Audio_Find_Incall_Output_Device
MODE_IN_COMMUNICATION	Find forced device for FOR_COMMUNICATION	Audio_Find_Communcation_Output_Device

6.3.4 Mutex

In the section, we list the mutex and cv (condition vector) in AudioPolicyService.

Name	Level	Protect
mLock	AudioPolicyService	Protect the operation of AudioPolicyService. Mainly for mpAudioPolicy, mInputs, …
mLock	AudioPolicyService:: AudioCommandThread	Protect the private variable of AudioCommandThread. Mainly for mAudioCommands, …

mFunLock	AudioPolicyService:: AudioCommandThread	Added by MTK to resolve ALPS00255939. Protect the following three functions: (mLock is unlocked by conditional vector). volumeCommand, parametersCommand, and voiceVolumeCommand.
mWaitWorkCV	AudioPolicyService:: AudioCommandThread	It is used to synchronize the thread command and threadLoop. Company with AudioCommand.
AudioCommand	AudioPolicyService:: AudioCommandThread	There is a cv (command->mCond) in the class. It is used to synchronize the thread command and threadLoop.

6.3.5 Interface

About the interface or function provided by AudioPolicy, please refer to AudioPolicyService.h for detail.

We list the public interface in the following table:

Function Name	Description
getServiceName	Get the service name for binder service.
dump	Dump the command and device status for debug.
setDeviceConnectionState	Set the device connection / disconnection state
getDeviceConnectionState	Query the device state
setPhoneState	Set phone state (Normal / In Call / Ringtone / …)
setForceUse	Set force to use specified device (forced_config) under certain purpose (force_use)
getForceUse	Query the device (forced_config) under specific purpose (force_use)
getOutput	Get the audio i/o handle (HW module: primary / A2DP / …) for output
startOutput	Start audio i/o for output
stopOutput	Stop audio i/o for output
releaseOutput	Release the audio i/o handle for output
getInput	Get the audio i/o handle (primary / …) for input
startInput	Start audio i/o for input
stopInput	Stop audio i/o for input
releaseInput	Release the audio i/o handle for input
initStreamVolume	Set the min/max volume index for specific stream type
setStreamVolumeIndex	Set the volume index for specific stream type
getStreamVolumeIndex	Get the volume index for specific stream type

getStrategyForStream	Convert from stream type to strategy
getDevicesForStream	Convert from stream type to device. To decide the output device.
getOutputForEffect	To get the device that the effect is attached to.
registerEffect	Register the audio effect. In the function, it checks the load and memory size.
unregisterEffect	unregister the audio effect
setEffectEnabled	Enable / Disable the effect by ID.
isStreamActive	Query whether the stream type is active.
SetPolicyManagerParameters	MTK proprietary message. To set volume or FM output device.
queryDefaultPreProcessing	To query the audio effect on input.
onTransact	It is for service provider to receive the command/message from user.
binderDied	This API is used to catch the event when service user is dead
setParameters	The set/get parameter, set volume, start/stop tone.
setStreamVolume	Set stream volume (not index)
startTone	Start tone
stopTone	Stop tone
setVoiceVolume	Set voice volume (in speech call)

6.4 Audio Flinger

In this chapter, we introduce the class, thread, and interface of AudioFlinger.

6.4.1 Class

AudioFlinger itself is a class. It declares many classes to handle specific operation. We list the functionality of each sub-class in the following table.

Class Name	Inherit From	Description
SyncEvent	RefBase	To sync event. For example, to play a new track after the previous one is finished. Its purpose is to control flow and to avoid the mixture of sound.
Client	RefBase	To record the user.
NotificationClient	IBinder:: DeathRecipient	When client is died, client uses it to notify AudioFlinger.
ThreadBase	Thread	The thread base of mixer / direct / duplicating / record thread.
TrackBase	ExtendedAudio BufferProvider RefBase	The track base of playback / record track. It defines basic operations (start/stop) and status.

ConfigEvent	x	To keep the configured event. For example input/output opened, or configuration changed.
PMDeathRecipient	IBinder::DeathRecipient	To handle the power management when client died.
SuspendedSessionDesc	RefBase	To keep the description of suspended session.
PlaybackThread	ThreadBase	The thread is for playback. It includes thread information, master volume, and track management.
Track	TrackBase VolumeProvider	It handles the buffer information and track status for playback.
TimedTrack	Track	The track is for playback. It includes the time information, and can help to achieve synchronization.
OutputTrack	Track	To handle the output to HAL for playback.
MixerThread	PlaybackThread	To mix the active tracks hook on the thread.
DirectOutputThread	PlaybackThread	The thread to handle the direct output.
DuplicatingThread	MixerThread	To duplicate the output to another device. For example, we can output the audio on loudspeaker and A2DP together by using DuplicatingThread.
TrackHandle	android:: BnAudioTrack	AudioFlinger provides TrackHandle to AudioTrack. Then, AudioTrack can operate track via the TrackHandle.
RecordThread	ThreadBase AudioBufferProvider	Thread for recording. To receive the input data from HAL.
RecordTrack	TrackBase	It handles the buffer information and track status for recording.
RecordHandle	android:: BnAudioRecord	AudioFlinger provides RecordHandle to AudioRecord. Then, AudioRecord can operate track via the RecordHandle.
EffectModule	RefBase	A wrapper to control the effect engine. It keeps a list of EffectHandle objects corresponding to all client applications, handles state machine, and provides operations of effect.
EffectHandle	android:: BnEffect	To implement the IEffect interface. It provides resources to receive parameter updates, keeps track of effect control.
EffectChain	RefBase	To represent a group of effects associated to one audio session.

SuspendedEffectDesc	RefBase	To keep the description of suspended effect
AudioHwDevice	x	To keep the hardware module name and device pointer.

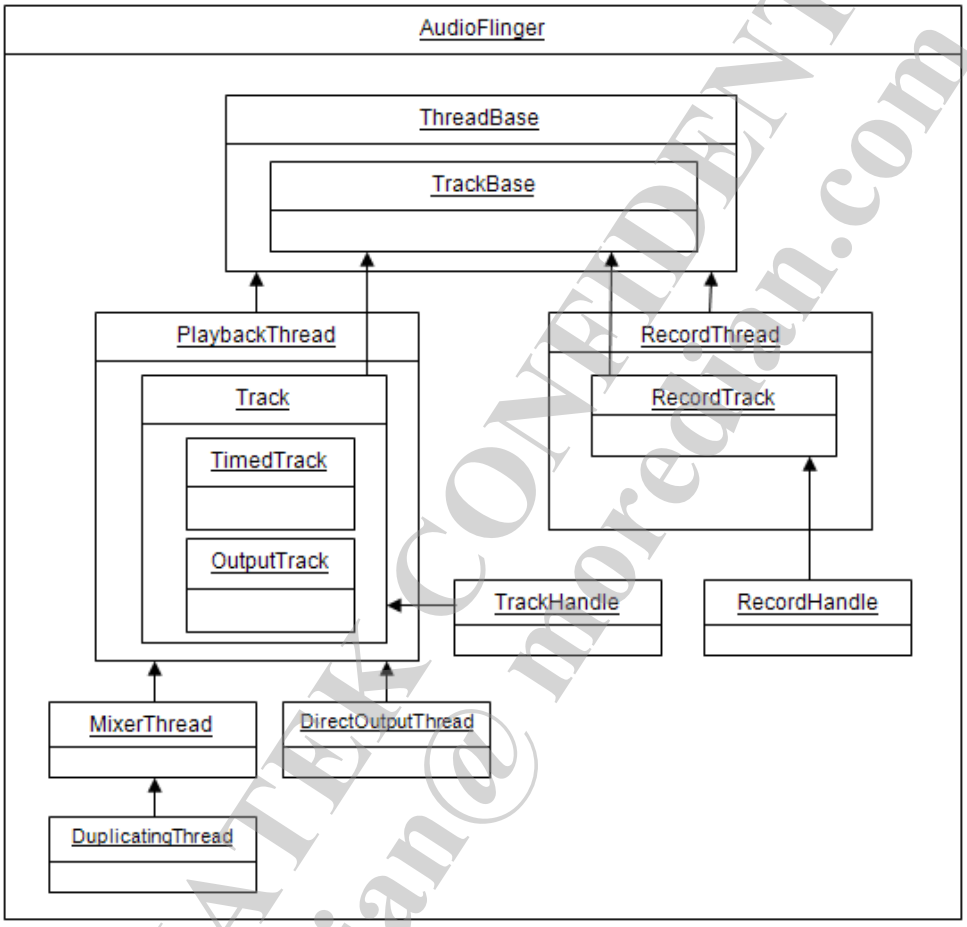


Figure 51. Structure of AudioFlinger

6.4.2 Thread

In previous Figure, there are five threads. The functionality of each thread is described as below.

Thread Name	Trigger	Description
PlaybackThread		A thread for playback. It provides the engine for MixerThread and DirectOutputThread, but the PlaybackThread isn't created.
MixerThread	(1) HAL (write) (2) Time (sleep)	A thread to mix the tracks for playback.

	(3) mWaitWorkCV (standby)	
DirectOutputThread	(1) HAL (write) (2) Time (sleep) (3) mWaitWorkCV (standby)	A thread to direct output without modification for playback. It is for hardware decoder or verification.
DuplicatingThread	As MixerThread	A thread to duplicate the output on another device. When client wants to play audio on two devices, for example loudspeaker and A2DP, client can use it to achieve intention.
RecordThread	(1) Time (5ms sleep) (2) HAL (read) (3) mWaitWorkCV (standby)	A thread for recording.

In the below figure, we show the flow of threadLoop of PlaybackThread.

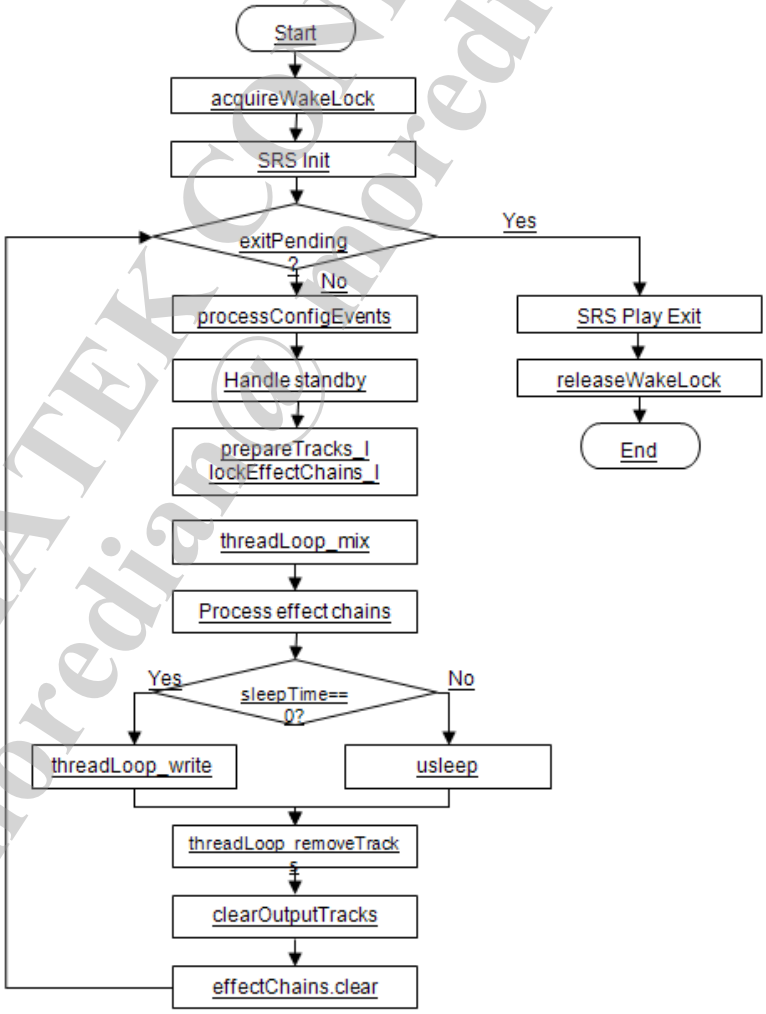


Figure 52. Thread loop of Playback Thread

The flow of threadLoop of RecordThread is shown in following figure

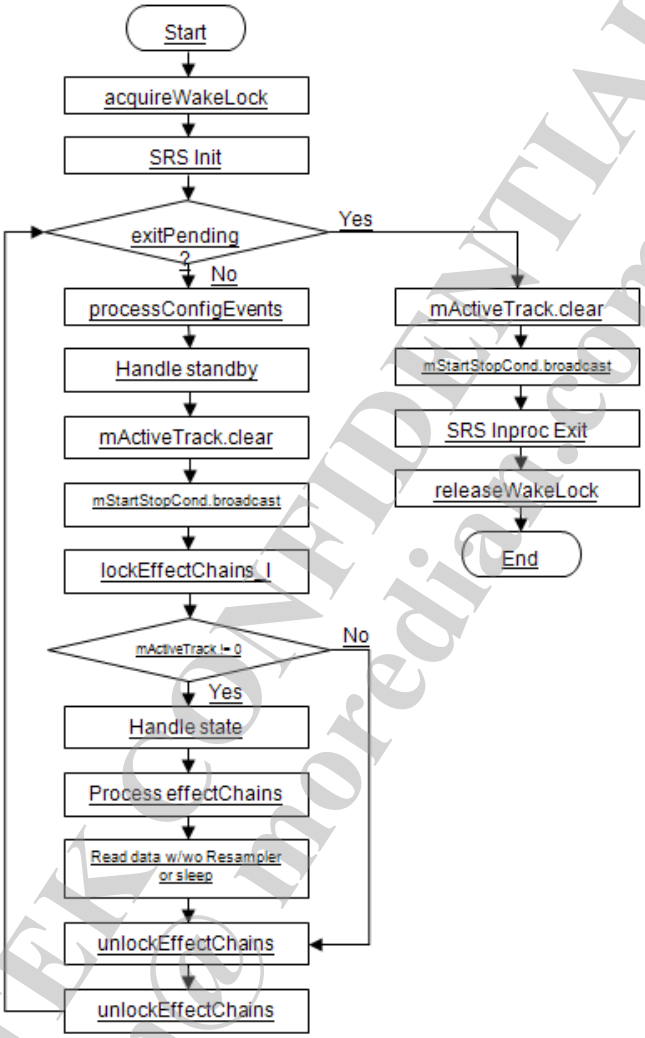


Figure 53. Thread loop of Record Thread

6.4.3 Buffer

In the section, we will show the buffer size on each block. We will focus on playback. At first, we show the data flow of simple playback without audio effect or fast mixer. In Figure 9, there are two pieces of buffer – one is cblk, and the other is mixer.

The buffer size of mixer must be larger than the data amount of hardware latency because mixer has to prepare enough data for hardware. Currently, the latency time is 20ms. However, there is another factor -- bufferSize to specify the minimal memory requirement, which is equal to 8192 by default. So the idea buffer size of mixer is

Max(20 ms *48000 Hz * 2 * 4 / 1000, 8192) = 8192.

The frame count is called mNormalFrameCount.

However, the buffersize can be overwritten. The actual size will get from AudioStreamOut.

For the size of cblk, it is usually the multiple of latency. The latency is calculated from nNormalFrameCount / sampling rate. The multiplier is usually 2. So the size of cblk is at least 7680 bytes for 48k Hz stereo in idea. The actual size is decided by AudioTrack.cpp.

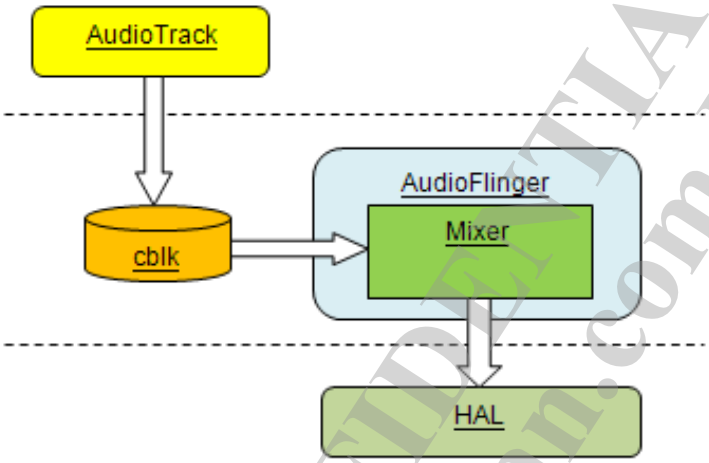


Figure 54. Data Flow of Normal Mixer

For Fast Mixer, the data flow is shown in following Figure.

The buffer size of Fast Mixer is equal to the data amount of HAL, whose latency should be less than 24ms.

The buffer size of cblk for fast track is double of Fast Mixer.

The buffer size of monopipe is four times of Mixer buffer size.

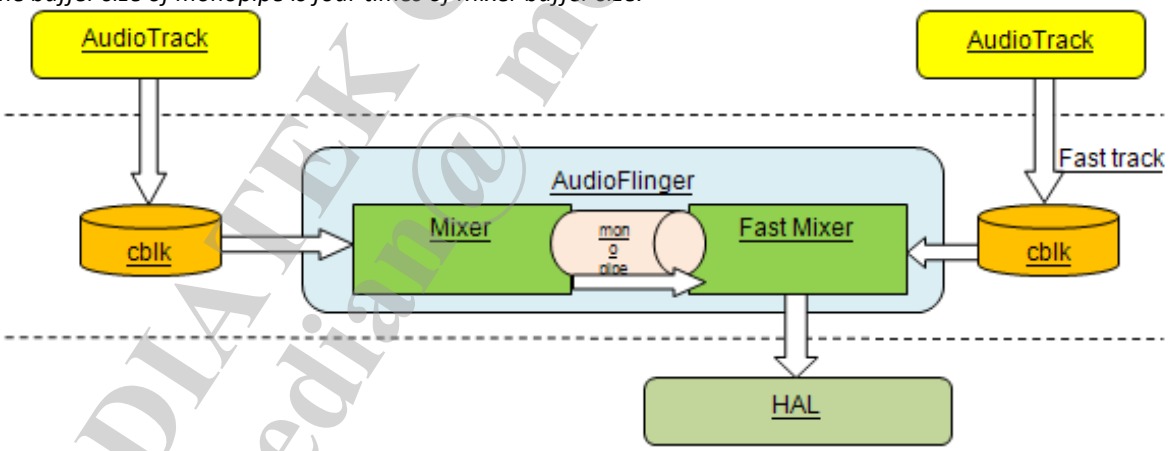


Figure 55. Data Flow of Fast Mixer

For the data flow of AudioEffect, it uses substitution to modify buffer pointer.

At first, we look the simple case without AudioEffect as the following figure. The track data is converted by SRC in Mixer. The converted data is put on Temp Buffer, and then it is accumulated to MixBuffer. It uses MainBuffer to indicate the destination.

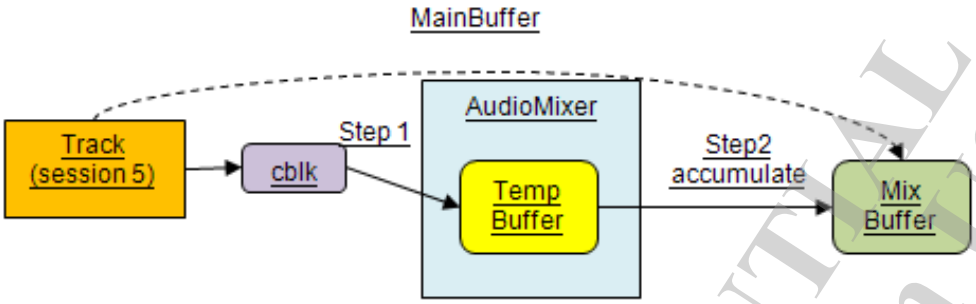


Figure 56. Data Flow without AudioEffect

When there is AudioEffect, the MainBuffer will be redirected to another buffer. The input and output of AudioEffect will reuse the same buffer; the buffer will be directed to MixBuffer at the final stage of EffectChain. The flow is shown in following figure.

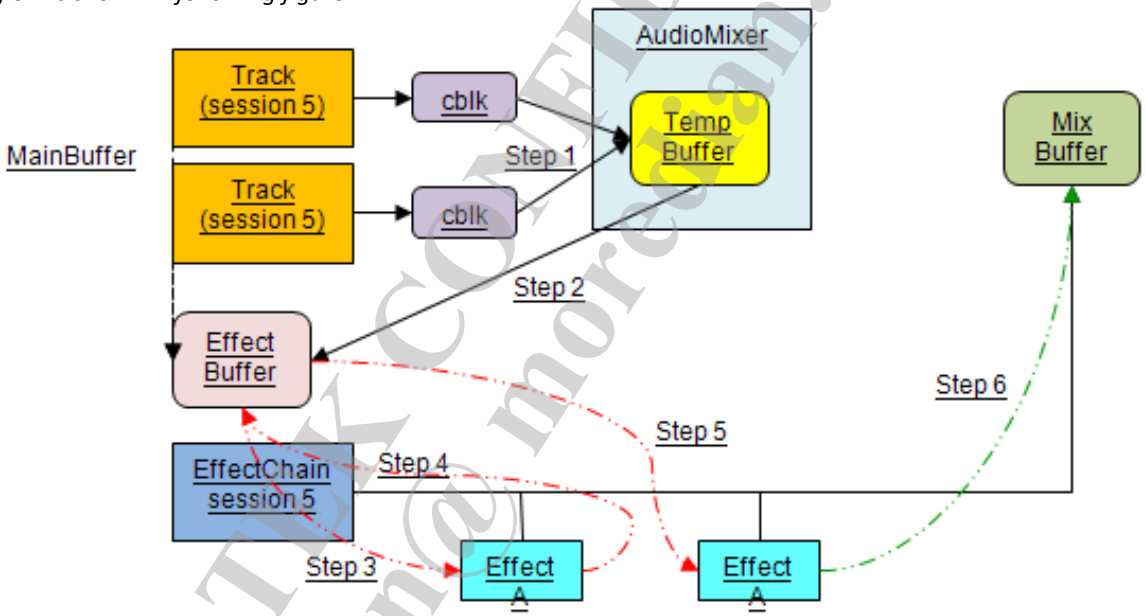


Figure 57. Data Flow with AudioEffect

6.4.4 Mutex

We use mutex to protect critical data and avoid race condition. In the section, we list the mutex in AudioFlinger, and list the priority.

In order to avoid deadlock, we have to set the priority of each mutex, get it in order, and free it in inverse order. We list the general rule in the following table. Then, we will assign the priority to each mutex.

Priority	Mutex category
1	AudioFlinger
2	ThreadBase
3	EffectChain
4	EffectModule

We list all mutex of AudioFlinger in the following table.

In Class	Mutex Name	Priority	Description
AudioFlinger:: SyncEvent	mLock	8	Currently, the mutex isn't used.
AudioFlinger:: Client	mTimedTrackLock		To protect the variable: mTimedTrackCount.
AudioFlinger:: ThreadBase	mLock	3	When thread is operating, it uses the mutex to protect data access. For example, mNewParameters, mTracks, volume, …
AudioFlinger:: PlaybackThread:: TimedTrack	mTimedBufferQueueLock	6	To protect the buffer and buffer queue in TimedTrack.
	mMediaTimeTransformLock	7	To protect information about time transformation. For example, mMediaTimeTransform, mMediaTimeTransformTarget, and mMediaTimeTransformValid.
AudioFlinger:: EffectModule	mLock	5	To protect the Audio Effect. Mainly for mHandles.
AudioFlinger:: EffectChain	mLock	4	To protect the effect chan. Mainly for mInBuffer, mEffects, …
AudioFlinger	mLock	1	When AudioFlinger is operating, it uses the mutex to protect data access. For example, mNotificationClients, mPlaybackThreads, volume …
	mHardwareLock	2	To protect hardware device. When we operate hardware,
AudioFlinger:: Track	mPauseCondLock		To let the AudioFlinger do the volume ramp after the track is paused and before it is flushed.

6.4.5 Synchronization of Parameters

In the section, we show the synchronization of setParameters between client and server.

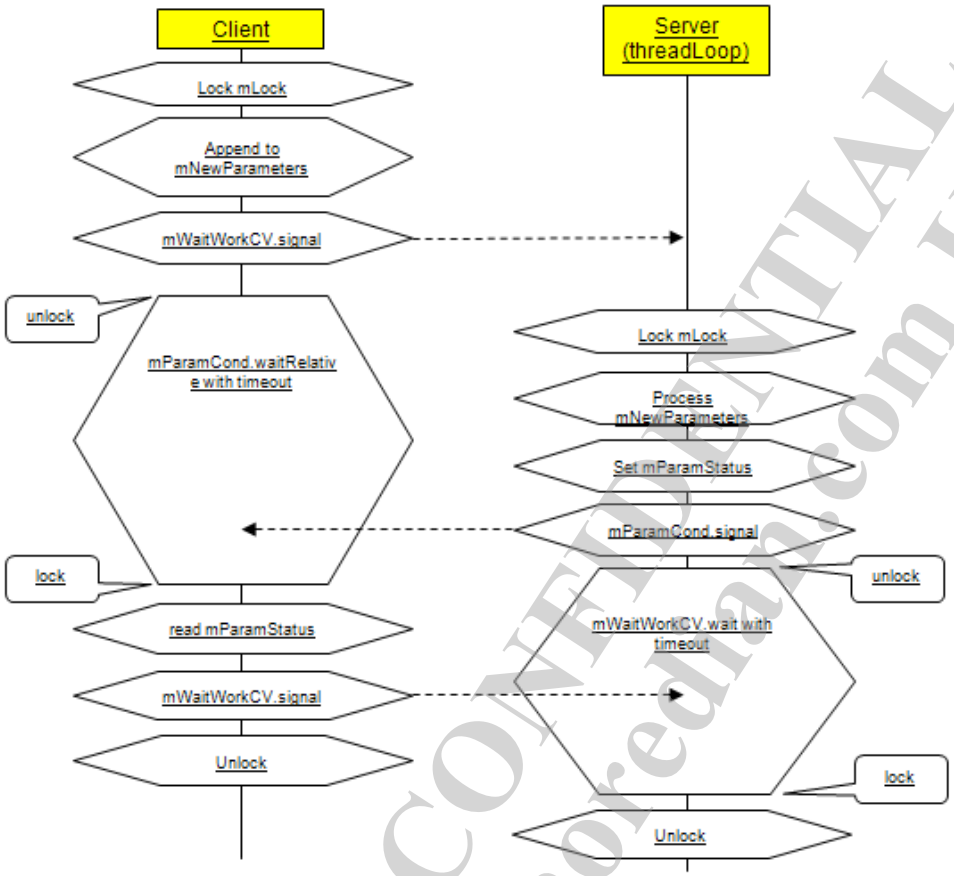


Figure 58. Parameter Synchronization

6.4.6 Interface

About the interface or function provided by AudioFlinger, please refer to AudioFlinger.h for detail.

We list the public interface in the following table:

Function Name	Description
dump	Dump debug information, such as session, hardware status, thread, and device.
createTrack	Create playback track for audio track.
openRecord	Create record track for AudioRecord.
sampleRate	Get the sampling rate of playback.
channelCount	Get the channel number of playback.
format	Get the format of playback.
frameCount	Get the frame count for playback. It is the size of normal mixer.
latency	Get the latency of playback.
setMasterVolume	Set the master volume for playback.
setMasterMute	Mute / un-mute the playback.

masterVolume	Query the master volume.
masterVolumeSW	Query the master software volume.
masterMute	Query whether master volume is mute or not.
setStreamVolume	Set the volume of specified stream type.
setStreamMute	Mute / un-mute the specified stream type.
streamVolume	Query the volume of specified stream type.
streamMute	Query whether the stream type is mute or not.
setMode	Set mode, such as normal / ringtone / in call / in communication.
setMicMute	Mute / un-mute microphone.
getMicMute	Query whether microphone is mute or not.
setParameters	Set parameters to thread.
getParameters	Get parameters from thread.
registerClient	Register the client/user to AudioFlinger.
getInputBufferSize	Query the size of input buffer.
openOutput	Open the output device. (HW module: primary / A2DP / USB audio)
openDuplicateOutput	Open duplicate output in order to play sound on two devices.
closeOutput	Close the output device.
suspendOutput	Suspend output
restoreOutput	Restore output which is suspended.
openInput	Open input device.
closeInput	Close input device.
setStreamOutput	Set the output device of specified stream type.
setVoiceVolume	Set voice volume (in speech call).
getRenderPosition	Get the render position. (frame amount, and DSP consumption)
getInputFramesLost	Get the number of lost input frame.
newAudioSessionId	Get a unique ID.
acquireAudioSessionId	Increase the session reference.
releaseAudioSessionId	Decrease the session reference.
queryNumberEffects	Query the number of effect.
queryEffect	Query the effect by ID.
getEffectDescriptor	Query the effect by uuID.
createEffect	Create audio effect.
moveEffects	Move the effect from a thread to another thread.
loadHwModule	Load hardware module (primary / A2DP / USB Audio).

onTransact	It is for AudioFlinger to receive the command/message from AudioSystem.
createSyncEvent	Create sync event.

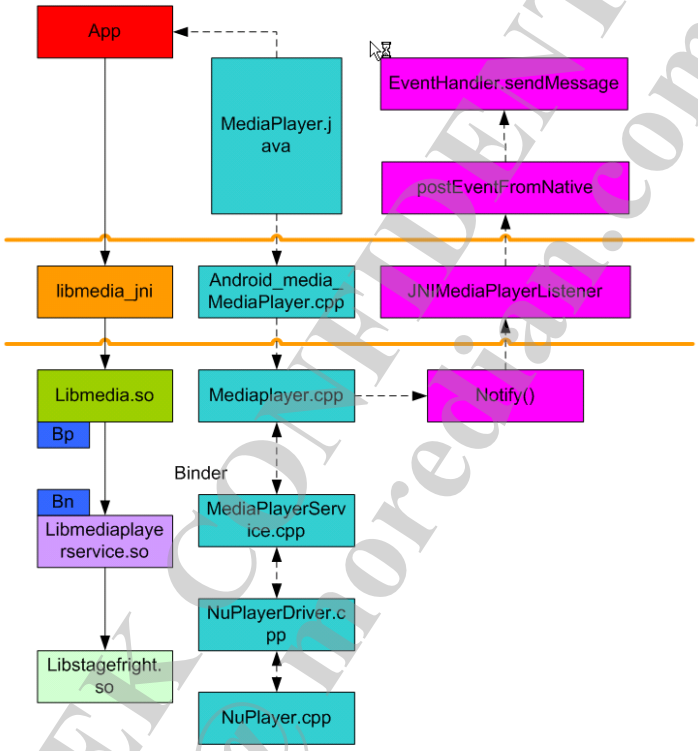
The following shows the MTK proprietary interface:

Function Name	Description
xWayPlay_Start	Start downlink path of PCM2Way.
xWayPlay_Stop	Stop downlink path of PCM2Way.
xWayPlay_Write	Write the output/downlink data to PCM2Way.
xWayPlay_GetFreeBufferCount	Get free buffer count for downlink.
xWayRec_Start	Start uplink path of PCM2Way.
xWayRec_Stop	Stop uplink path of PCM2Way.
xWayRec_Read	Read the input/uplink data from PCM2Way.

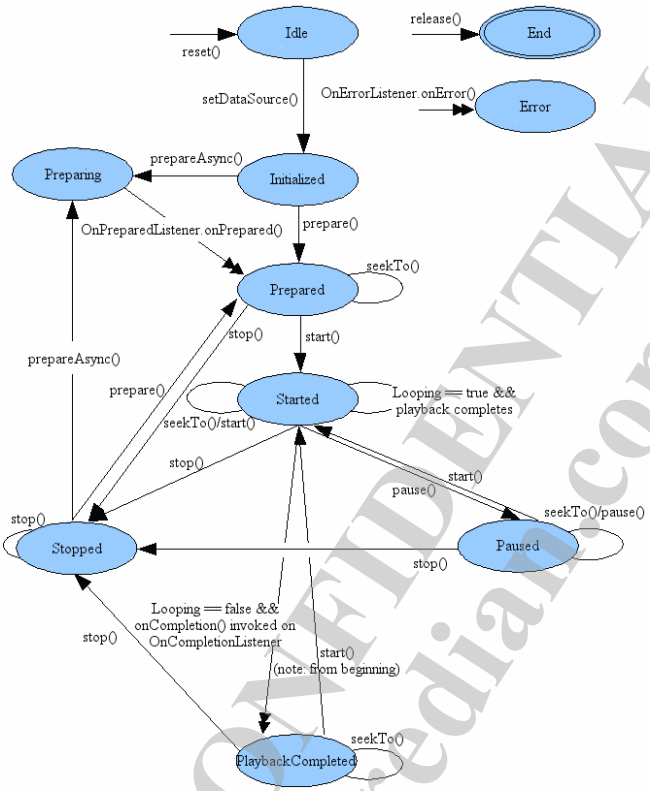
7 NuPlayer

7.1 NuPlayer

7.1.1 NuPlayer Framework

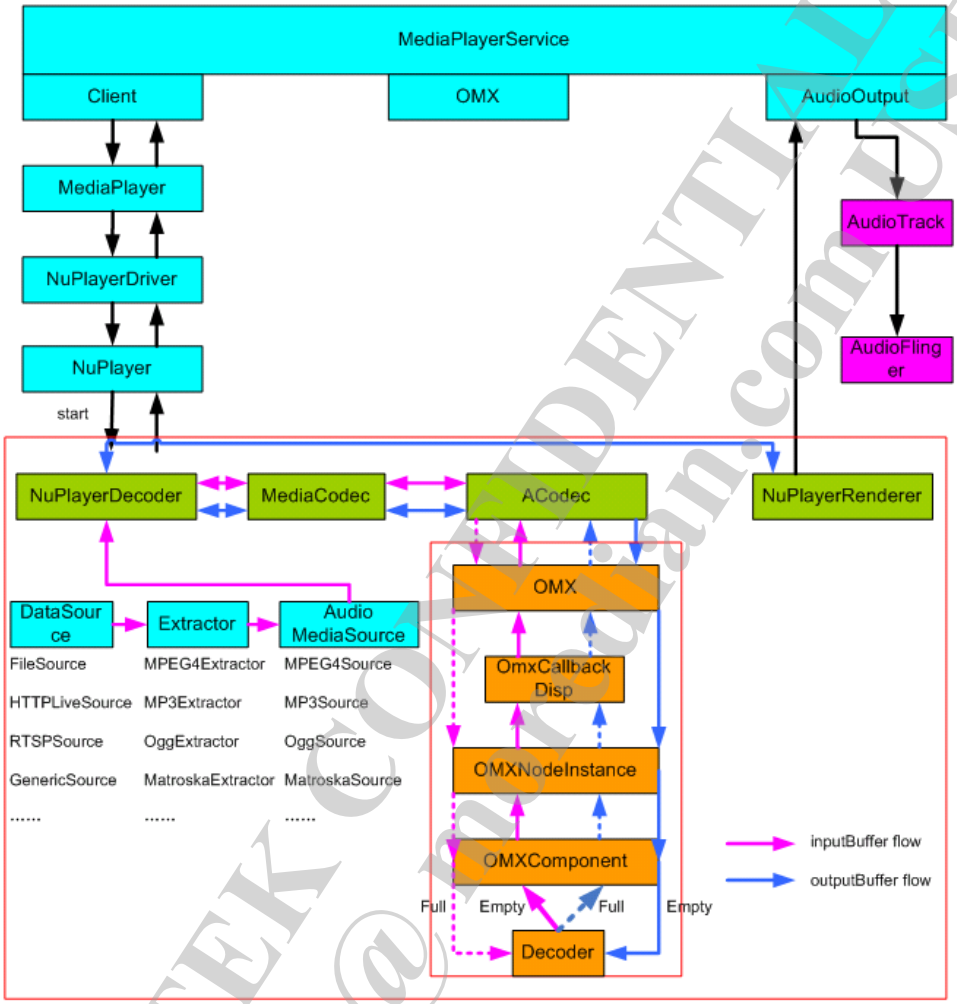


MediaPlayer playback framework



The life cycle and the state of Media object(java)

7.1.2 Object Introduction



Relationships among different objects in NuPlayer

- **InputBuffers & OutputBuffers**
  - **InputBuffer:**
    - It contains Raw-Data (Compressed Data) from audio files, which is waiting for being decode to PCM data.
    - In NuPlayer system, it contains a certain amount of Input Buffers, which are used circularly.
  - **OutputBuffer:**
    - They are PCM data, which comes from audio decoder.
    - The same as InputBuffer, there are always a certain amount of Output Buffers. They are be used circularly.
    - It will be written to AudioTrack for playback.
- **NuPlayer**
  - It is a controller for NuPlayerDecoder and NuPlayerRenderer.

- After NuPlayer is started, NuPlayerDecoder and NuPlayerRenderer will be created. And they will run automatically.
- NuPlayer will monitor the state of NuPlayerDecoder and NuPlayerRenderer and notify it to applications.

➢ **NuPlayerDecoder**

- For InputBuffer, it will fetch Raw-Data from audio files and send it to decoder for decoding. The data flow order is as follow:

- When InputBuffer is used completely, it collect them and for next frame decoding.
- For OutputBuffer, it will send them to NuPlayerRenderer for playback. When it is used completely, it collects them and sends them to decoder for next frame.

➢ **NuPlayerRenderer**

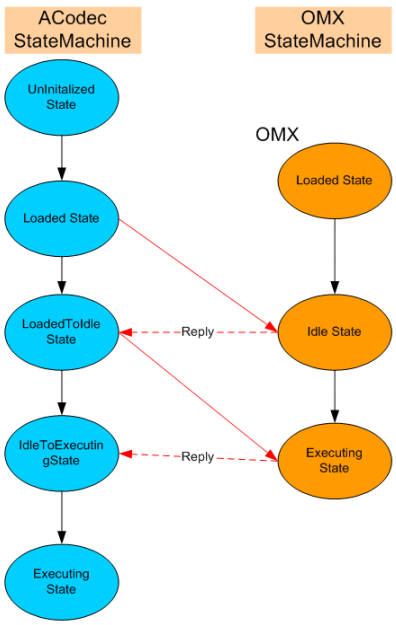
- It is a controller for AudioTrack. It will control AudioTrack directly.
- Write Output Buffers to AudioTrack buffers for playback.
- Compute some information for playback, such as position, AV Sync and so on.

➢ **MediaCodec**

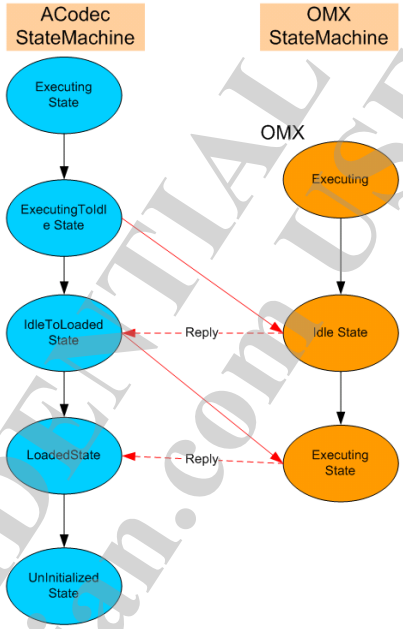
- For NuPlayer playback, it only to transmit command to ACodec from NuPlayerDecoder.
- It also can be used for java code for decoding.

➢ **Acodec**

- It communicate with OMX for 2 things:
  - Send some commands, such as configuration information, set parameters and so on.
  - Get empty Input Buffers and full output Buffers.
  - Send full Input Buffers and empty Output Buffers.
  - State Machine switch for OMX.

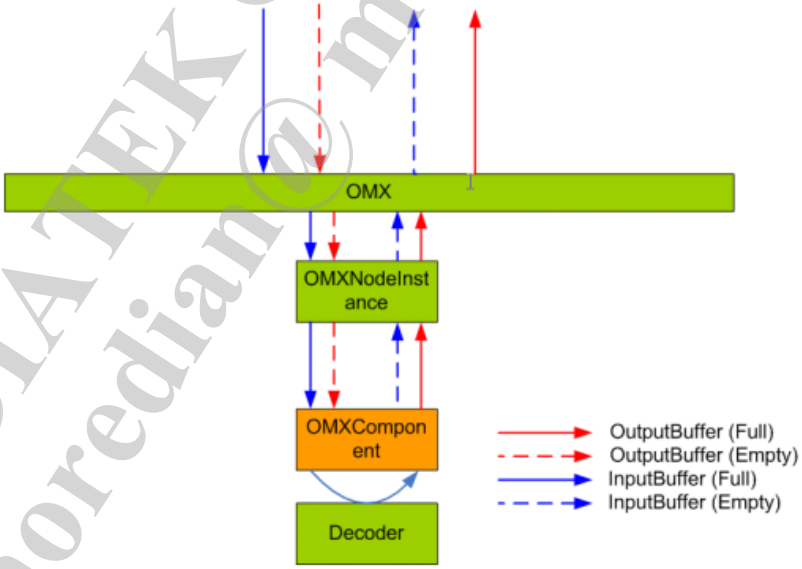


Start Playback State Machine



End Playback State Machine

- OMX Component
  - Send Full InputBuffer to Decoder and Get Full OutputBuffer.
  - Return Empty InputBuffer.
  - State machine.



- **AudioTrack/AudioOutput**

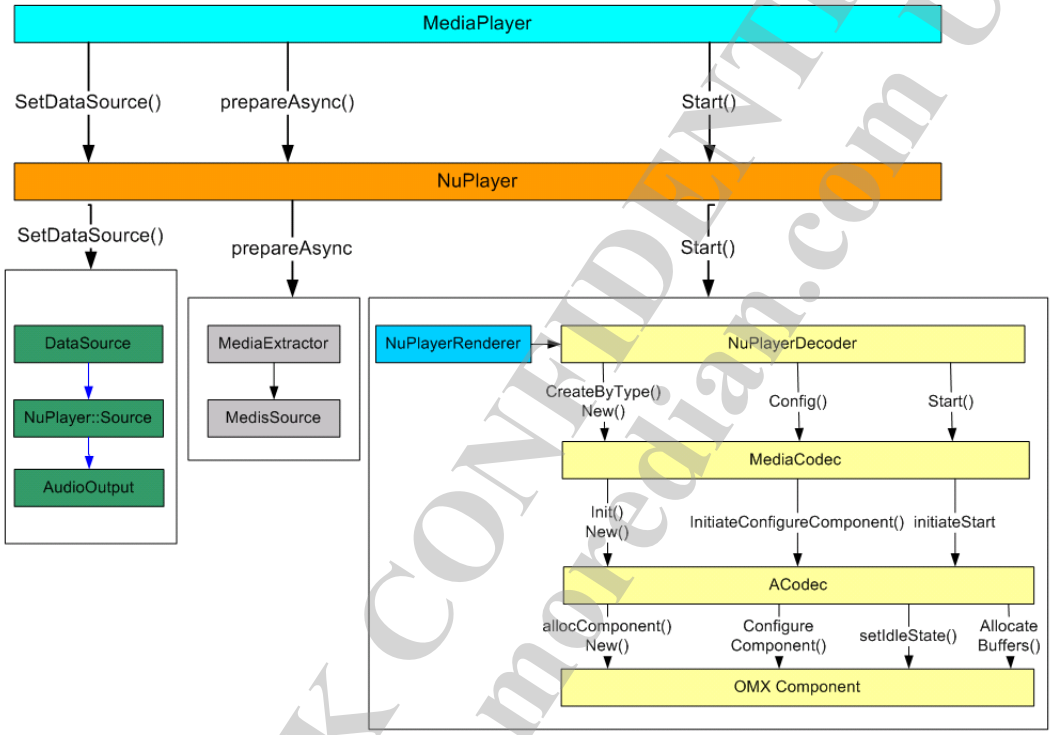
Send PMC data to AudioFlinger

7.1.3 NuPlayer Control Flow

- **Sample Code For MediaPlayer**

i. New a MediaPlayer
ii. SetDataSource()
iii. Prepare()
iv. Start()
v. Now the mp3 file is playing

➢ **Control flow graphics**



7.1.4

8 Audio features

8.1 Phone call record enhancement

Phone call record enhancement is used to support more in-call recording type & better recording quality

8.1.1 Introduction to phone call record enhancement

Original MTK phone call record has following limitation

- Support 8K sample rate format record (modem side only provides 8K sample rate format data)
- Support Downlink + Uplink record. (Support records uplink or downlink data separately)

However, android designed following input source

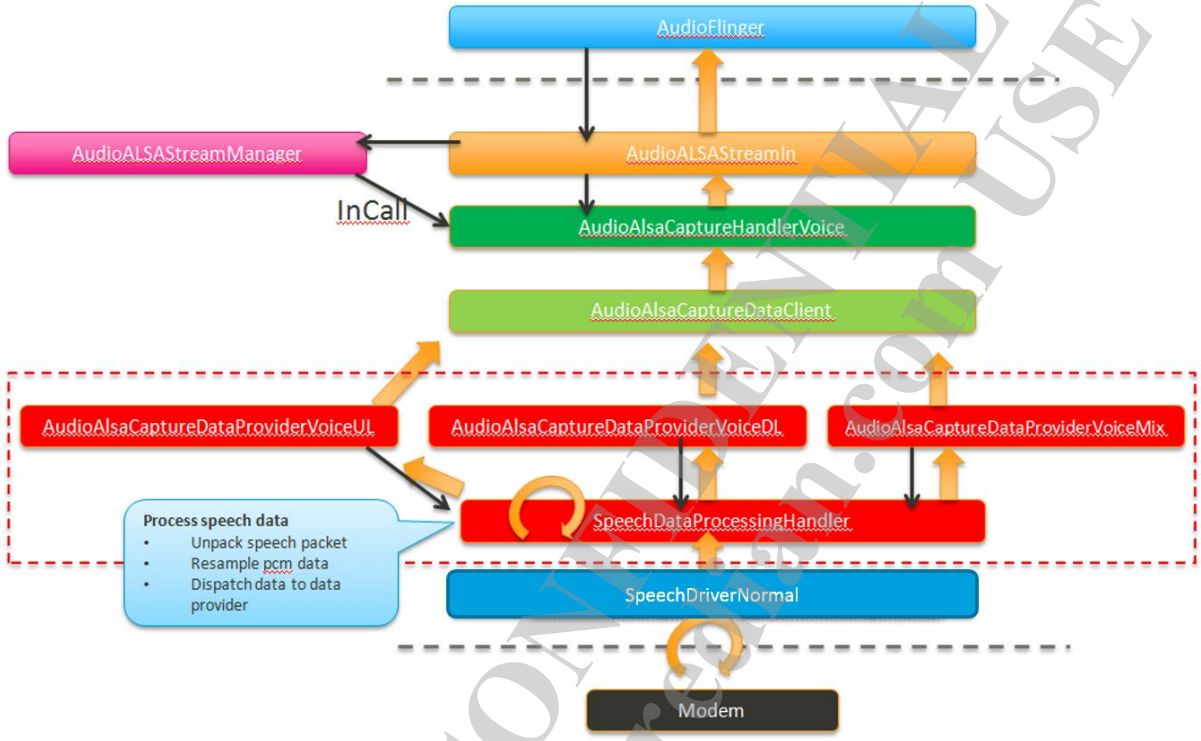
```
typedef enum {
    AUDIO_SOURCE_DEFAULT          = 0,
    AUDIO_SOURCE_MIC              = 1,
    AUDIO_SOURCE_VOICE_UPLINK     = 2, for incall uplink record
    AUDIO_SOURCE_VOICE_DOWNLINK   = 3, for incall downlink record
    AUDIO_SOURCE_VOICE_CALL       = 4, for incall uplink + downlink record
```

The phone call record enhancement is designed to resolve above limitation.

- Can support 16K sample rate (reserve to 48K) format record for better record quality (modem side can provides 8K/16K sample rate format data, depends on telephony network (WB/NB))

Can support Downlink/Uplink/Downlink + Uplink record concurrently

8.1.2 Speech Data Processing Architecture



8.1.3 Customization

For legal & privacy issue, many country don't allow recording the downlink voice data.

Customer could disable the downlink recording capability by following modification

In vendor/mediatek/proprietary/custom/%PROJECT%/hal/audioflinger/audio/audio_custom_exp.h

#define INCALL_DL_RECORD_DISABLED

8.1.4 Debug

Following command could dump the StreamIn PCM to /sdcard/mtklog/audio_dump

- adb shell setprop streamin.pcm.dump 1

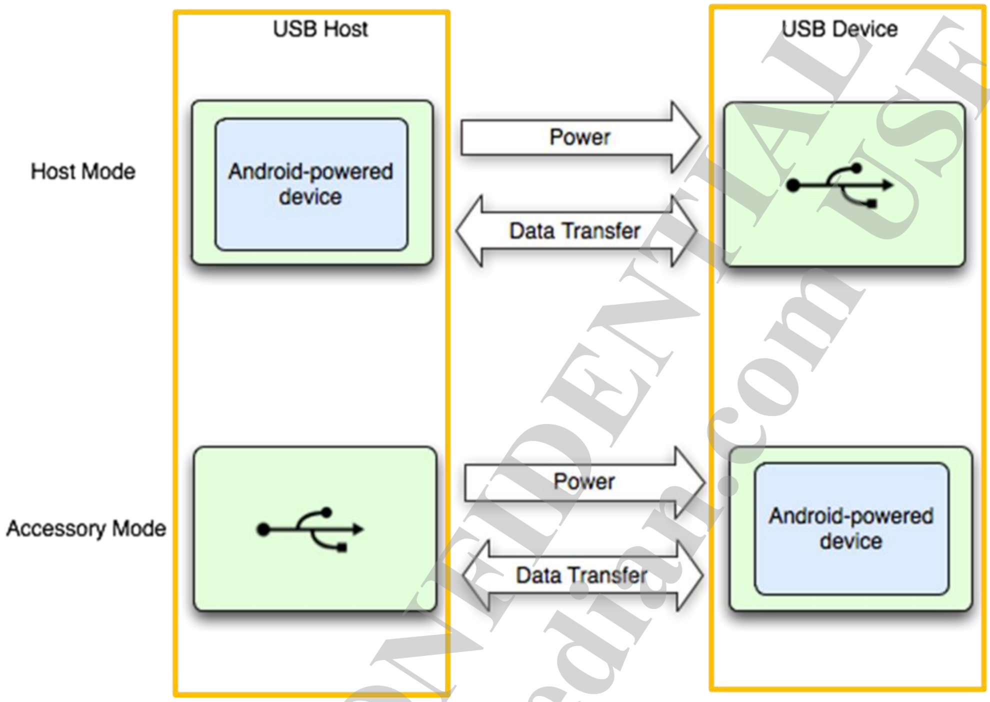
8.2 USB Playback / Record

8.2.1 Introduction to USB Audio

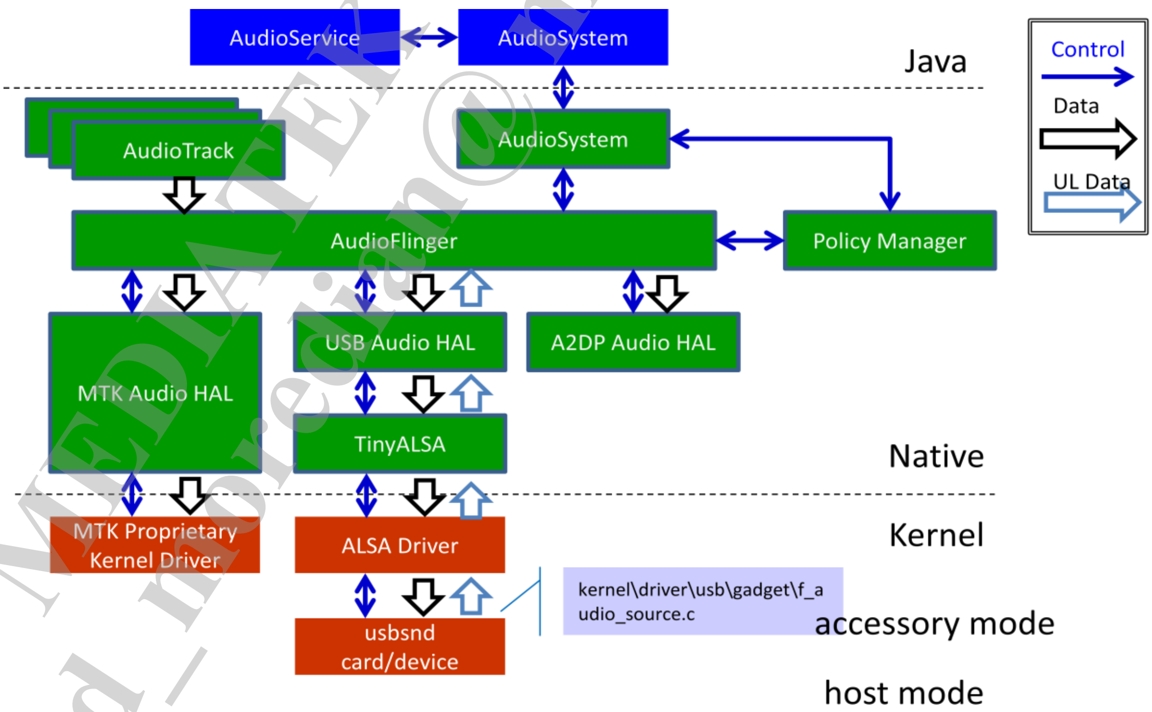
USB Audio host/slave mode introduction

- In USB host mode, the Android-powered device acts as the host
- In USB accessory mode, the external USB hardware act as the USB hosts

Host/Slave mode working like below figure.



USB Audio software architecture is as below figure. The USB audio HAL will use Alsa API to control the usb audio hardware and set PCM format data.



8.2.1.1 How to enable USB audio host mode:

1. Need to modify Audio_Policy.conf to add USB audio host mode supported (Previously on Android/KK, it only supports USB Audio Slave mode).

```
usb {
  outputs {
    usb_accessory {
      sampling_rates 44100
      channel_masks AUDIO_CHANNEL_OUT_STEREO
      formats AUDIO_FORMAT_PCM_16_BIT
      devices AUDIO_DEVICE_OUT_USB_ACCESSORY
    }
    usb_device {        //for USB host mode playback
      sampling_rates dynamic
      channel_masks dynamic
      formats dynamic
      devices AUDIO_DEVICE_OUT_USB_DEVICE
    }
  }
  inputs {              //add it for usb host mode record
    usb_device {
      sampling_rates 44100
      channel_masks AUDIO_CHANNEL_IN_STEREO
      formats AUDIO_FORMAT_PCM_16_BIT
      devices AUDIO_DEVICE_IN_USB_DEVICE
    }
  }
}
```

2.Modify kernel config file to support USB audio mode.

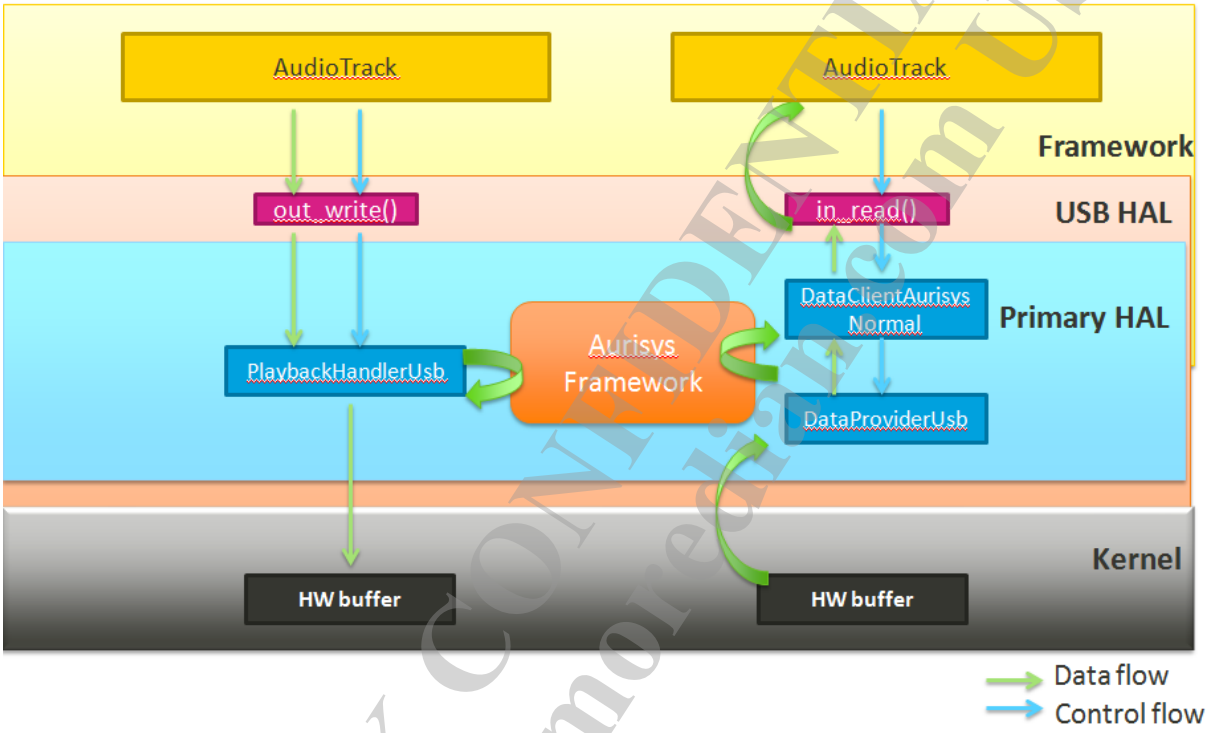
Add CONFIG_SND_USB=y and CONFIG_SND_USB_AUDIO=y into kernel config file

8.2.1.2 Supported format:

For USB audio input, currently setting is to support 44.1K Hz\stereo\16 bit PCM format. And for USB audio output, currently the sample rate/format/channels are set as "**dynamic**" which will check the usb device's capabilities to decide. These are described in audio_policy.conf. It will have SRC process if the usb devices do not support the claim format.

8.2.2 Architecture

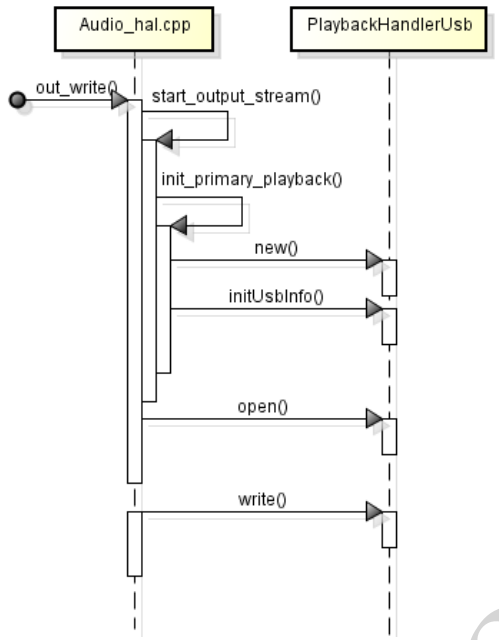
What follows is the USB audio architecture in HAL. For Google default USB Playback and Record design is with no effect processing. In this section, we will introduce MTK proprietary software architecture for USB effect in Playback and Record.



The control and data flow about Android default USB processing is in USB HAL and there is no effect processing . MTK proprietary software architecture for USB effect is designed in primary HAL , using Aurisys architecture. For USB playback we apply the same effect process with device headset, but the default value for usb device is off , and the effect for USB playback is not shown in tunning tool. For USB Record we apply the same effect process with device headset, but default we bypass the effect for usb device , and the effect for USB Record is shown in tunning tool, customer can have their parameter for usb record.

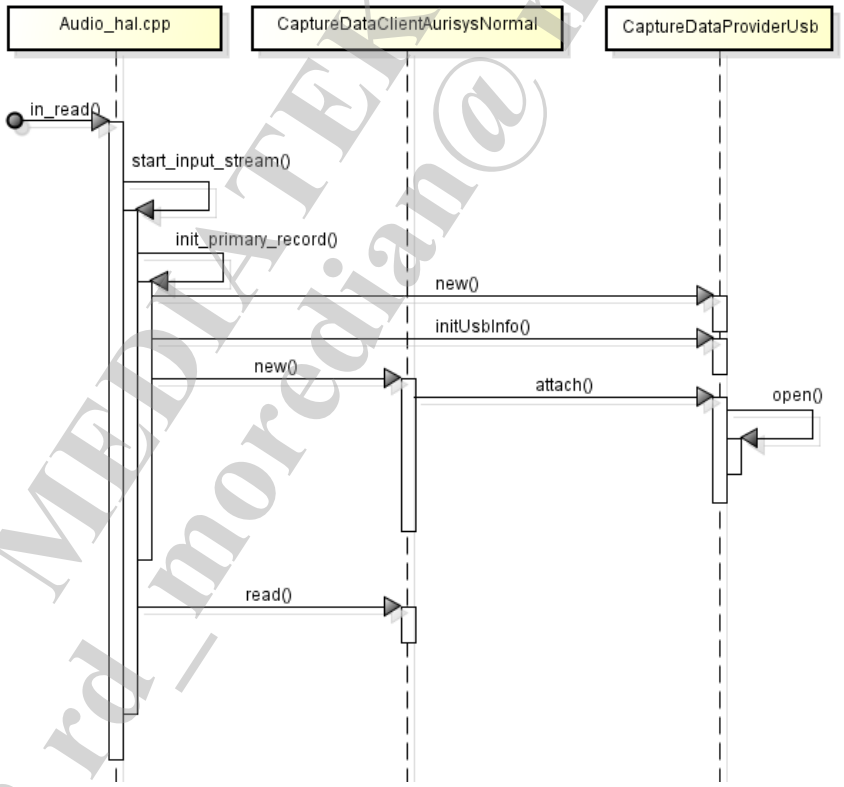
8.2.3 USB Playback Control sequence

The following is the control sequence about USB playback.



8.2.4 USB Record Control sequence

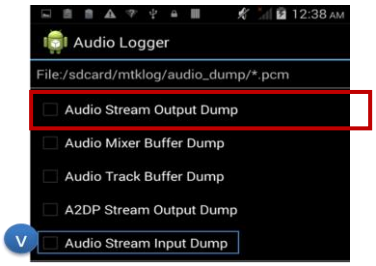
The following is the control sequence about USB Record.



8.2.5 Debug and Tuning Method

8.2.5.1 USB output PCM dump

- (1) Engineer Mode > Hardware Testing > Audio > Audio Logger >



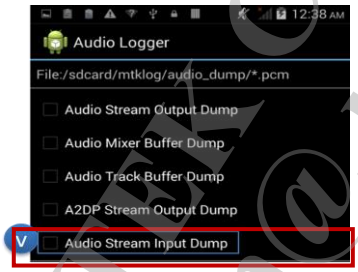
(2) Command: "**adb shell setprop streamout.pcm.dump 1**"

Dump files: usbstreamout.pcm under /sdcard/mtklog/audio_dump

AudioALSACaptureDataProviderUsb.pcm under /sdcard/mtklog/audio_dump

8.2.5.2 USB input PCM dump

- (1) Engineer Mode > Hardware Testing > Audio > Audio Logger >



(2) Command: "**adb shell setprop streamin.pcm.dump 1**"

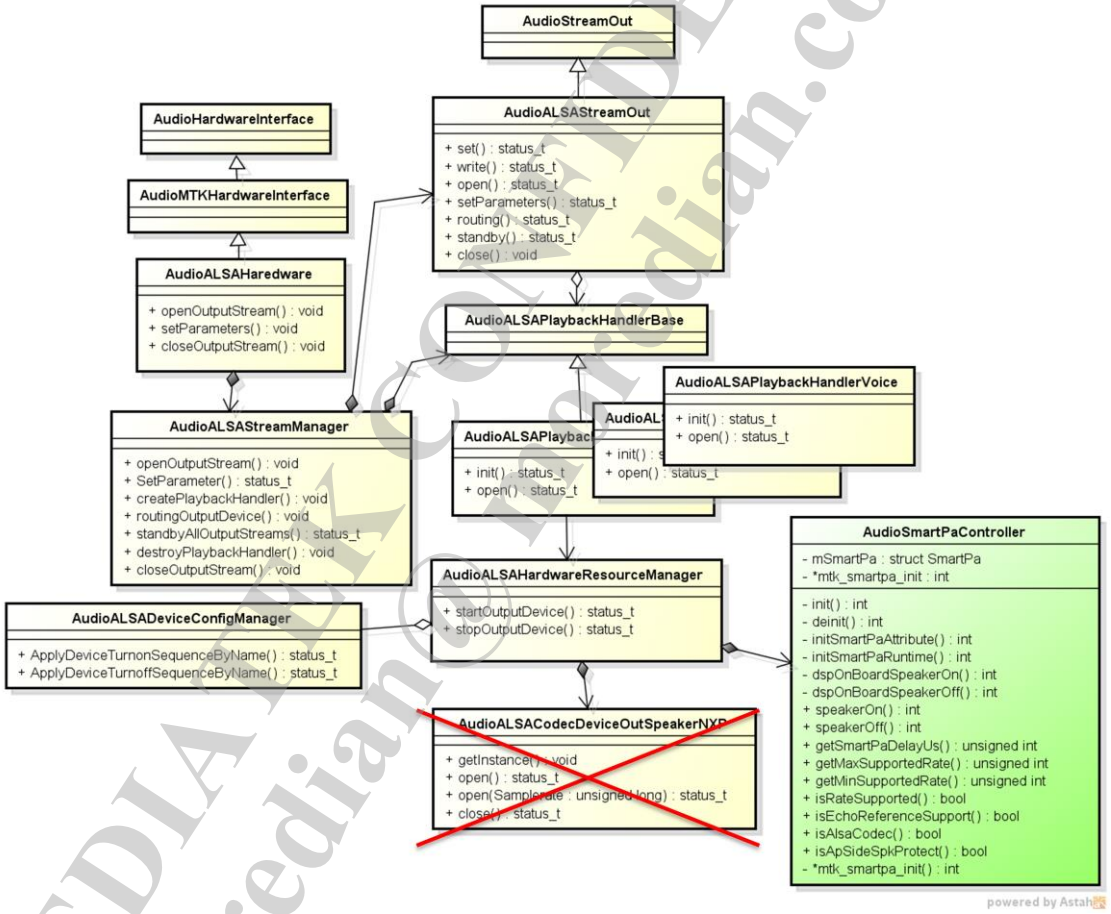
Dump files: usbstreamin.pcm under /sdcard/mtklog/audio_dump

AudioALSAPlaybackHandlerUsb.pcm under /sdcard/mtklog/audio_dump

9 Mediatek SmartPA Framework

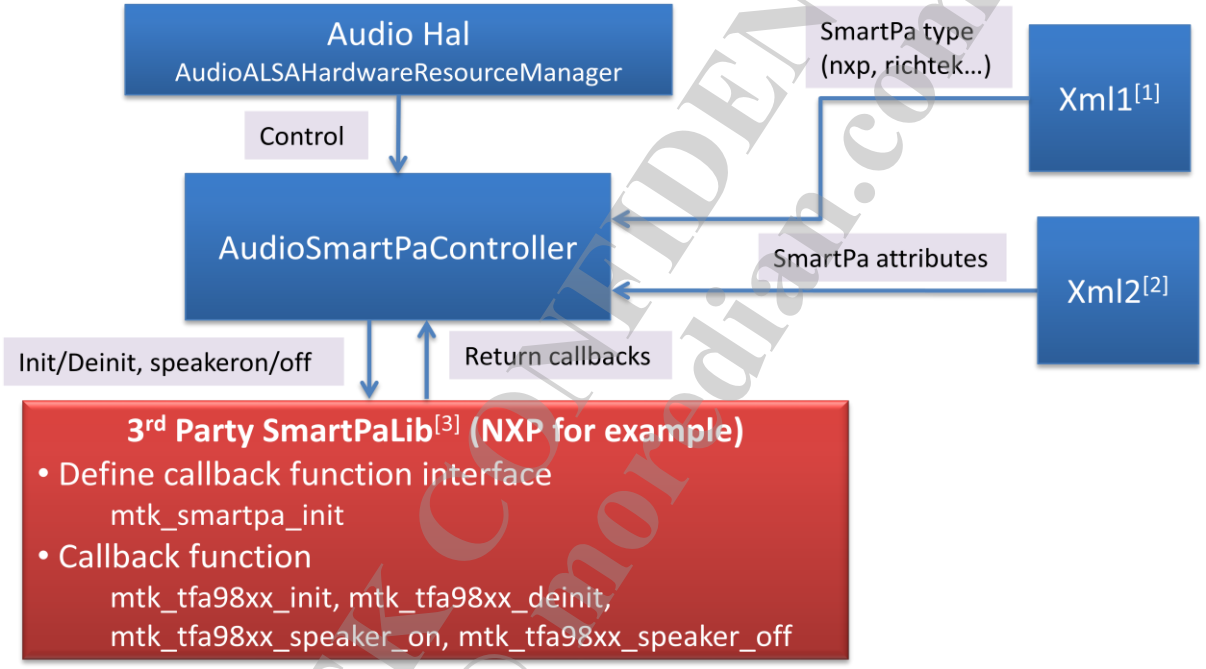
9.1 Overview

SmartPA(Smart Power Amplifier) is a speaker amp IC which includes speaker excursion, thermal protection and even acoustic . Different types of SmartPA can be integrated in MTK platform. For a good maintainability in integrating, class AudioSmartPaController is used to control SmartPA, instead of AudioALSACodecDeviceOutSpeakerNXP. Playback and feedback data transfer through I2s interface. Speaker protection algorithm can be done in AP side or SmartPA chip depends on SmartPA type.



9.2 AudioSmartPaController Introduction

AudioSmartPaController is controlled by AudioALSAHardwareResourceManager in audio initialization and speaker on/off scenario. In audio initialization, AudioSmartPaController will query which SmartPA type is in used from xml1 and the corresponding attributes from xml2. Moreover, AudioSmartPaController will check if the interface "mtk_smartpa_init" is defined. SmartPA library of different vendors can define their own callback functions in mtk_smartpa_init, including init, deinit, speaker on/off. Audio HAL will use these callback functions to do speaker controls.



[1]: /system/vendor/etc/audio_param/AudioParamOptions.xml

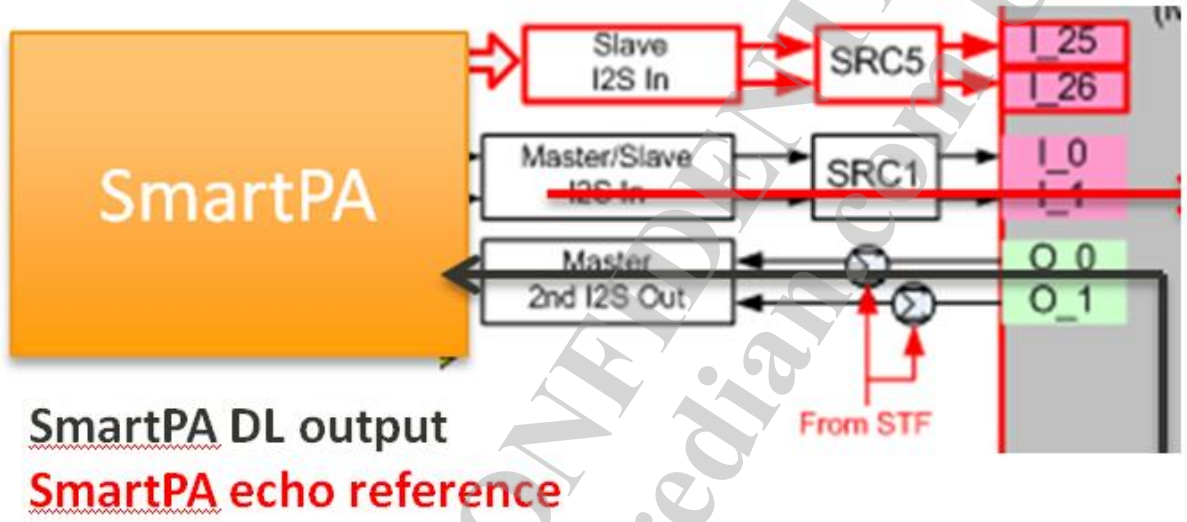
- AudioParamOption.xml will parse the string of MTK_AUDIO_SPEAKER_PATH from ProjectConfig.mk. MTK_AUDIO_SPEAKER_PATH will be introduced in chapter 9.4.

[2]: /system/vendor/etc/audio_param/SmartPa_AudioParam.xml & SmartPa_AudioParamUnitDesc.xml

9.3 SmartPA Control Flow

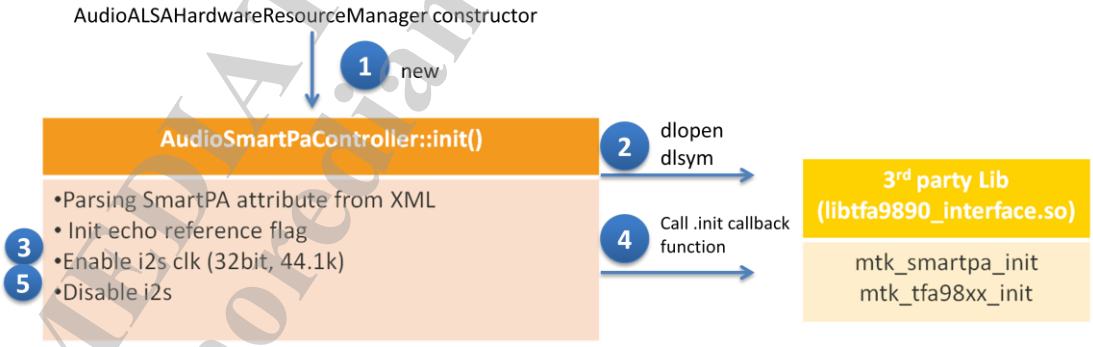
9.3.1 Speaker Path

The reference design of SmartPA i2s is 2nd i2s out and Master i2s in. The diagram below shows data out to 2nd i2s out and echo reference from Master i2s in.



9.3.2 Audioserver Initialization

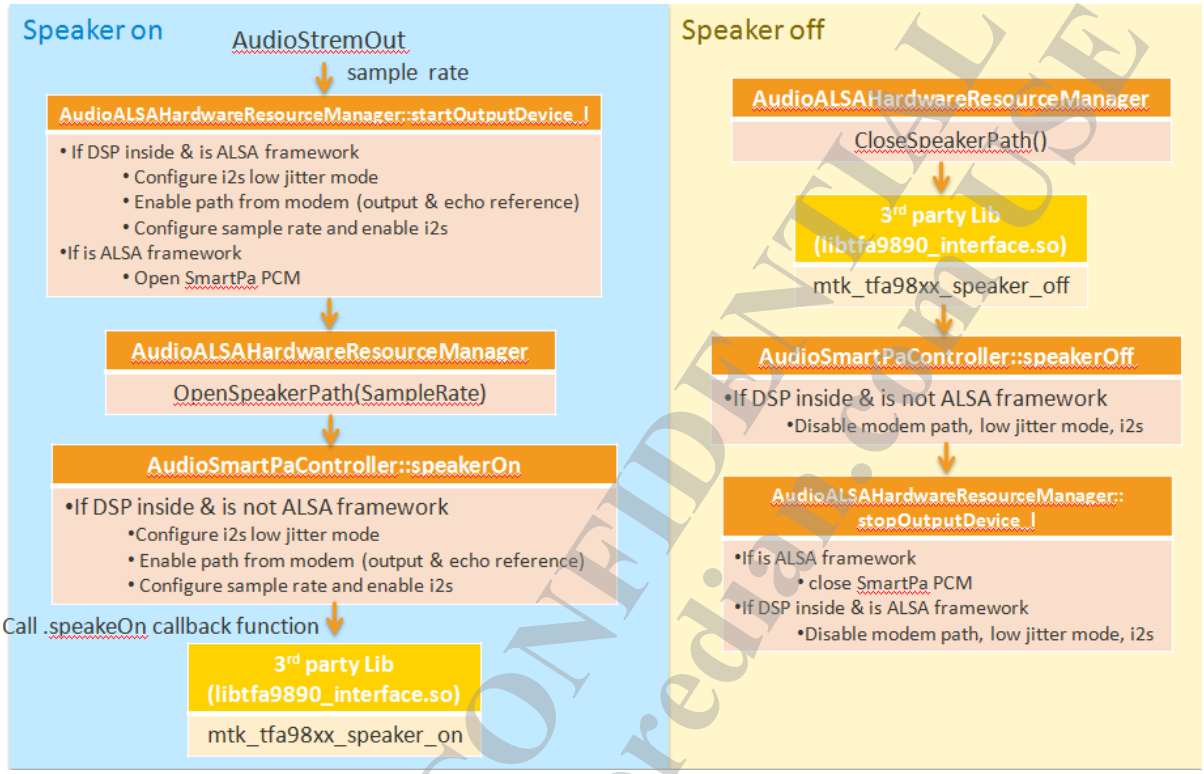
In Audioserver initial stage, resourceManager constructor will call init in smartpa controller. First, it will do parsing SmartPA attribute from xml. If SmartPA has dsp inside, this FO will be in enabled in Android.mk. The echo reference send back after dsp processed. After that, we use dynamic linking to check if function "mtk_smartpa_init" is defined in 3rd party lib. If yes, init callback function will be called. Moreover, some type of SmartPA need to enable i2s before init, such as NXP. Otherwis, it will encounter i2c error.



9.3.3 Speaker Path in Normal Playback/Speech

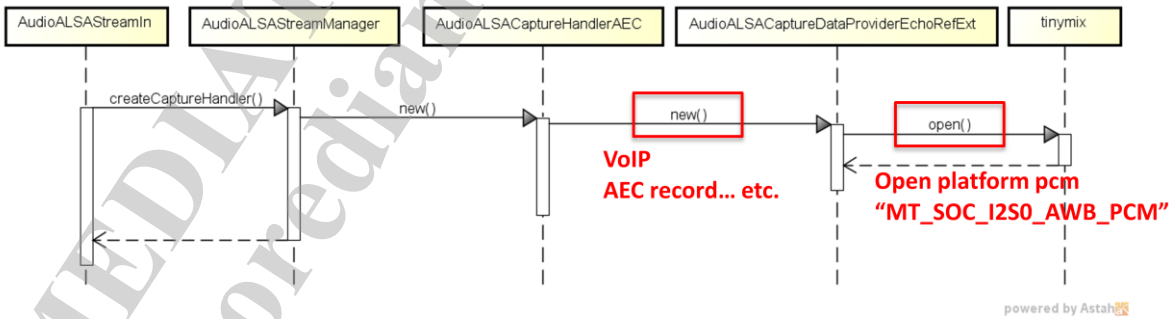
In playback control flow, startOutputdevice is always called. The first thing of this function is to use xml attribute to check if SmartPA is in ALSA framework and DSP inside. If yes, It will enable i2s first and SmartPa pcm is opened. If SmartPA is DSP inside but not in ALSA framework, i2s will be enabled before speaker turn on.

Speaker turn off sequence is basically the opposite way of speaker turn off.



9.3.4 Speaker Path in AEC (Acoustic Echo Cancellation)

The difference between normal playback and AEC scenario is i2s input configuration. I2s in is configured in ALSA platform "MT_SOC_I2S0_AWB_PCM", instead of using kcontrol in user space. If any AEC scenario with speaker mode is triggered, AudioALSACaptureDataProviderEchoRefExt class will the platform pcm and control i2s input.



9.4 Basic Development Information

9.4.1 Device Configuration

There are two configurations of speaker path:

- *MTK_AUDIO_SPEAKER_PATH* in alps/device/mediate/$(project)/ProjectConfig.mk

- Kconfig in alps/kernel-3.18(or kernel-4.4)/arch/arm64/config/$(project defconfig and debug_defconfig)

The setting of MTK_AUDIO_SPEAKER_PATH is as the table below.

Parameter	Description
smartpa_maxim_98926	Maxim SmartPA
smartpa_nxp_tfa9887	tfa9887 is one of NXP SmartPA
smartpa_nxp_tfa9890	tfa9890 is one of NXP SmartPA
smartpa_richtek_rt5509	RT5509 is RichTek SmartPA
int_lo_buf	Other speakers which connect to PMIC interface. Ex:Aiwa

Maxim and RichTek SmartPA will need to configure Kconfig additionally.

SmartPA	Kconfig
Maxim	CONFIG_SND_SOC_MAX98926
RichTek	CONFIG_SND_SOC_RT5509
NXP	CONFIG_MTK_SMARTPA_SOUND

In summary, the setting of speaker path can be referenced in following examples. The file path of ProjectConfig.mk and Kconfig can be referenced above.

1. Maxim SmartPA

- ProjectConfig.mk : MTK_AUDIO_SPEAKER_PATH=smartpa_maxim_98926
- $(project defconfig and debug_defconfig): CONFIG_SND_SOC_MAX98926=y

2. NXP SmartPA

- ProjectConfig.mk : MTK_AUDIO_SPEAKER_PATH=smartpa_nxp_tfa9887(or smartpa_nxp_tfa9890)
- $(project defconfig and debug_defconfig): CONFIG_SND_SOC_RT5509=n
- $(project).dts :

```
mtksmartpa {
        compatible = "mediatek,mtksmartpa";
};
```

3. RichTek SmartPA

- ProjectConfig.mk : MTK_AUDIO_SPEAKER_PATH=smartpa_richtek_rt5509
- $(project defconfig and debug_defconfig): CONFIG_MTK_SMARTPA_SOUND=y

4. Other speaker

- ProjectConfig.mk : MTK_AUDIO_SPEAKER_PATH=int_lo_buf

9.4.2 SmartPA XML Configuration

The SmartPA XML files are in the following path:

alps/device/mediatek/common/audio_param_smartpa/SmartPa_AudioParam.xml
alps/device/mediatek/common/audio_param_smartpa/SmartPa_ParamUnitDesc.xml

The descriptions of the XML attributes are listed in the following table. The attributes will be queried in audio HAL when audioserver initial.

Attribute	Description
have_dsp	Smart PA has dsp inside or not.
is_alsa_codec	SmartPA driver support ALSA framework or not
chip_delay	Group delay time of SmartPA lib (unit: sample)
supported_rate_list	Supported sampling rate
spk_lib_path/ spk_lib64_path	defined 3rd party smartPA ibrary path.
codec_ctl_name	Only used by SmartPA in AP side. Ex: Maxim
is_apll_needed	SmartPA with DSP inside need to be enabled APLL. Note: MTK i2s interface jitter is above 10 ns
is_i2s_need_in_init	I2s clock need to be enabled or not before SmartPA initialization. Ex: NXP

❖ **SmartPa_ParamUnitDesc.xml**

```
<?xml version="1.0" encoding="UTF-8"?>
<ParamUnitDesc version="1.0">
    <CategoryTypeList>
        <CategoryType name="Speaker type">
            <Category name="smartpa_nxp_tfa9887"/>
            <Category name="smartpa_nxp_tfa9890"/>
            <Category name="smartpa_maxim_98926"/>
            <Category name="smartpa_richtek_rt5509"/>
        </CategoryType>
    </CategoryTypeList>
    <ParamUnit>
        <Param name="have_dsp" type="int"/>
        <Param name="is_alsa_codec" type="int"/>
        <Param name="chip_delay_us" type="uint"/>
        <Param name="supported_rate_list" type="uint_array"/>
        <Param name="spk_lib_path" type="string"/>
        <Param name="spk_lib64_path" type="string"/>
        <Param name="codec_ctl_name" type="string"/>
        <Param name="is_apll_needed" type="int"/>
        <Param name="is_i2s_need_in_init" type="int"/>
    </ParamUnit>
</ParamUnitDesc>
```

Add new type of SmartPa here

❖ **SmartPa_AudioParam.xml**

```
<AudioParam>
    <ParamTree>
        <Param path="smartpa_nxp_tfa9887" param_id="0"/>
        <Param path="smartpa_nxp_tfa9890" param_id="1"/>
        <Param path="smartpa_maxim_98926" param_id="2"/>
        <Param path="smartpa_richtek_rt5509" param_id="3"/>
    </ParamTree>
    <ParamUnitPool>
        <ParamUnit param_id="0">
            <Param name="have_dsp" value="1"/>
            <Param name="is_alsa_codec" value="0"/>
            <Param name="chip_delay_us" value="22000"/>
            <Param name="supported_rate_list" value="8000,11025,12000,1600
            <Param name="spk_lib_path" value="libtfa9887_interface.so"/>
            <Param name="spk_lib64_path" value="libtfa9887_interface.so"/>
            <Param name="codec_ctl_name" value=""/>
            <Param name="is_apll_needed" value="1"/>
            <Param name="is_i2s_need_in_init" value="1"/>
        </ParamUnit>
        <ParamUnit param_id="1">
            <Param name="have_dsp" value="1"/>
            <Param name="is_alsa_codec" value="0"/>
            <Param name="chip_delay_us" value="22000"/>
            <Param name="supported_rate_list" value="8000,11025,12000,1600
            <Param name="spk_lib_path" value="libtfa9890_interface.so"/>
            <Param name="spk_lib64_path" value="libtfa9890_interface.so"/>
            <Param name="codec_ctl_name" value=""/>
            <Param name="is_apll_needed" value="1"/>
            <Param name="is_i2s_need_in_init" value="1"/>
```

Every SmartPa has an index. An index has a corresponding attribute set. Customers can add attribute set if they add a new SmartPa.

9.4.3 SmartPA Configuration in Audio HAL

The feature option *SMART_PA_SUPPORT* defined in Android.mk is used in audio HAL. The following files are SmartPa related. You can search keyword "*SMART_PA_SUPPORT*" in every file to find SmartPa related code.

File Path	File
vendor\mediatek\proprietary\hardware\audio\common\V3\aud_drv	AudioALSAPlaybackHandlerFast.cpp AudioALSAHardware.cpp AudioALSAPlaybackHandlerNormal.cpp AudioALSACaptureHandlerAEC.cpp AudioSmartPaController.cpp
vendor\mediatek\proprietary\hardware\audio\common\V3\include	AudioALSAHardwareResourceManager.h AudioSmartPaController.h
vendor\mediatek\proprietary\hardware\audio\$(platform)\aud_drv	AudioALSAHardwareResourceManager.cpp
vendor\mediatek\proprietary\hardware\audio\common\speech_driver	SpeechDriverNormal.cpp
vendor/mediatek/proprietary/hardware/audio/$(platform)	Android.mk

9.4.4 SmartPA Library & Parameter Configuration

❖ **SmartPA library Configuration**

In SmartPA library, 3rd party will need to implement the callback functions (init, deinit, speakerOn, speakerOff). These callback functions will be used in Audio HAL to control SmartPA. The location should be placed in the following path:

alps/vendor/mediatek/proprietary/hardware/smartpa/

❖ **SmartPA Parameter**

The default SmartPA parameter binary file can be placed in alps/device/mediatek/$(platform)/smartpa_param. If customers want to use different parameter in specific project, parameter can be placed in /device/mediatek/$(project). Vendors can load their parameter binary file in init callback function.

Here takes RichTek SmartPA(RT5509) as example. Put the binary file “rt5509_param” under alps/device/mediatek/$(platform)/smartpa_param and add the configuration below:

Files	Configuration
alps/device/mediatek/common/device.mk	ifeq ($(strip $(MTK_AUDIO_SPEAKER_PATH)),smartpa_richtek_rt5509) PRODUCT_COPY_FILES += device/mediatek/$(shell echo $(MTK_PLATFORM) \| tr '[A-Z]' '[a-z]')/smartpa_param/rt5509_param:$(TARGET_COPY_OUT_VENDOR)/etc/smartpa_param/rt5509_param:mtk PRODUCT_COPY_FILES += \ $(call add-to-product-copy-files-if-exists, $(MTK_TARGET_PROJECT_FOLDER)/rt5509_param:$(TARGET_COPY_OUT_VENDOR)/etc/smartpa_param/rt5509_param:mtk) endif

10 Mediatek Aurisys and Open DSP

10.1 Overview

The requirement for sound quality enhancements that add listening pleasure has been increasing. To enrich the listening experience of using mobile devices, there is an increasing number of sound processing solutions provided by vendors.

Aurisys is introduced to facilitate the sound processing solution development and use of MTK platforms. Aurisys is a framework constructed upon Android Audio Framework. It includes standardized interfaces for sound processing and tuning, integrated DSP sound subsystem, and software debug interface. This concept of Aurisys will be briefly described in this document.

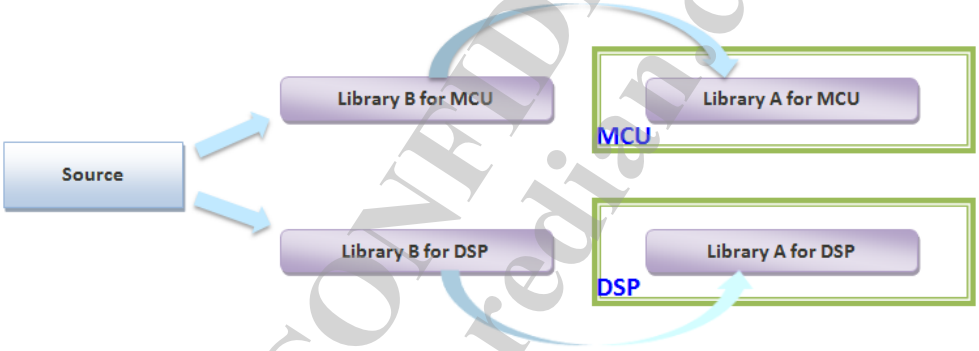


Figure 59. Aurisys Concept

10.1.1 Aurisys Structure

As depicted in above figure, the Aurisys structure contains Scene Handler, Library Manager, integrated DSP framework, Modem/Audio HW subsystems, and standardized software interfaces:

- **Aurisys Scene Handler (ARSH)**
  The Aurisys Scene Handler is created as the middleware between the audio system and the sound processing IPs. It is created scene by scene. For example, the Aurisys Playback Handler is served for playback effects in the scene of playback. It provides interfaces to call the corresponding IPs and manages the IPs.
- **Aurisys Library Manager (ARLM)**
  The Aurisys Library Manager maintains the library information of each Aurisys Scene Handler. It contains the list and status of sound IPs.
- **Aurisys Software Interface (ARSI)**
  ARSI is the abbreviation of Aurisys Software Interface. It is provided for interfacing with sound enhancing tasks. To process sounds, a unified interface which is portable between MTK platforms is provided. The interface to parse and transmit parameters between PC tool, APMCU, and DSP is also included. The interface is designed for ease of use while being general enough so that new algorithms can be easily added to the existing frameworks. The difference between ARSI and ARSH is that ARSI

provides interface between ARSH and sound IPs; while ARSH provides interface for Android Audio HAL to request for processing sounds.

- **Modem**

  The modem means the modem IC. The MTK internal chipset contains a DSP to process voice enhancement and voice codec. In Aurisys structure, we disable the voice enhancement, but reserve codec in modem IC so that we can just add algorithms in the integrated open DSP in the Application side.

- **Audio HW**

  It refers to the interface between processors and HW devices to transfer sound data. The devices include speakers, microphones, earphones, BT devices, USB devices, etc.

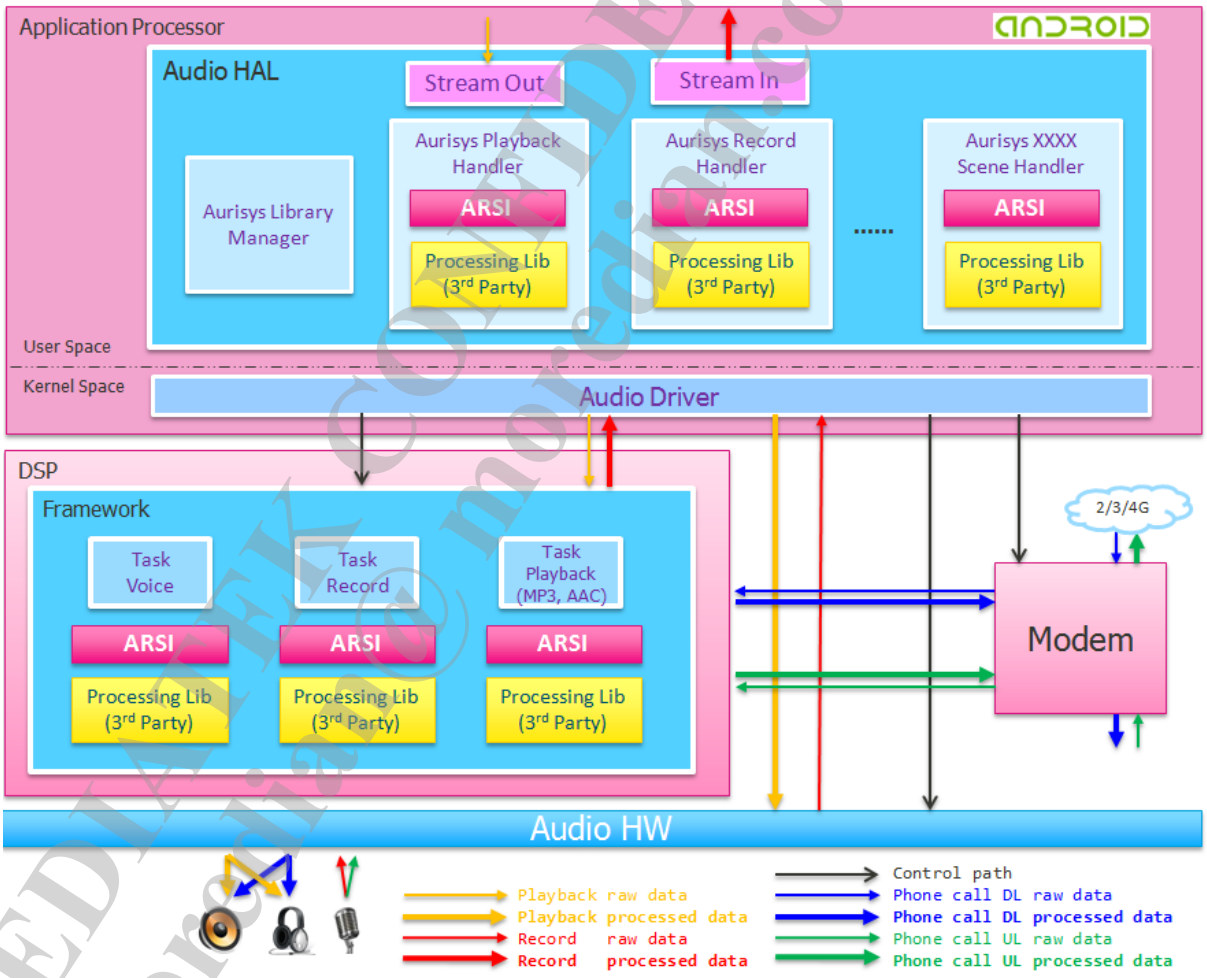


Figure 60. Aurisys Structure

10.2 Sound Trigger

10.2.1 Background

The feature of always-on listening to voice command is so called sound trigger or VOW (voice wakeup). The sound trigger needs three key factors

- Always on detection
  - Automatically detect voice input pressure (7 days x 24 hours)
- Low power detection
  - Screen most unwanted voice input in lowest power domain
- Smart detection
  - Only correct key word can power up and unlock phone

To accomplish above requirement, we provide low power VAD (voice activity detection) in PMIC as VOW phase1 to screen most unwanted noise or voice input. When the detection of phase1 is passed, the voice PCM will be sent to Open DSP for detection of voice command, such as “OK Google”, “open camera”. This stage is so-called VOW phase2. During these two phase2, AP could be in suspend and save most power. When detection of phase2 is also passed, it will wake up AP to connect more application for rich experiences.

- Specification
  - Sampling rate: 16 KHz
  - Single microphone. Multi-microphone is not supported
  - Available computing power for VOW phase2 is 82MHz
  - Seamless record is supported.

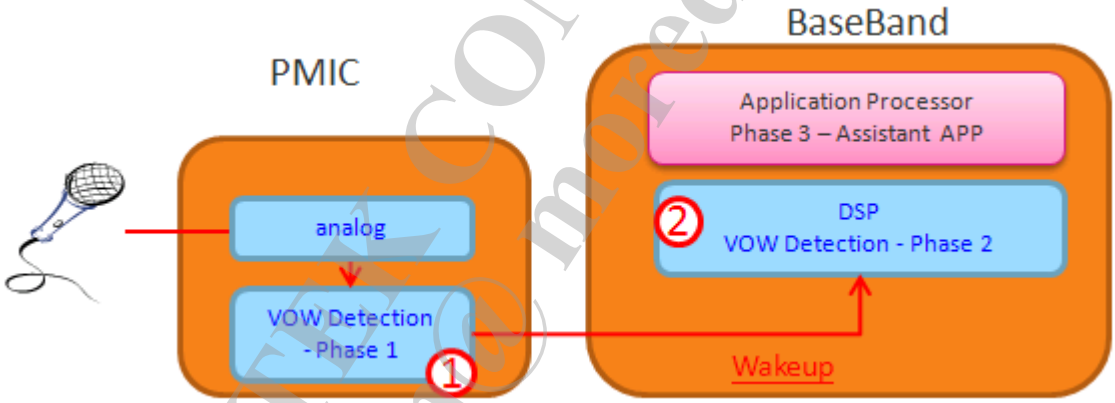


Figure 61. VOW concept

10.2.2 AP framework

After Android Lollipop, the framework Voice Interaction Manager Service (VIMS) is introduced to all. It defines the framework and HAL of sound trigger.

We implement the sound trigger HAL and Mediatek voice wakeup APP based on the framework of VIMS. Customers can also replace the algorithm in Open DSP for differentiation. If algorithm is replaced, then the corresponding sound model and training APP may need to be replaced, too.

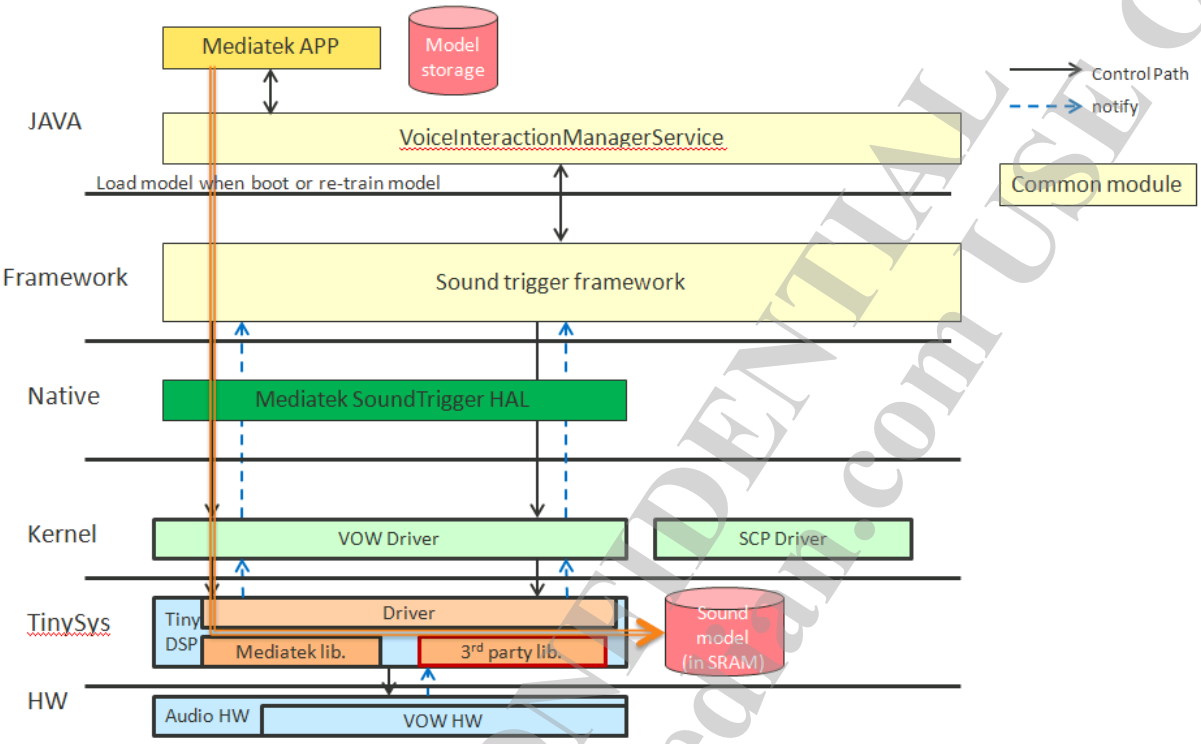


Figure 62. VOW implementation based on Android Framework "Voice Interaction Manager Service"

10.2.3 Keyword customization

Mediatek VOW solution provides two types of voice recognition

- Mode1. Speaker independent
  - Support customized fixed command
  - Need training
- Mode2. Speaker dependent
  - Support arbitrary command
  - Need training

For Mode 1, if customer wants to customize the unique fixed command, we can also help to generate corresponding speech model based on the audio record files provided by customer.

The suggested format and quantity of audio record files

- At least 35 males and 35 females, each one speaks the same keyword 10 times in single file. Each speaker records in different files.
- Record file format: 48 KHz, 16bits, PCM or Wav
- Record environment: clean and noisy-free office or meeting room.
- Record device: use the mass production phone (for example, your X20 project to record). Distance between phone and mouth is around 20~30cm. Speakers speak in normal and stable volume and speed.

Ask ACS for more details.

10.2.4 Seamless Record and Hotword Record

For some application, seamless record or Hotword record from sound trigger device is needed.

- Seamless Record

  For applications that need "seamless" stream after keyword is recognized in VOW phase2, the AudioRecord input source could be set as AUDIO_SOURCE_HOTWORD instead of AUDIO_SOURCE_VOICE_RECOGNITION or AUDIO_SOURCE_MIC or others. In this way, application can get seamless stream from sound trigger device.

```
typedef
enum {
        AUDIO_SOURCE_DEFAULT             = 0,
        AUDIO_SOURCE_MIC                 = 1,
        AUDIO_SOURCE_VOICE_UPLINK        = 2,
        AUDIO_SOURCE_VOICE_DOWNLINK      = 3,
        AUDIO_SOURCE_VOICE_CALL          = 4,
        AUDIO_SOURCE_CAMCORDER           = 5,
        AUDIO_SOURCE_VOICE_RECOGNITION   = 6,
        AUDIO_SOURCE_VOICE_COMMUNICATION = 7,
        AUDIO_SOURCE_REMOTE_SUBMIX       = 8, /* Source for the mix to be presented remotely.      */
                                              /* An example of remote presentation is Wifi Display */
                                            /*  where a dongle attached to a TV can be used to   */
                                              /*  play the mix captured by this audio source.      */
        AUDIO_SOURCE_CNT,
        AUDIO_SOURCE_MAX                 = AUDIO_SOURCE_CNT - 1,
        AUDIO_SOURCE_HOTWORD             = 1999, /* A low-priority, preemptible audio source for
                                                    for background software hotword detection.
                                                    Same tuning as AUDIO_SOURCE_VOICE_RECOGNITION.
                                                    Used only internally to the framework. Not exposed
                                                    at the audio HAL. */
    } audio_source_t;
```

- Hotword Record

  For applications that need directly using sound trigger device to get stream, they can set input source of AudioRecord as AUDIO_SOURCE_HOTWORD and start record to get Hotword stream.

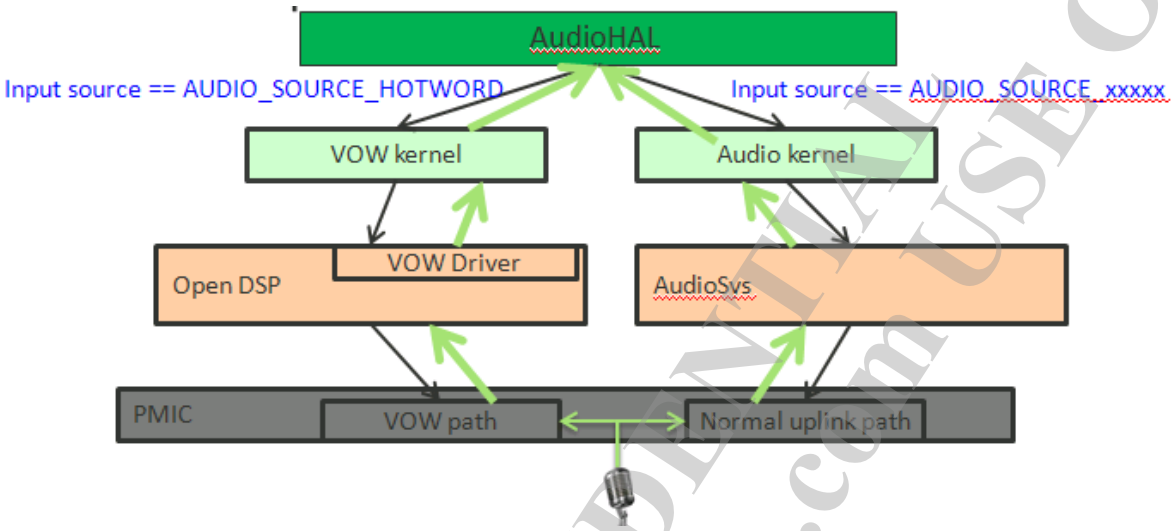


Figure 63. Hotword Record vs. Normal record

10.2.5 Limitation

Since normal record and sound trigger use the same microphone, sound trigger can't be concurrent with normal record and phone call scenarios.

10.3 Summary

The information of supported features is listed as follows. All of these features can be implemented by ARSI-PROC and utility functions.

- Sound Trigger
  - We provide a hardware VAD (voice activity detection) to reduce the active period of the processor. We also provide a MTK proprietary sound trigger library to do the keyword detection. The sound trigger algorithm can be replaced by a vendor's solution. Besides, we also support digital microphone with vendor's VAD and keyword detect solutions.
- Speaker Protection
  - The speaker protection solution is possible to be supported on both application processor and DSP.
- Recording Enhancement/VOIP
  - The sound enhancement for recording and VOIP can be processed in the application processor. The MTK proprietary library for recording and VOIP are only supported on application processor.

Besides the above mentioned features, other applications can also be carried out. For example, Proximity is possible to be implemented by the provided utility functions. However, considering the limited resource of DSP, the concurrent cases should be designed carefully. **Please visit Mediatek Website and customer support Website (MediaTek On-Line, MOL) to get more information.**

11 Audio Tuning Tool

11.1 Introduction

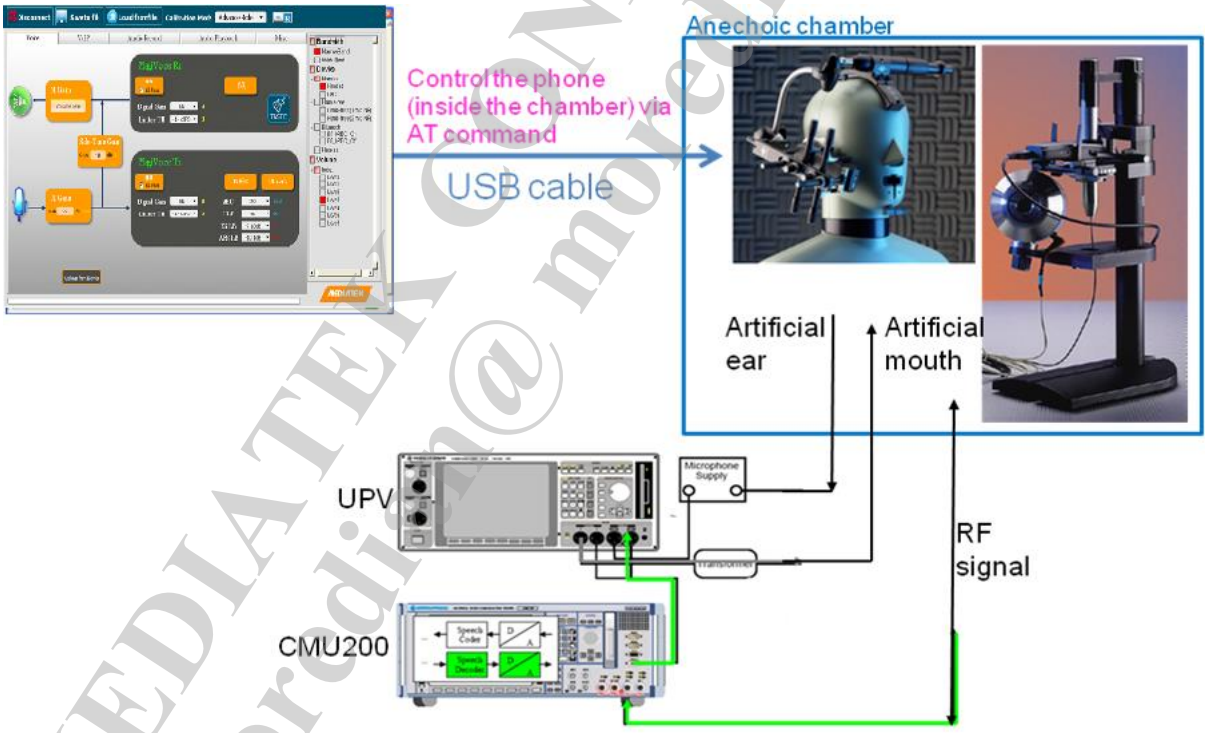
The audio and speech parameters calibration is important step during the phone design. The audio tuning tool helps engineers to tune parameters run-time more efficiently.

So user doesn't need to take the phone out from the chamber and re-setup it, save time and convenient.

This section will introduce the ability of each audio tuning page in Audio Tuning Tool of V2.2 which support on MT6771.

For detail skill audio tuning skills training, please reference from ACS support team for audio training document.

Below diagram descript that how to calibration by tools



11.2 Classification of each functional block

In the top bar of tool, there're some buttons lists:

- **Disconnect**
  Audio tuning tool can be disconnected with target device immediately.

- **Save to file**
  Save the parameters by ini files.
- **Load from file**
  Load the parametes from ini files.
- **Calibration Mode**

There're two calibration modes supported, only in **Voice** Tuning pages.

- **Standard Mode** for basic tuning. Here lists basic 4 speech devices in the right side of tuning page.

Handset/Hands-free/ Bluetooth/Headset

The right side displays the categories of the parameter architecture of audio tuning.

- **Advanced Mode** for advanced tuning. Here lists more detail device types includes the 4 basic types.

Moreover, the speech parameters of each volume index can be stored individually.

The 2nd layer bar list all the tuning pages which are distinguished by audio scenarios.

Here lists Voice, VoIP, Audio Record, Audio Playback and Misc pages.

The right block defines the **categories** of parameters by tree architecture.

For example, in the diagram below, here lists 2 bandwidths and 4 speech devices in the right side category in the voice tuning page.

It means the parameters in this page can be stored by each set of Bandwidth-Device individually.

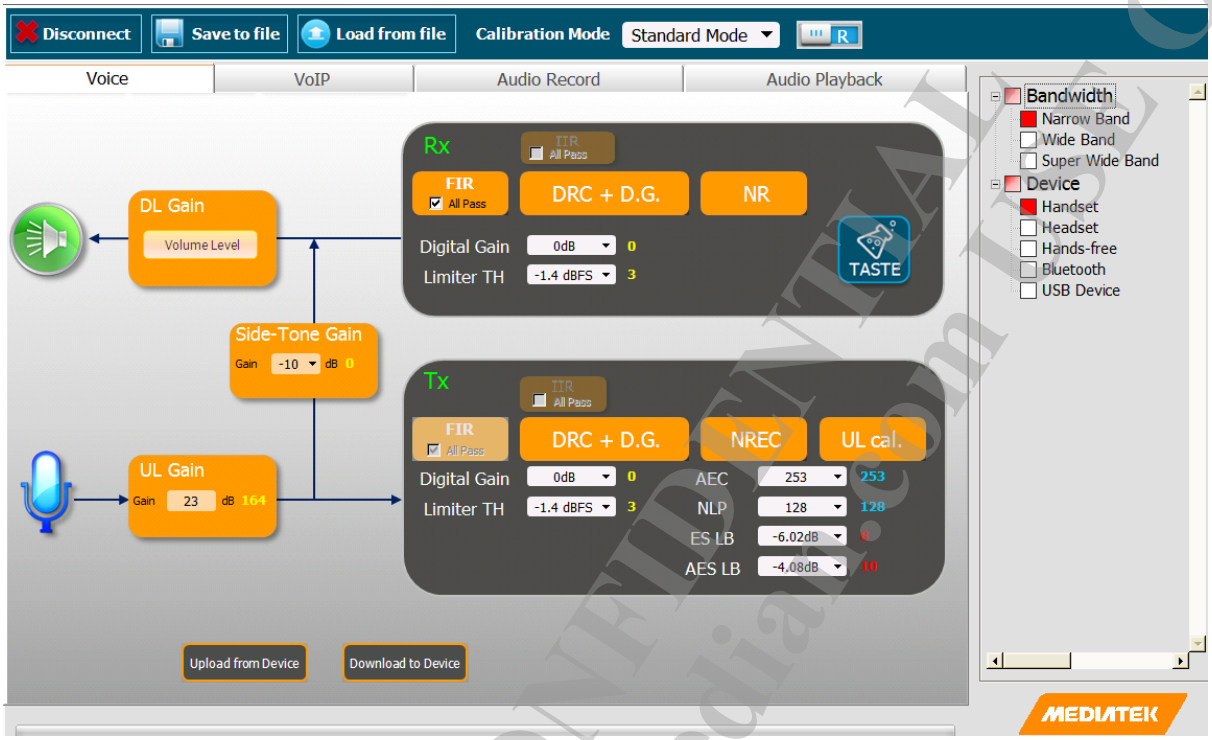
In the most right button of top bar, Read or Write mode can be selected by engineer.

- **Read mode**

Upload from device button shows in the bottom list. Engineers can click the button for uploading parameters from target device and see the parameters in each GUI.

- **Write mode**
- **Download to device** button shows in the bottom list. Engineers can click the button for downloading parameters to target device.

Moreover, in the advanced mode and download mode , engineer can copy parameters with the same device group. (Please reference ACS for more detail training)



In the following section, all tuning pages will be showed sequentially.

11.3 Voice Page

In Voice tuning page, Speech parameters during voice call can be tuned.

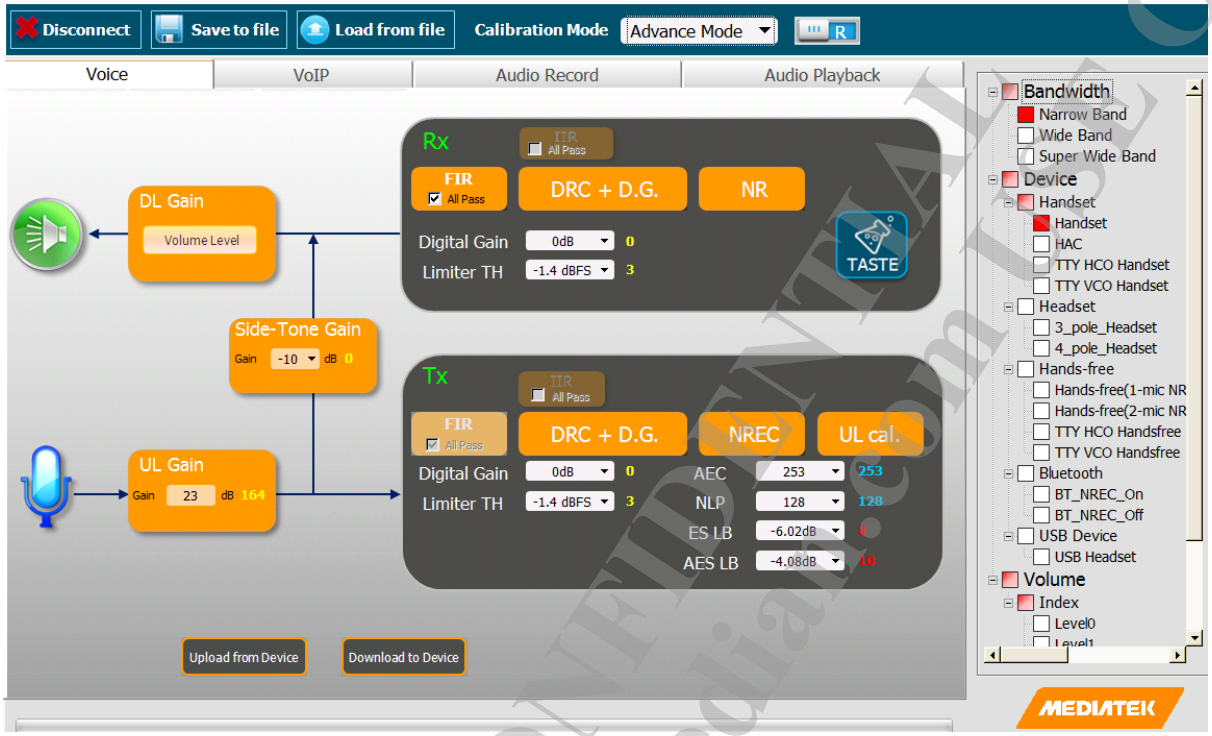
The right block displays the categories of the parameter architecture of audio tuning.

Standard Mode lists basic 5 speech devices in the right side of tuning page.

- Handset/Hands-free/ Bluetooth/Headset/USB Device

- **Advanced Mode** for advanced tuning. Here lists more detail device types includes the 5 basic device groups.
  - Handset: Handset, HAC, TTY_VCO_Handset and TTY_HCO_Handset
  - Hands-free: Hands-free(1-mic NR), Hands-free(2-mic NR), HAC, TTY_VCO_Handsfree and TTY_HCO_Handsfree
  - Bluetooth: BT_NREC_On and BT_NREC_Off
  - Headset: 3_pole_Headset and 4_pole_Headset
  - USB Device: USB Headset

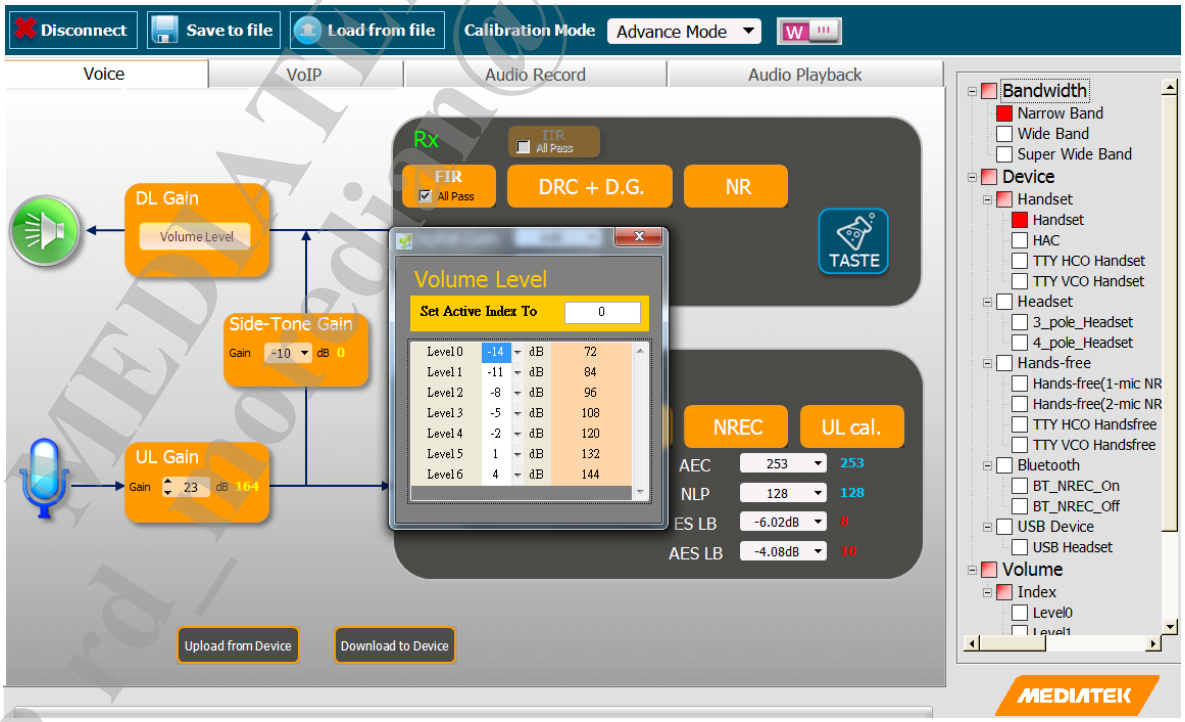
Moreover, the speech parameters of each **volume index** can be stored individually.



The left part is the Volume Gain setting part.

- DL Gain
- Side-Tone Gain
- UL Gain

The digital and analog mixed gain can be adjusted in the pop-up window.



The middle part is the speech parameters of algorithm tuning part.

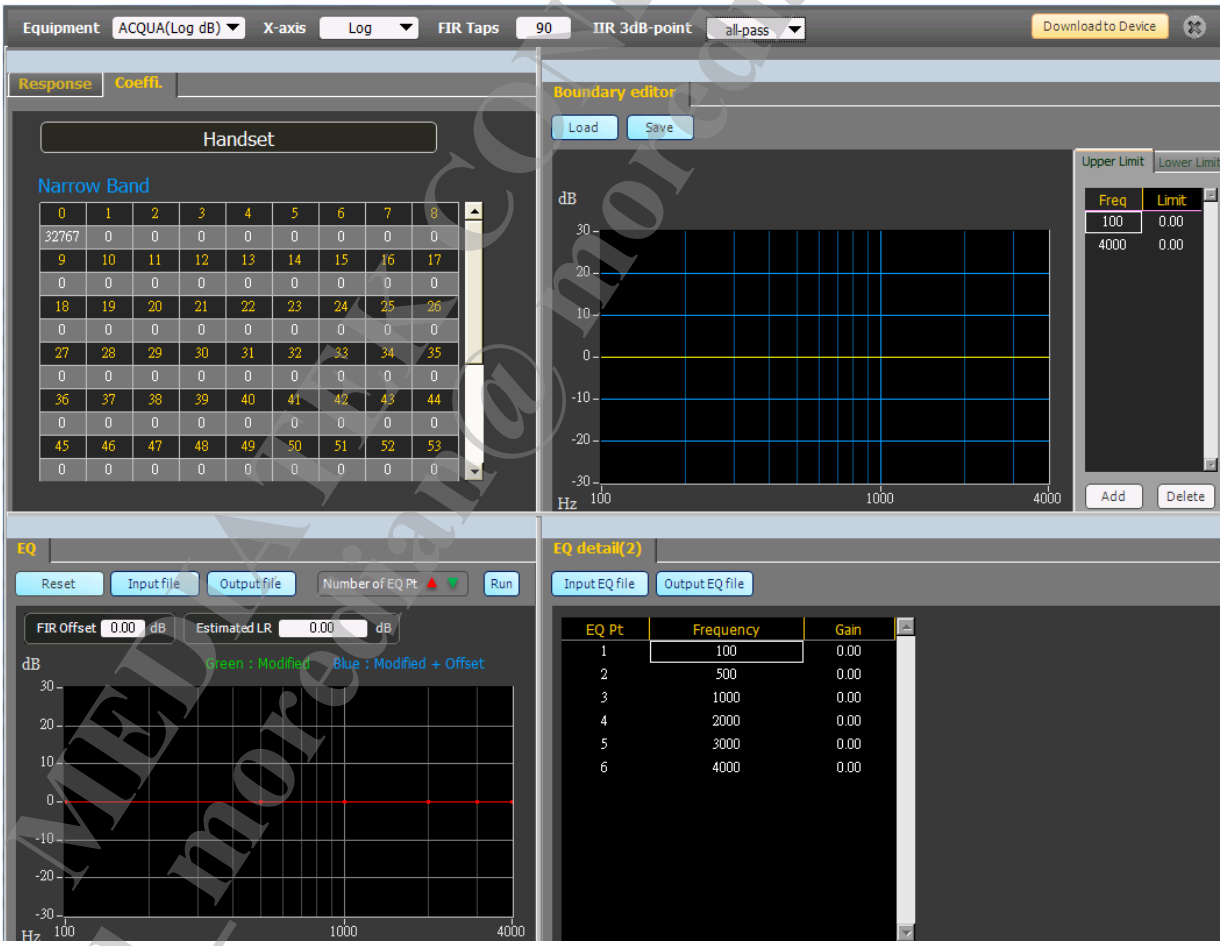
- **FIR/IIR**

If clicking the **FIR/IIR** button, the Acoustic FIR Tuning page will pop-up.

The TX and RX FIR/IIR can be adjusted in the tuning page.

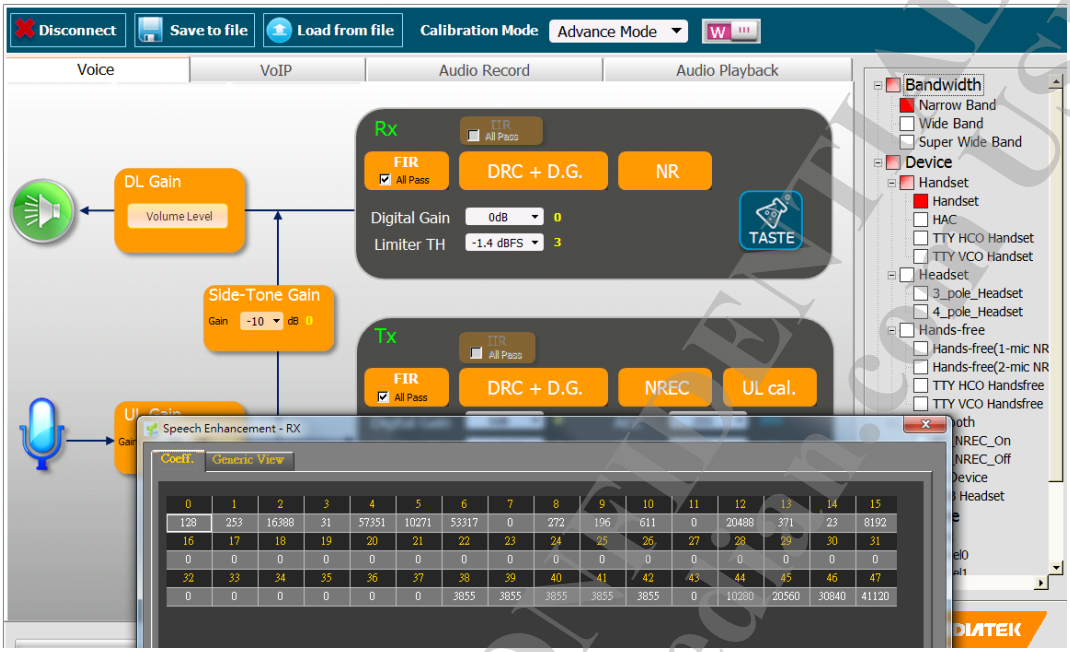
In AudioParamOptions.xml, if MTK_IIR_ENH_SUPPORT=yes, then IIR button exists.
If IIR is enabled mode parameter10, then the button can be clicked by user; otherwise, the button would be grey and can't not be clicked.

Bit	Bit 5	Bit 4	Bit 3	Bit 2
Function	RX FIR	RX IIR	TX FIR	TX IIR
Enable/disabled	1/0	1/0	0/1	1/0



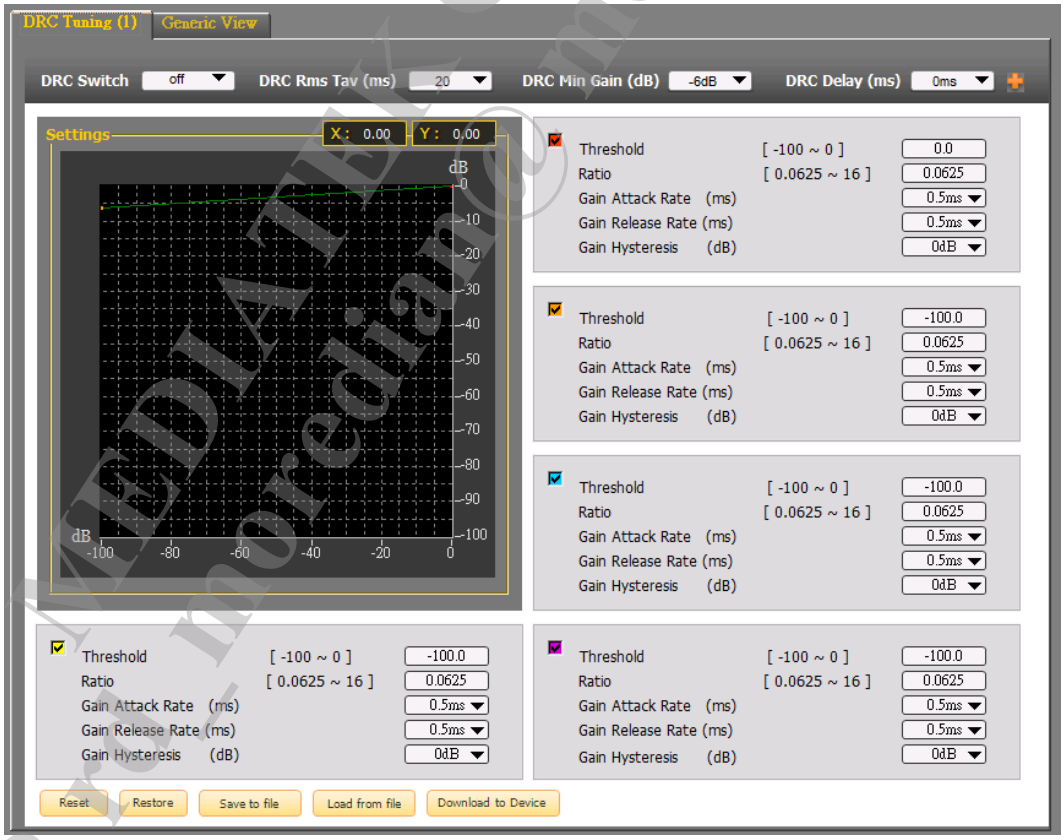
- NR/NREC

  If click the **NR/NREC** button, the window contains 48 speech mode parameters will pop-up.



- **DRC+D.G**.

If click the **DRC+D.G** button, the DRC Tuning page will pop-up.

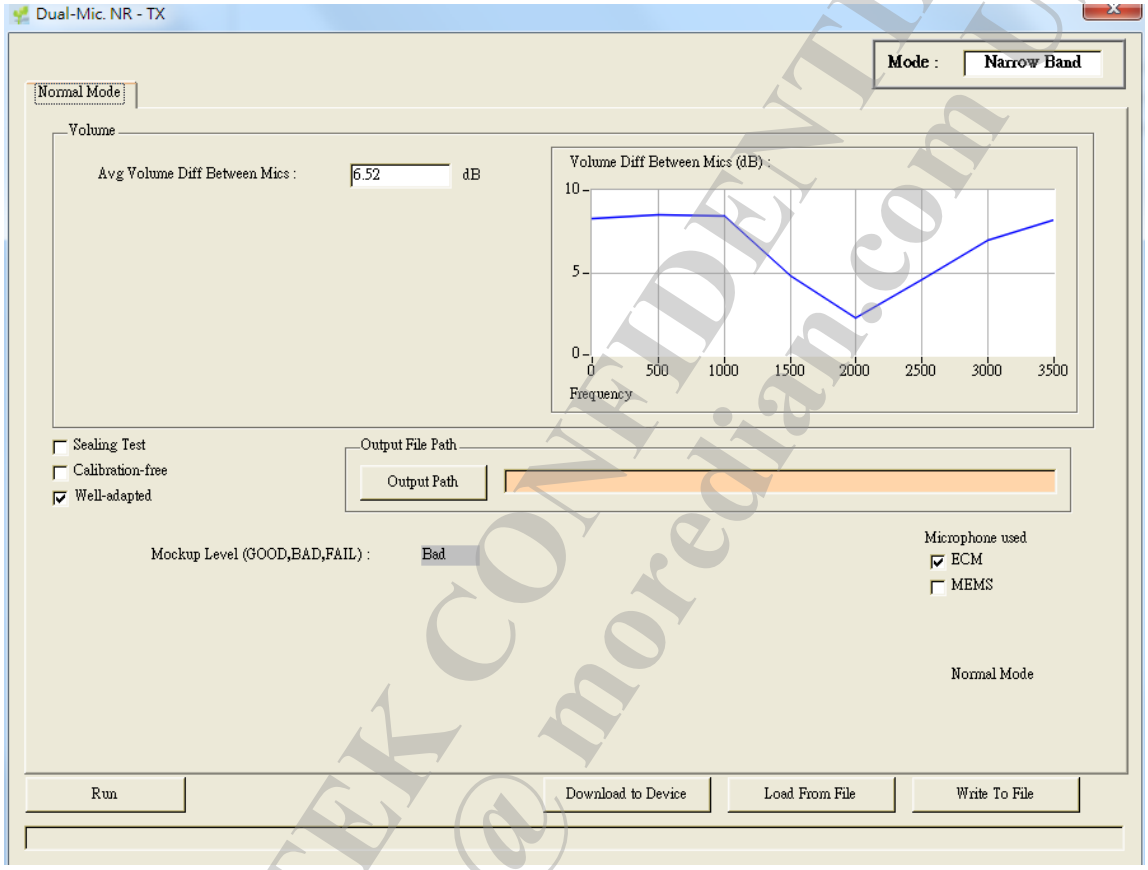


- UL Cal

  If clicking the **UL Cal** button, the Uplink Calibration page will pop-up.

  The DMNR calibration can be processed in the tuning page.

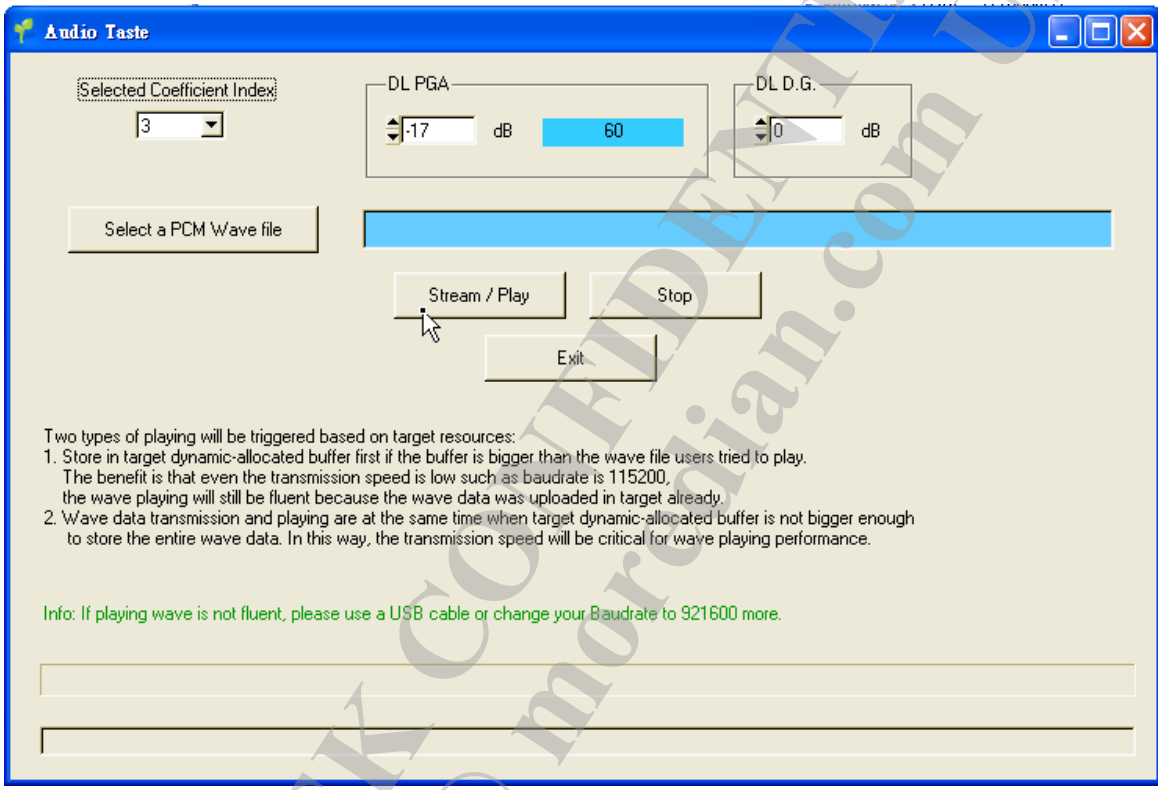
  Only device Handset and hands-free(2-mic NR) need to do the uplink calibration. The button won't exist if other devices be chosen.



- Audio Taste

By double clicking TASTE icon, Audio Taste test page will pop-up.

In this page, tester can play a mono audio wave file with 16k sample rates in the downlink direction. During the playing, downlink PGA gain and downlink digital gain can be adjusted to check the amplitude and effect.



11.4 VoIP Page

The right block displays the categories of the parameter architecture of audio tuning: 5 VoIP devices

- Hands-free/ Handset /Headset/ Bluetooth/USB Device

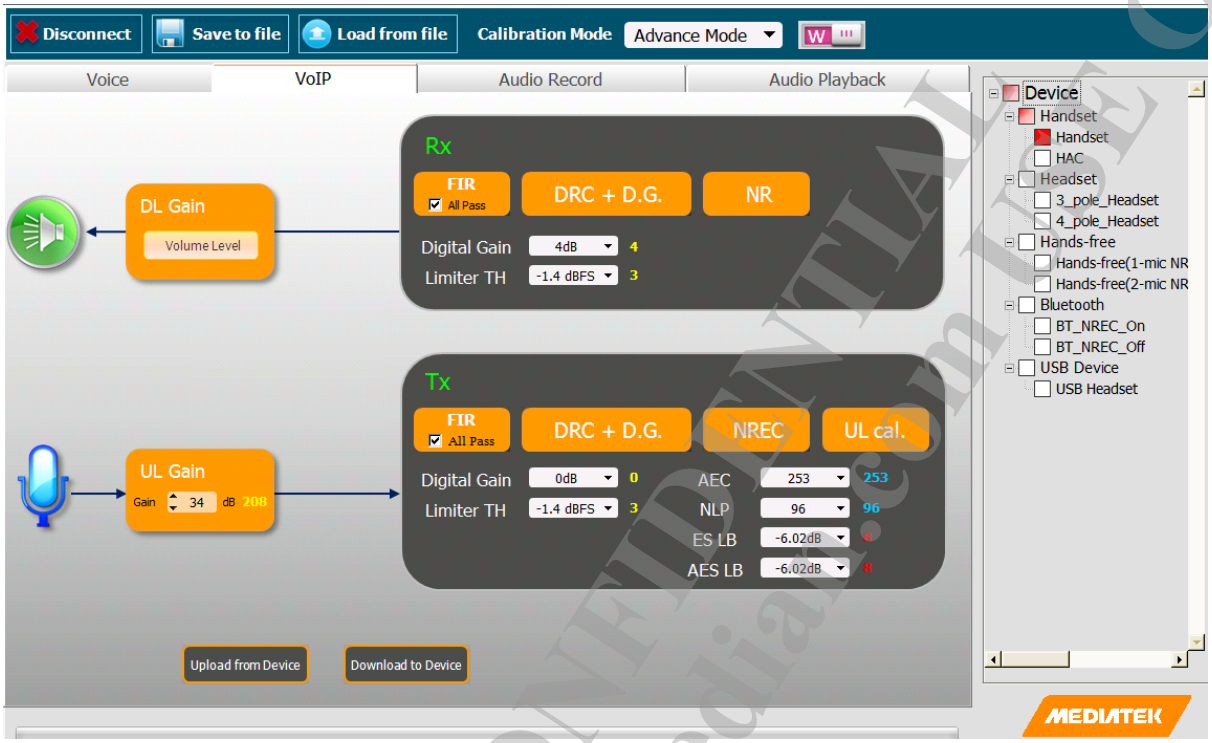
The left functional block shows the Volume Tuning block and Algorithm Tuning block.

Volume Tuning block

- DL Gain:
  - Gain: Downlink analog gain.
  - Volume Level: Downlink digital gain.
- UL Gain: Uplink digital and analog mixed gain.

Algorithm Tuning block.

- The UL/DL enhancement design are as same as voice, Please refer to 11.3 Voice Page



11.5 Audio Record Page

The right block displays the categories of the parameter architecture of audio tuning:

- Application – Device
  Application includes Sound record, Camera recording, ASR improvement and Voice reorganization & CTS verifier.
  Device includes Headset, Handset, Bluetooth and USB

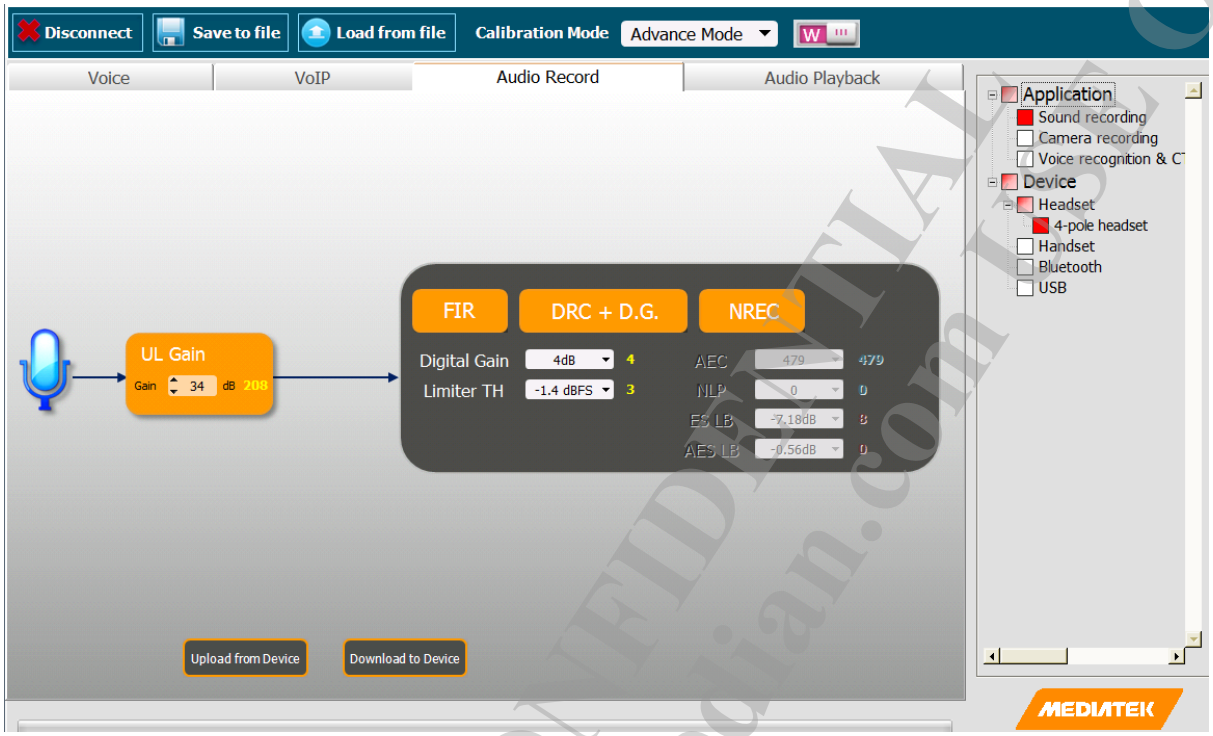
The left functional block shows the Volume Tuning block and Algorithm Tuning block.

Volume Tuning block

- UL Gain: Uplink digital and analog mixed gain.

Algorithm Tuning block.

- Magi Voice algorithm in TX direction is supported.
- The Uplink FIR can also be adjusted.



11.6 Audio Playback Page

The right block displays the categories of the parameter architecture of audio tuning:

- Volume type – Device
  Volume type includes Ring and Media.
  Device includes Headset, Speaker, Headset+Speaker and USB

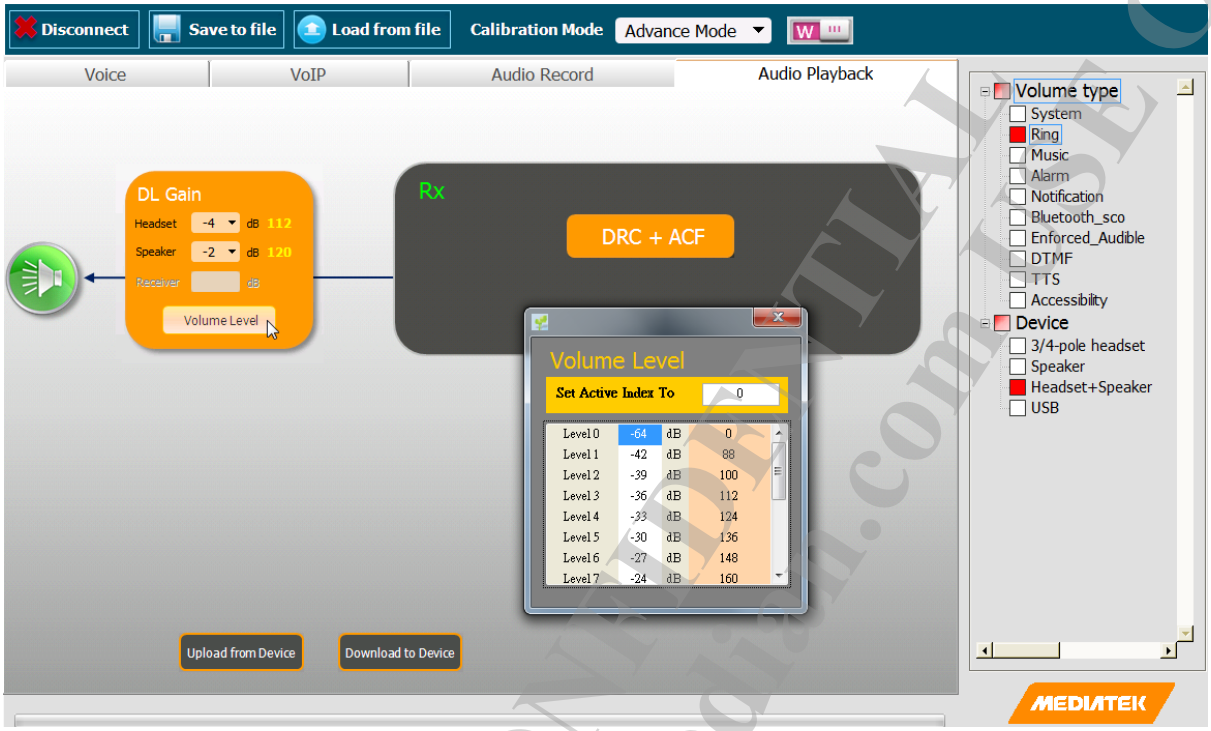
The left functional block shows the Volume Tuning block and Algorithm Tuning block.

Volume Tuning block

- DL Gain:
  - Headset/Speaker: Downlink analog gain.
  - Volume Level: Downlink digital gain.

Algorithm Tuning block.

- BesSound algorithm in RX direction is supported.
- DRC+ACF



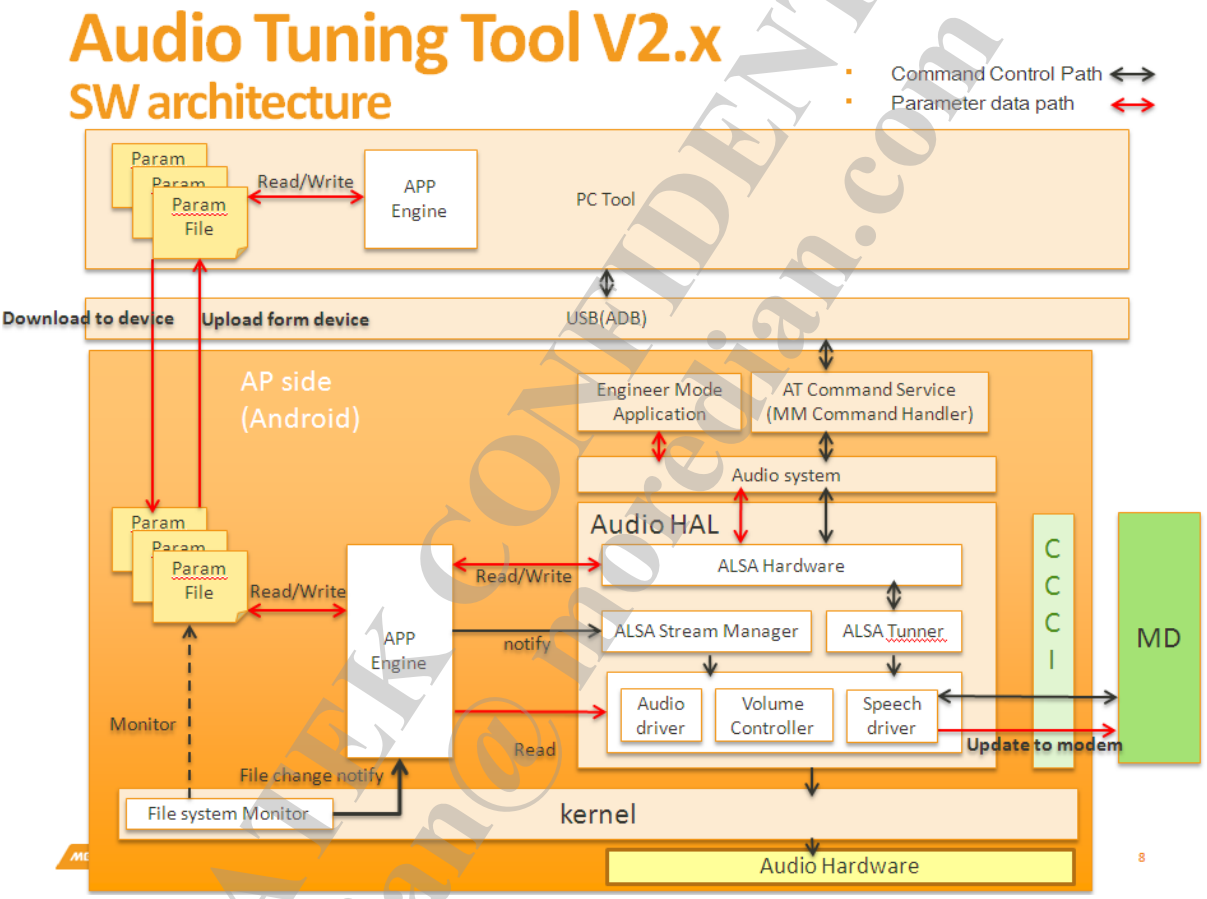
11.7 SW Architecture

The diagram below shows the SW architecture in target device.

Audio Tuning Tool (PC Tool) connects with target devices via USB (ADB) protocol.

The parameters are stored with XML format which are parsed by Audio parameter parser engine.

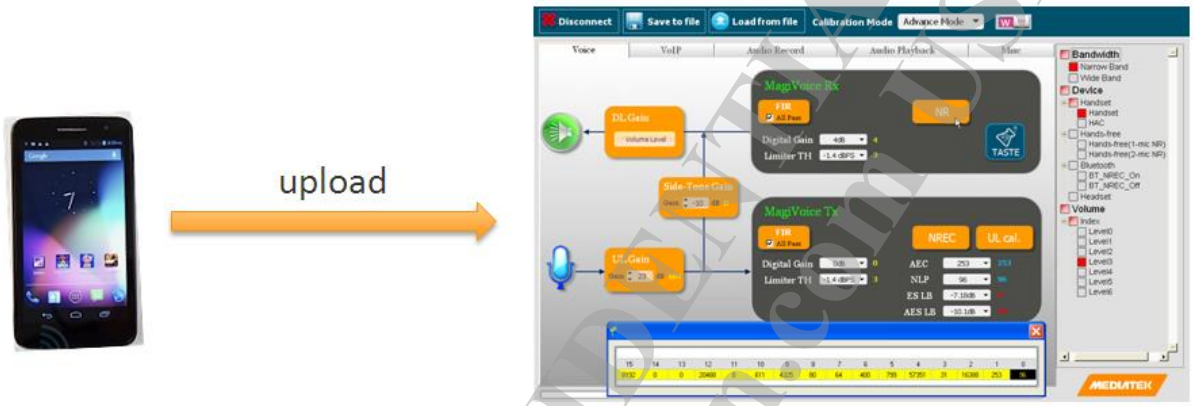
The engine will trigger audio drivers to do continuous process if need.



11.8 Parameter Update Flow

- **XML Process Flow: upload**

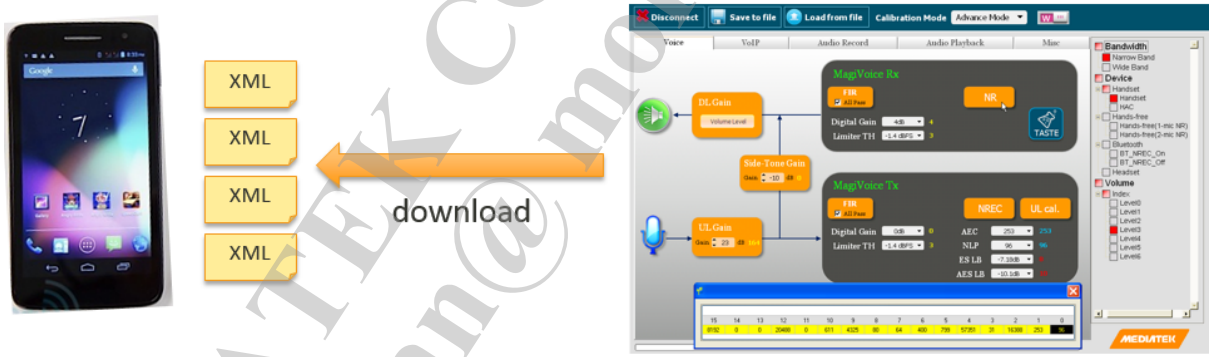
  When uploading form Device, xml files will be stored in pc



  - XML path in Device
    - (Android O) system/vendor/etc/audio_param
    - (Android O) data/vendor/audiohal/audio_param/

- **XML Process Flow: download**

  When downloading to Device, tool will send xml files to Device and save a copy in pc



  - XML path in Device
    - (Android O) data/vendor/audiohal/audio_param/

- **SW Load XML path**

  The audio XML files are stored in

  - alps\device\mediatek\common\audio_param
    - The setting is available for all projects
  - alps\device\mediatek\${Project}\audio_param
    - The setting is available for ${Project} projects
    - The applying priority is
  - ${Project} > common

  ${Project} is the suggested parameter checkin path after finishing audio tuning.

12 USB Phone Call

In this section, we will introduce MTK proprietary software architecture for USB phone call. The architecture minimizes the latency and power increased comparing to 3.5mm headset phone call. The latency measured with proprietary architecture is 22.7 ms longer than 3.5mm headset phone call. The power consumption measured with proprietary architecture is 27.6 mA large than 3.5mm headset phone call. Both measurements have the device latency and power consumption excluded. Google default AOSP path are not used, due to its long latency path, more than 400ms are added to the phone call latency path on Nexus 5, up to 300ms on HTC 10 device.

12.1.1 Architecture

12.1.1.1 Hardware Architecture

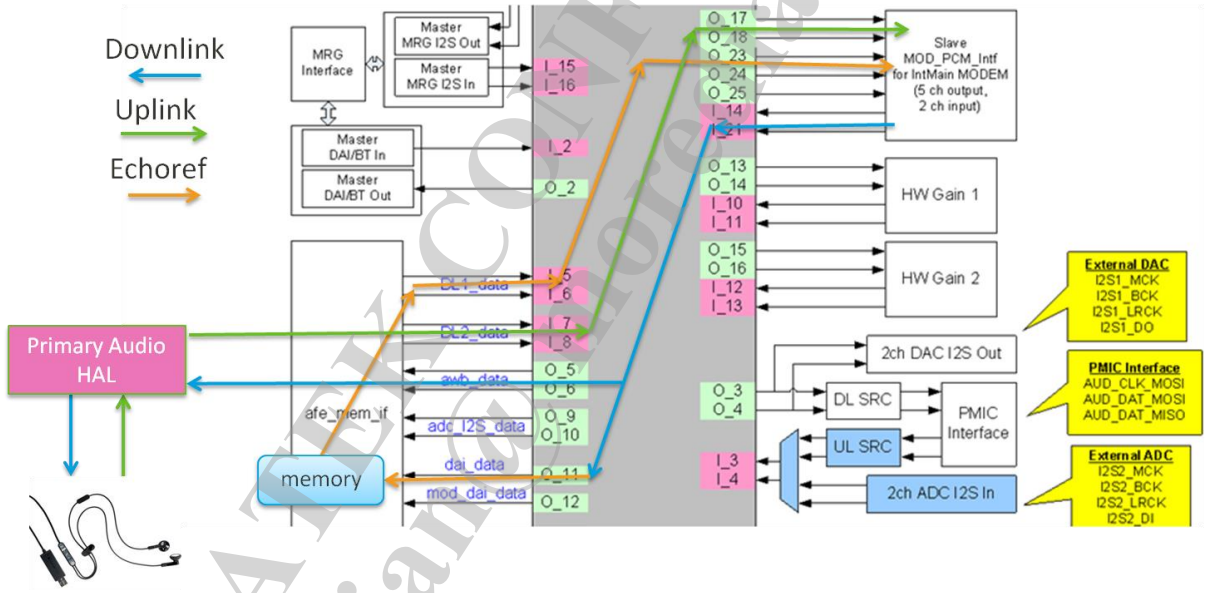


Figure 64. USB Phone Call Hardware Architecture with USB playback and capture device

The USB phone call hardware architecture is illustrated in Figure 64.

Speech Downlink path, Modem → PCMIF → UL MEMIF → Primary Audio HAL→ USB sound card → USB device

Speech Uplink path, USB Microphone→USB sound card→ Primary Audio HAL→ DL MEMIF→ PCMIF→ Modem

Speech Echo Reference path, Modem DL → PCMIF → UL MEMIF → memory→ DL MEMIF → PCMIF→ Modem

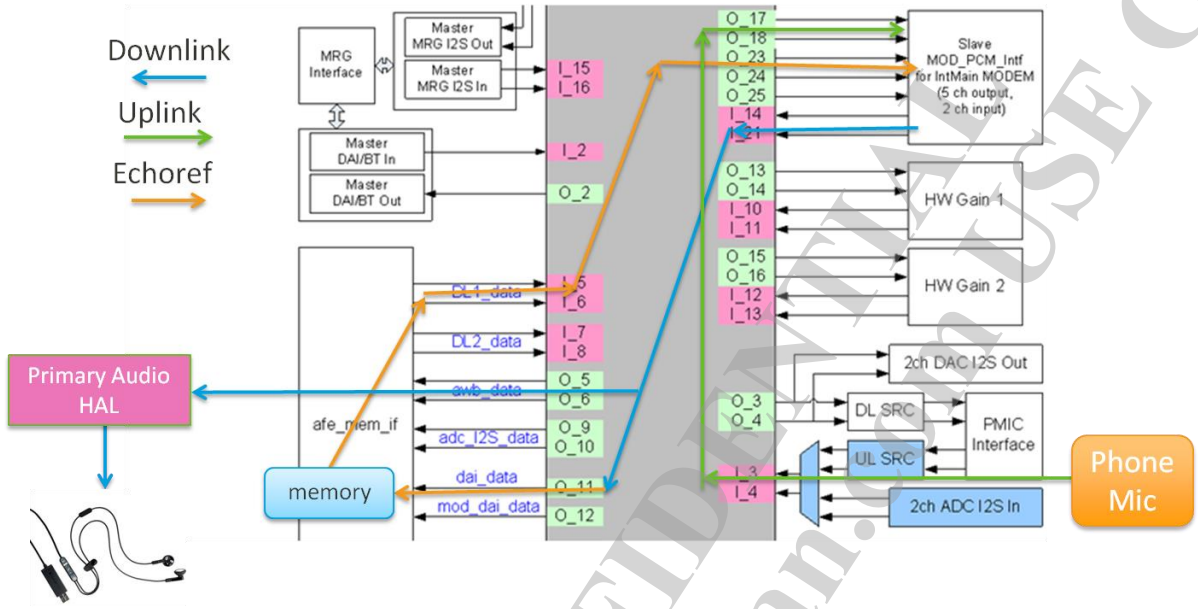


Figure 65. USB Phone Call Hardware Architecture with USB playback only

The hardware architecture of USB phone call using USB playback only device is illustrated in Figure 65. The difference is in Speech uplink path, which the source is come from phone microphone, hardware path is same as normal phone call.

12.1.1.2 Software Architecture

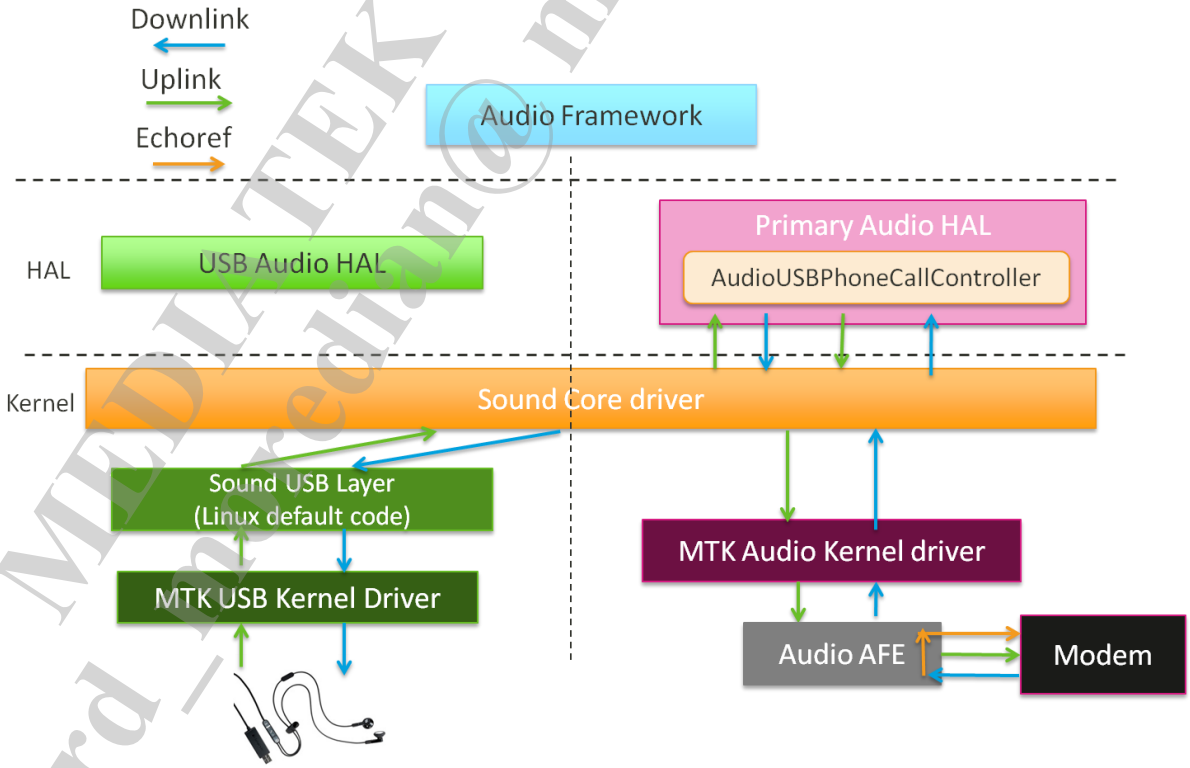


Figure 66. USB phone call software architecture

The software architecture of USB phone call is illustrated in

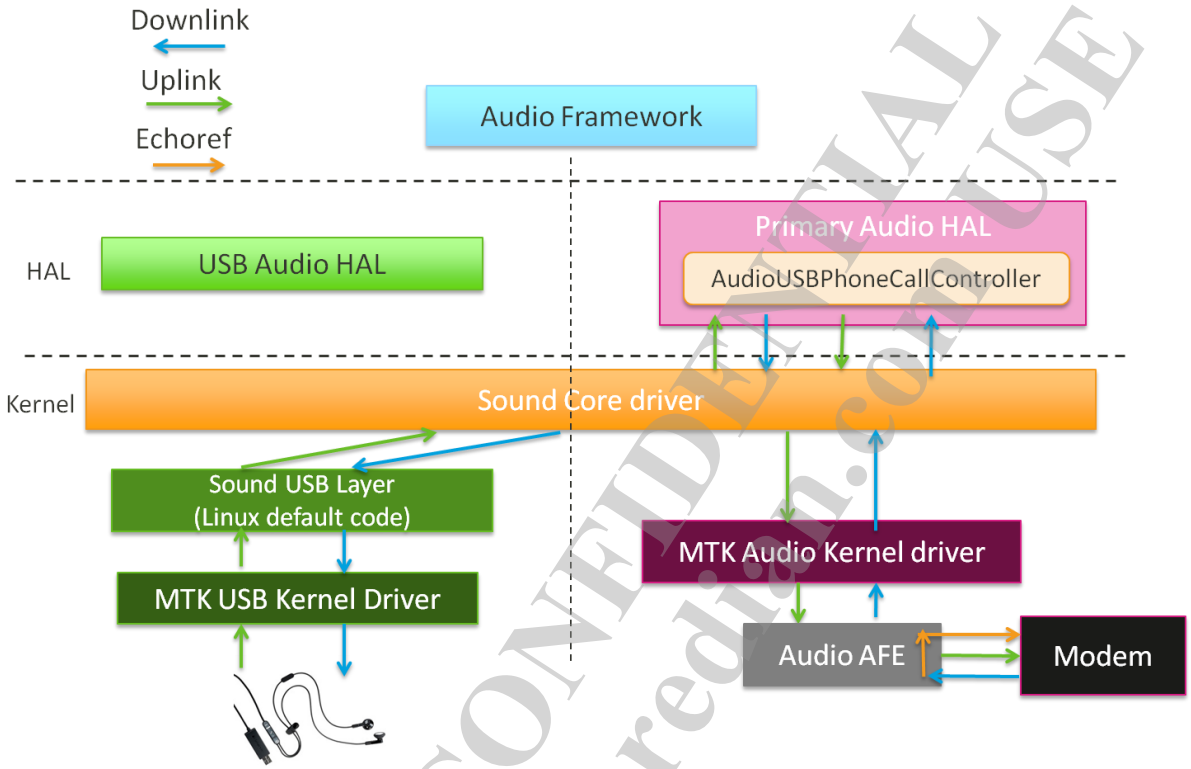


Figure 66. As shown in the figure, USB phone call is mainly composed of Primary Audio HAL, Audio Kernel and USB Sound Kernel. Unlike Google default architecture, the data does not flow through audio frameworks and USB Audio HAL. The main controller is AudioUSBPhoneCallController in Primary Audio HAL, which is responsible for the data transfer between Primary Audio Kernel Driver and USB Sound Kernel Driver. The standard interface, ALSA, is used to playback/capture data from primary kernel and USB kernel.

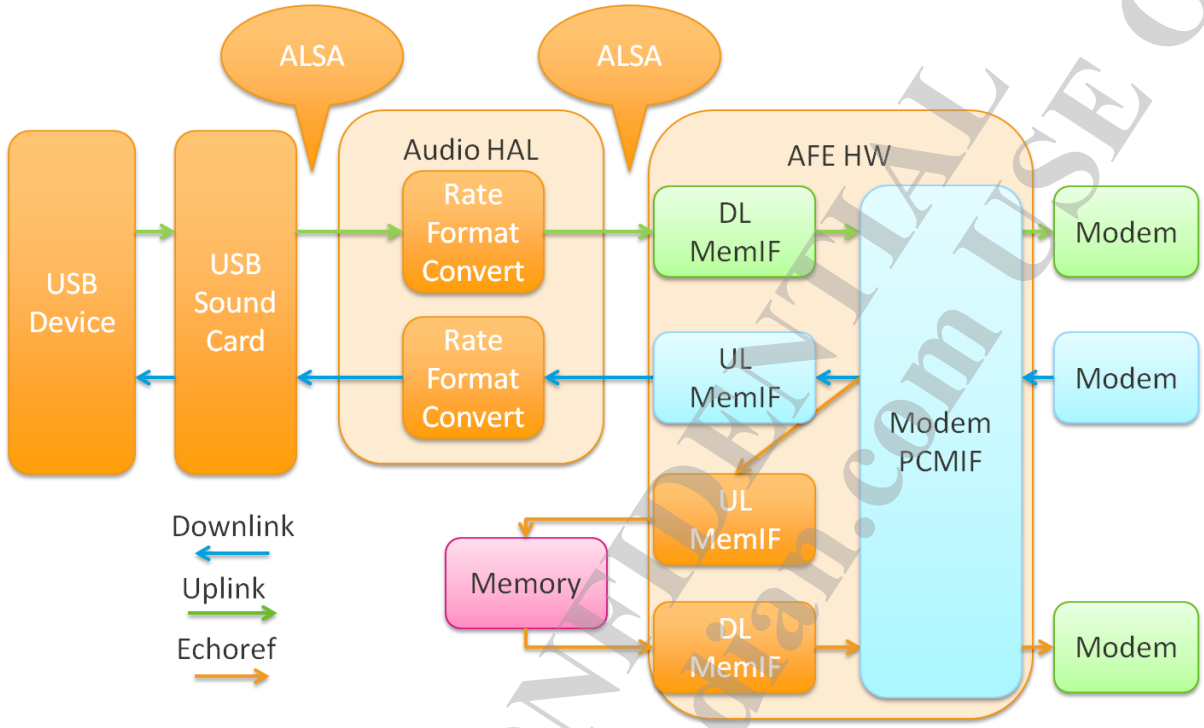


Figure 67. USB phone call software architecture

A detail view of USB phone call related modules is shown in Figure 67.

12.1.1.3 MTK_USB_PHONECALL Feature Option

MTK_USB_PHONECALL is the Feature Option for USB phone call, the value description is listed below.

Value	Description
AP	USB phone call using AP side solution
NONE	Not support USB phone call

Table 1. MTK_USB_PHONECALL Feature Option Description

12.1.2 Audio USB Phone Call Controller

12.1.2.1 Control Flow

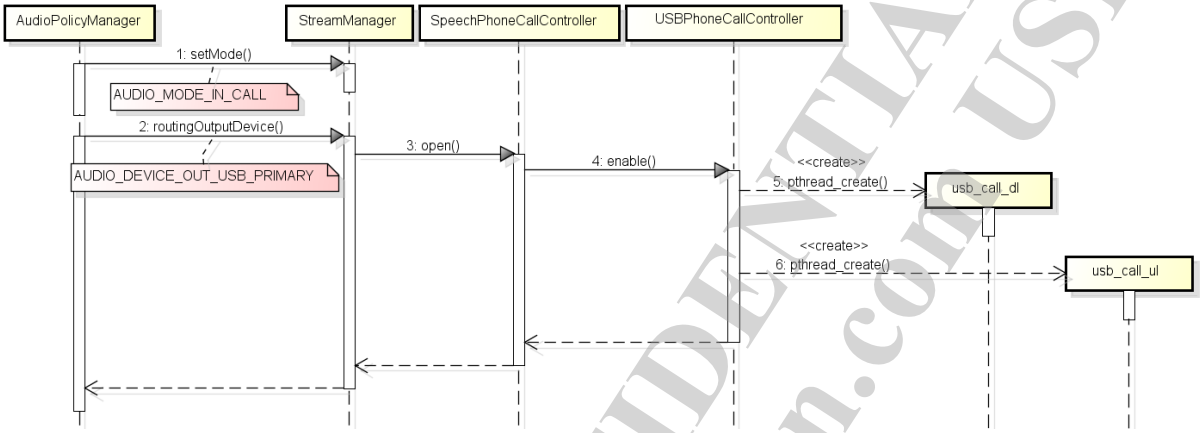


Figure 68. USB phone call enable control sequence

USB phone call enable control sequence is just like normal phone call, the audio mode and device message will be sent to AudioALSASpeechPhoneCallController. When AUDIO_DEVICE_OUT_USB_PRIMARY is used for phone call, the data control will hand over to AudioUSBPhoneCallController. Two threads are created for transferring between modem speech data and USB device, usb_call_dl and usb_call_ul.

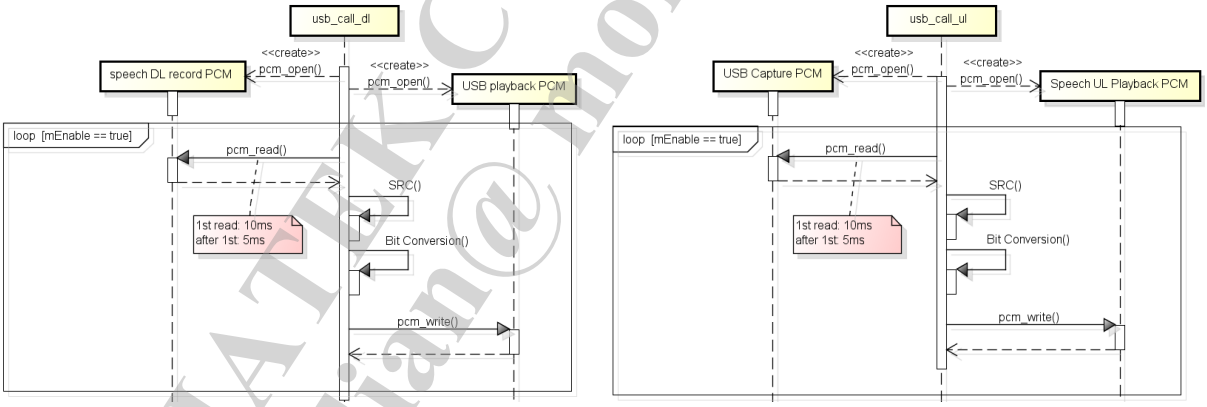


Figure 69. USB phone call usb_call_dl/usb_call_ul thread control

As shown in Figure 69, the responsibility of two threads is to transfer data between audio AFE and USB audio driver, since audio AFE and USB hardware has no hardware connection. ALSA PCM is used for write/read data to/from audio AFE and USB audio driver.

For thread usb_call_dl, first, it will read speech downlink data from audio AFE. The data will pass through sample rate conversion and bit conversion, since the rate and format may be different between two modules. The data is then written to USB audio driver, and playback to USB device.

For thread usb_call_ul, first it will read data from USB microphone through USB audio driver. The data will pass through sample rate conversion and bit conversion. The data is then written to audio AFE, and float to modem uplink.

12.1.2.2 Buffer and Latency

To meet the criteria of phone call specification, we need to minimize the latency introduced by USB phone call path. The buffer used is illustrated in Figure 70, between audio HAL and two modules driver. The buffer configuration is list below, total buffer size is 20 ms(period size * period count = 5ms * 4 = 20ms).

Period size = 5ms = interrupt period = 5ms

Period count = 4

Start threshold = 5ms

First read = 10ms, after first read = 5ms

Under this configuration, the latency is around 10ms for one speech direction. An illustration of data transferring is provided in Figure 70.

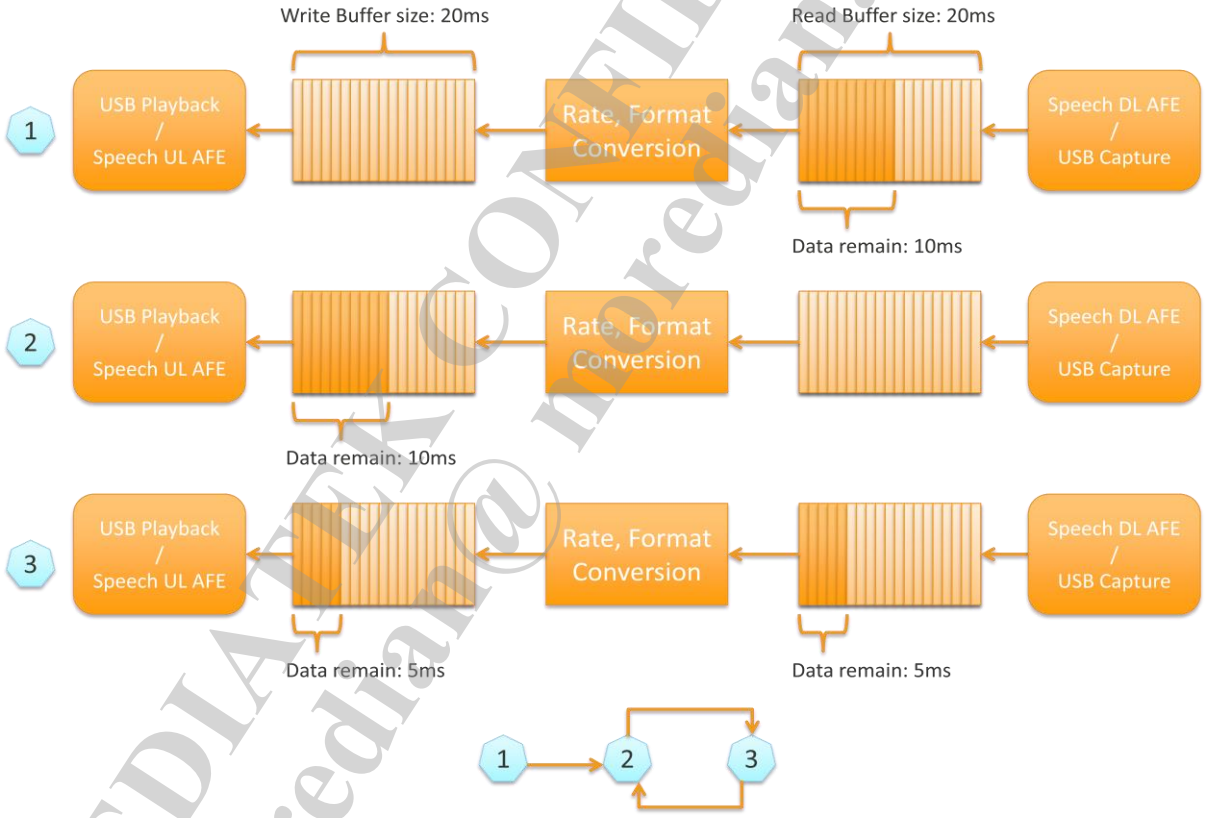


Figure 70. USB phone call buffer data transferring illustration

[1] HAL start read buffer, wait for 10ms data, HAL read 10ms data from read buffer.

[2] 10ms data is transferred to write buffer, the write buffer then start.

[3] HAL waits for 5ms in read buffer, at the meantime, the data remain in write buffer will be 5ms.

[4] 5ms data is transferred to write buffer.

Then the [3]..[4] step will keep repeating.

Since the rate between two modules is different, we need to do SRC in between. Due to the SRC process, the amount of first output data may varies. For example, 10ms input data after conversion may output 9ms. The latency for each USB call thread may not be exactly 10ms, but should be around 10ms. In addition, 3ms latency is added, 3urb is write to USB playback device, 1 urb = 1ms in USB phone call.

The theoretically latency introduced by USB phone call path can be calculated.

Total latency in USB call path:

USB phone call with USB playback + capture:

10ms (usb_call_dl) + 10ms (usb_call_ul) + 3ms (USB playback additional 3urb) + device latency

USB phone call with USB playback only:

10ms (usb_call_dl) + 3ms (USB playback additional 3urb) + device latency + audio HW uplink latency

12.1.2.3 RT Thread

As mentioned in Buffer and Latency section, USB phone call has small buffer. Therefore, during USB phone call, CPU should not be blocked more than 10ms to service USB phone call thread. The two threads are set to RT thread with Priority 1. User Debug and User load are recommended to verify USB phone call feature, since Eng load has low performance.

12.1.3 AudioPolicyManager in USB Phone Call

12.1.3.1 Modification

MTK doesn't support the AOSP USB device phone call. However we want to make/answer a phone call by USB devices. Technically speaking, there are several rule modifications here.

1. Define new devices for USB phone call, AUDIO_DEVICE_OUT_USB_PRIMARY and AUDIO_DEVICE_IN_USB_PRIMARY
   - Like BT A2DP and SCO, the SCO device is attached to Primary Module and the A2DP device is attached to A2DP module. They won't work at the same time. We also apply this for USB related devices. Make AUDIO_DEVICE_OUT_USB_PRIMARY and AUDIO_DEVICE_IN_USB_PRIMARY are attached to Primary module, and the original AUDIO_DEVICE_OUT_USB_DEVICE and AUDIO_DEVICE_IN_USB_DEVICE are attached to USB module. And make sure that USB module is suspended when making a phone call by the Primary USB device, USB module is active if there is no phone call by the Primary USB device.
   - Both of the new devices are not known by other module, like Java frameworks. There is no events for plugging in/out, audio volume control, and device routing instruction on the both of new devices from Java frameworks. We bind AUDIO_DEVICE_OUT_USB_PRIMARY to AUDIO_DEVICE_OUT_USB_DEVICE and AUDIO_DEVICE_IN_USB_PRIMARY to AUDIO_DEVICE_IN_USB_DEVICE for plugging in/out, audio volume control. It is easy to find the modification in the AudioPolicyManager codebase.

2. Routing Strategy between AUDIO_DEVICE_OUT_USB_PRIMARY and AUDIO_DEVICE_OUT_USB_DEVICE, and AUDIO_DEVICE_IN_USB_PRIMARY and AUDIO_DEVICE_IN_USB_PRIMARY

- The main principle for device routing
  - If AUDIO_DEVICE_OUT_USB_PRIMARY is connected in the system
    - If there is a phone call connected, the system will suspend the USB module and route Primary module to AUDIO_DEVICE_OUT_USB_PRIMARY. Also the system will invalidate all tracks which output to USB module, and close all inputs for correcting input devices. After invalidating tracks and closing inputs, the system will re-route to correct devices and it won't route to USB module in the phone call state.
    - If hang up the phone call, the system will restore the USB module. Also the system will invalidate all tracks which output to Primary module, and close all inputs for correcting input devices. After invalidating tracks and closing inputs, the system will re-route to correct devices and it is possible to route to USB module now.
  - If there is no AUDIO_DEVICE_OUT_USB_PRIMARY connected in the system, it is impossible to make/answer a phone call by USB devices.

12.1.3.2 Define New Input / Output Device

Because we cannot use original USB device to do routing in AudioPolicyManager, we define a new output device and a new input device for USB phone call.

- AUDIO_DEVICE_OUT_USB_PRIMARY // for downlink in the USB Phone Call
- AUDIO_DEVICE_IN_USB_PRIMARY // for uplink in the USB Phone Call

Here is the location of definition

Usage	File Path
Declare Device	system\media\audio\include\system\audio.h

Table 2. Declare Device Table.

12.1.3.3 Device Configuration Setting

Regarding the capability of modules in the AudioPolicyManager, we modify them for new devices. We add AUDIO_DEVICE_OUT_USB_PRIMARY into the Primary Out, and add AUDIO_DEVICE_IN_USB_PRIMARY into the Primary In. If adding the settings into the configurations, it must define the devices into the previous audio.h. Or the system will crash when booting.

Here is the location of definition

Android Version	File Path
Android M	device\mediatek\[project name]\audio_policy.conf

Android Version	File Path
Android N	**device\mediatek\common\audio_policy_config\ audio_policy_configuration.xml** device\mediatek\[chip platform name]\audio_policy_config\ audio_policy_configuration.xml device\mediatek\[project name]\audio_policy_config\ audio_policy_configuration.xml

Table 3. Audio Policy Configuration.

12.1.3.4 Logic modification in AudioPolicyManager module

Regarding device volume and device routing of USB phone call, all of main logic modifications are applied into AudioPolicyManager module, and the modifications are identified by a build option, MTK_USB_PHONECALL. We could find all modification by searching the option.

Here is the location of definition

Android Version	Folder/File Path
Android M	**frameworks\av\services\audiopolicy**
Audroid N	**frameworks\av\services\audiopolicy** frameworks\av\services\audioflinger (ToString for AUDIO_DEVICE_OUT_USB_PRIMARY/ AUDIO_DEVICE_IN_USB_PRIMARY)

Table 4. Logic modification in AudioPolicyManager.

12.1.4 Active Echo Cancellation

For USB phone call, the Active Echo Cancellation (AEC) is different from normal phone call, mainly because the latency is long (>20ms) and may varies (different USB device). The AEC requires the alignment between speech uplink data (mic data) and echo reference data. For internal AEC algorithm to work well it is recommended that echo reference data is 8ms ahead of mic data. The speech downlink data is the source for echo reference data. To achieve the alignment, the speech downlink data must be delayed before sending back to modem as echo reference data. The delay needed can be calculated below:

USB phone call with USB playback + capture:

[USB phone call with USB playback + capture latency] – 8ms

USB phone call with USB playback only:

[USB phone call with USB playback only latency] – 8ms

12.1.4.1 Delay using MEMIF

As shown in Figure 64, the speech downlink data flow through UL MEMIF and DL MEMIF then back to PCMIF echo reference port. The two MEMIF are used to delay the data, it's a pure hardware delay technique.

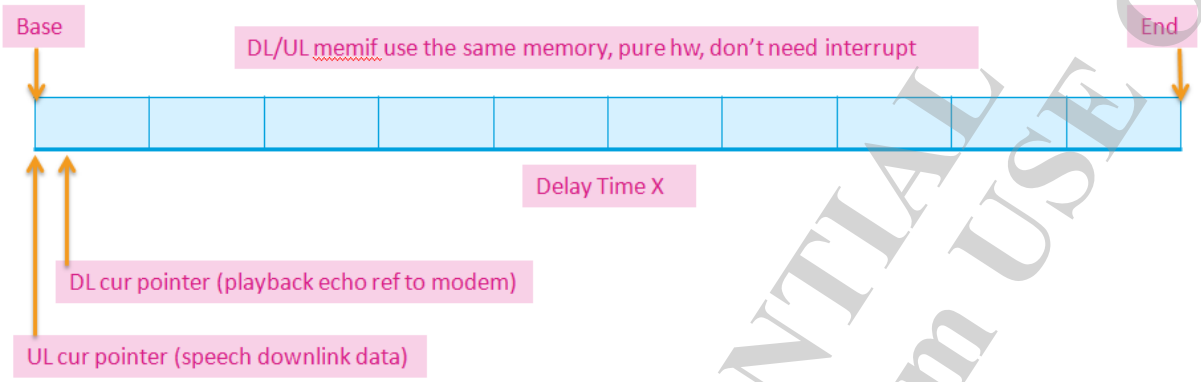


Figure 71. USB phone call, Delay using MEMIF

DL and UL MEMIF share the same memory space. The size of the memory is equal to the delay time needed. The DL MEMIF will be enabled first, and UL MEMIF is enabled after 30us. DL MEMIF will read out the data in the memory, which is mute. At the same time, UL MEMIF will write speech downlink data into the memory. After "Delay Time X", DL MEMIF will reach the speech data written "Delay Time X" time beforehand. And thus, the combine of two MEMIF can delay the data "Delay Time X".

Control Flow,
1. Setup dl/ulmemif with same base/end address
2. the size is equal the delay time
3. Enable dl memif
4. Delay 30us
5. Enable ulmemif
6. HW free run

→ Data from the memif will be delayed for Delay Time X

Figure 72. USB phone call, Delay using MEMIF, control flow

12.1.4.2 Debug and Tuning Method

The alignment of echo reference data and mic data can be checked using below method.

Step:

1. Enter command
   - adb root
   - adb shell setprop usbsph.debug 4 (USB_DBG_ECHO_REF_ALIGN = 0x1 << 2)
2. Insert USB device & 3.5mm line (headphone ←→PC)
3. Locate USB output and input as close as possible.
4. PC start recording from 3.5mm
5. Start phone call
   - L channel will be speech uplink data
   - R channel will be echo ref data

The hardware path of debug and tuning method can be seen in Figure 73.

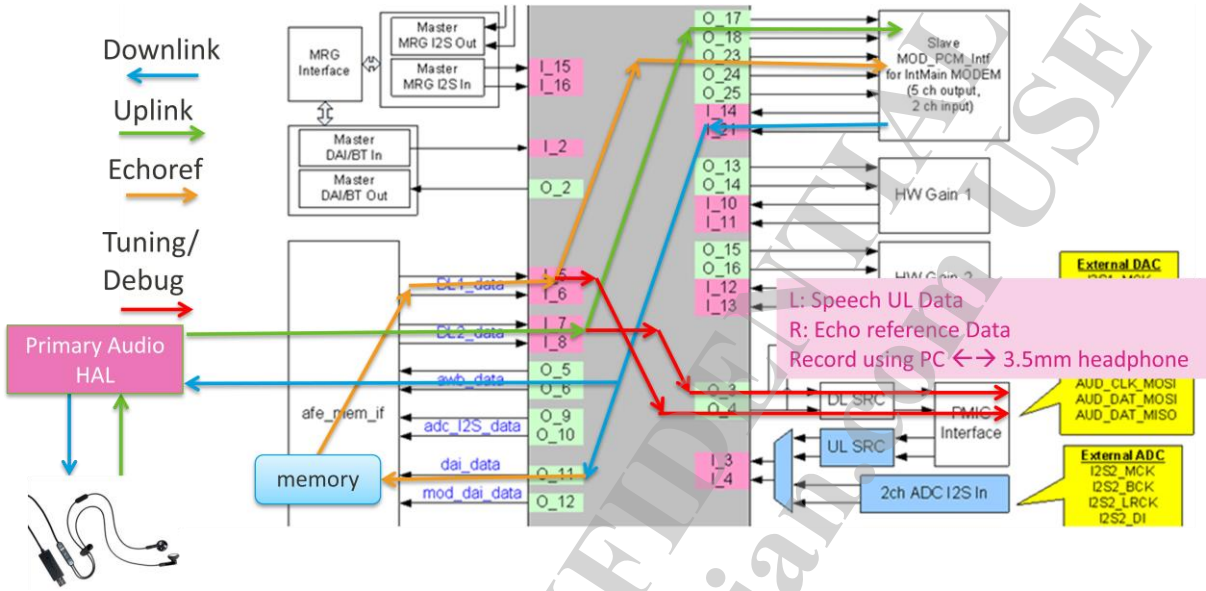


Figure 73. USB Phone Call, Echo Reference and Mic Data Alignment Tuning

12.1.5 Data Throttle Control

A variety of factor may affect the latency of USB phone call, which is not acceptable. One of the factors is the synchronization modes of the USB device. There are three possible types of synchronization modes, synchronous, adaptive and asynchronous. The synchronous mode guarantees the data rate of playback or capture. While asynchronous and adaptive mode may have small deviation of the data rate, which means the sample rate of USB device may be different from the host sample rate. These kind of synchronization mode will lead to the change of data remained in playback buffer size, so as the change in latency. The data throttle control mechanism is designed to handle this kind of situation.

12.1.5.1 Concept

The mechanism uses the Sample Rate Conversion (SRC) to change the data rate written to playback buffer. For example, when data remained in playback buffer increased, we increase SRC input rate, so that the data rate after SRC decreased. When data remained in playback buffer decreased, we decrease SRC input rate, so that the data rate after SRC increased. By changing the data rate generated by SRC we can compensate the change of the data remained in playback buffer.

12.1.5.2 State Machine

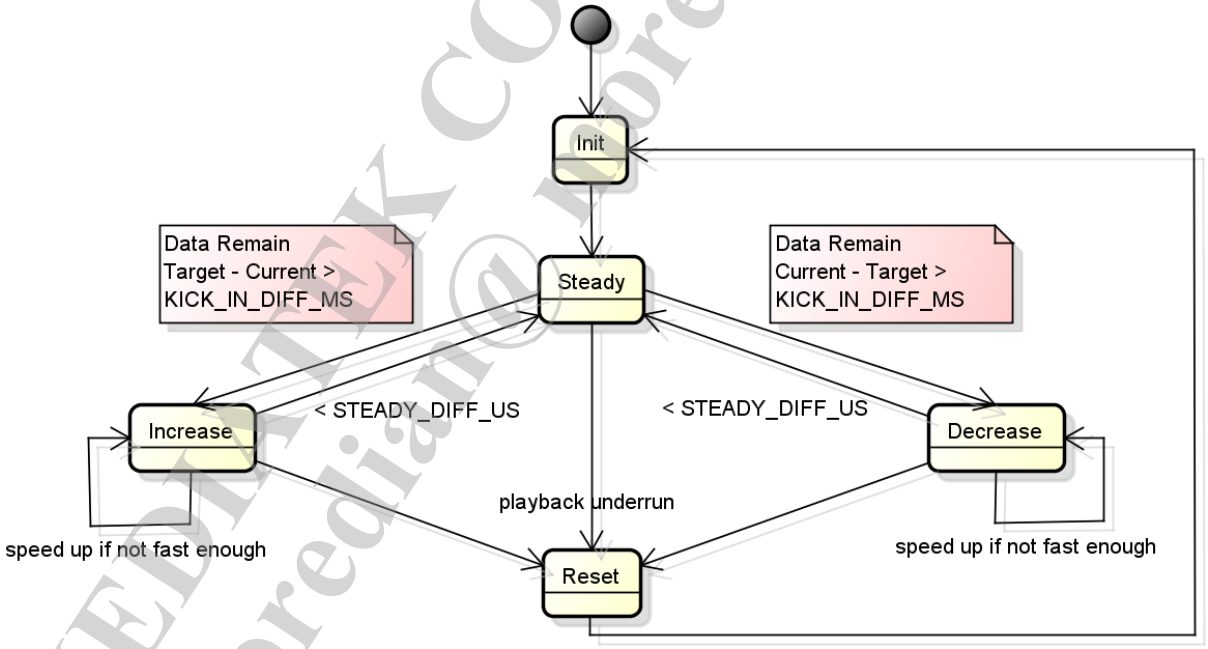


Figure 74. USB Phone Call, Throttle Control, State Machine Diagram

The throttle control mechanism will be executed in both usb_call_dl and usb_call_ul thread. The data remained in the playback buffer is the index of throttle control, which is checked after every write to playback buffer. The state machine diagram is showed in Figure 74. The data remained in the playback buffer determine the state of throttle control mechanism, each state condition will be described in following paragraph.

Init State:

Enter Init State when first write to playback buffer. In this state, we will get the data remained in the playback buffer, and the value will be the Target for throttle control. After getting the Target value, it will enter Steady State.

Steady State:

In Steady State, it will monitor the Current data remained in the playback buffer. And check the difference between Current and Target data remained value. State will change to Increase or Decrease state if the difference is too big.

Steady State → Increase State, if Current < Target - KICK_IN_DIFF_MS.

Steady State → Decrease State, if Current > Target + KICK_IN_DIFF_MS.

Increase State:

In Increase State, it will increase the output data rate of SRC by decreasing SRC input rate. It will further increase the output data rate of SRC if compensation is not fast enough. It will go back to Steady State if the difference between Current and Target is within STEADY_DIFF_US.

Decrease State:

In Decrease State, it will decrease the output data rate of SRC by increasing SRC input rate. It will further decrease the output data rate of SRC if compensation is not fast enough. It will go back to Steady State if the difference between Current and Target is within STEADY_DIFF_US.

Reset State:

Enter Reset State if underflow happened in playback buffer. The SRC output data rate will be reset to normal rate. Then it will go to Init State.

12.1.5.3 Parameter Description

Parameter	Description
KICK_IN_DIFF_MS	Throttle control kick in if difference between Current and Target exceed this value.
KICK_IN_COUNT	Check KICK_IN_CONUT times before throttle control kick in.
STEADY_DIFF_US	Back to Steady State if difference between Current and Target is within this value.
MS_CHANGE_PER_SECOND	Compensation speed, increase or decrease this value per second.
SPEED_UP_COUNT	Check SPEED_UP_COUNT times before speed up.

12.1.6 USB Phone Call Parameter

USB Phone Call Parameters are stored in XML

XML in code base,

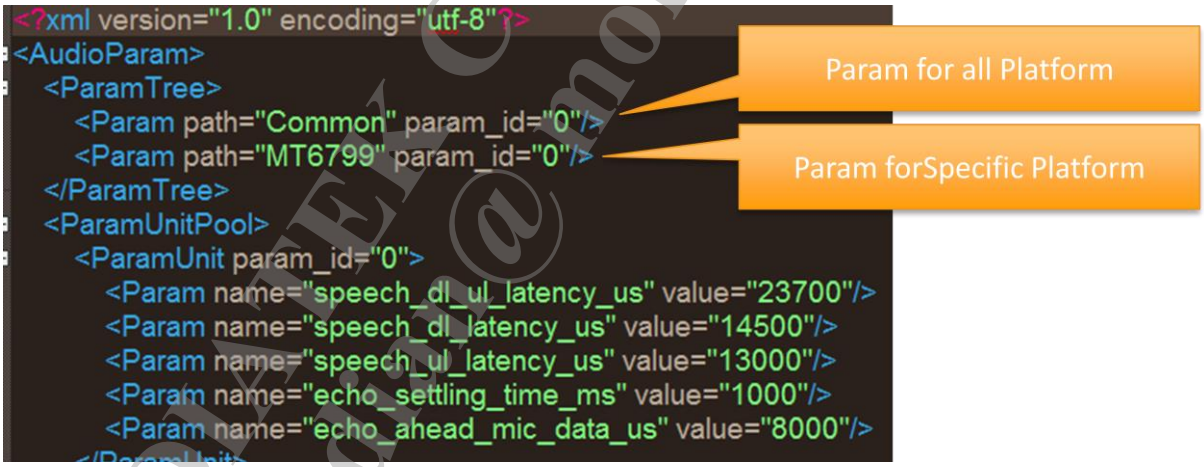
- Common: device\mediatek\common\audio_param\
- Platform: device\mediatek\$(PLATFORM)\audio_param (if needed)
  - Ex. device\mediatek\MT6758\audio_param
- Project: device\mediatek\$(PROJECT)\audio_param (if needed)
  - Ex. device\mediatek\k99v1_64_bsp\audio_param
- Project XML > Platform XML > Common XML
  - Project XML will override Platform XML, while Platform XML will override Common XML.

XML on device,

- Default xml: system/vendor/etc/audio_param/
- Customized xml (created by tuning tool): sdcard/.audio_param/

12.1.6.1 USB Call Parameter

File Name	Description
USBCall_ParamUnitDesc.xml	USB Phone Call driver related, parameter description
USBCall_AudioParam.xml	USB Phone Call driver related, parameter value

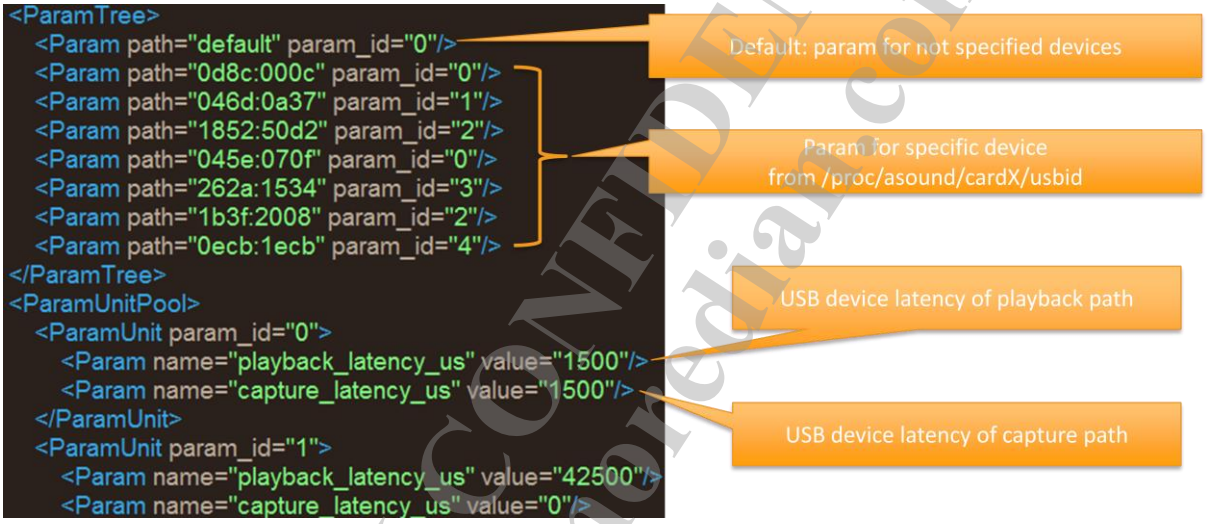


Parameter Name	Description
speech_dl_ul_latency_us	total sw path delay, when usb call with usb playback/capture device
speech_dl_latency_us	total sw path delay, when usb call with usb playback & phone microphone
speech_ul_latency_us	Not used
echo_settling_time_ms	for AEC, wait 1sec then start sending echo reference data to AEC module
echo_ahead_mic_data_us	For AEC, echo reference is desired to be 8ms ahead of mic data

Do not tune USBCall_AudioParam.xml if you didn't change driver code which will change software latency. Such as the value of USB_SPH_LATENCY_MS and USB_SPH_DL_2_HAL_PERIOD_CNT.

12.1.6.2 USB Device Parameter

File Name	Description
USBDevice_ParamUnitDesc.xml	USB device related parameter description
USBDevice_AudioParam.xml	USB device related parameter value



Parameter Name	Description
usbid (ex. 0d8c:000c)	The USB device id. Should match the string in /proc/asound/cardX/usbid
playback_latency_us	The USB device latency of playback path
capture_latency_us	The USB device latency of capture path

Add your device parameters if it's not already in USBDevice_ParamUnitDesc.xml, or the AEC may not work. To get the device delay information, consult the device vendor for the information, or use debug and tuning method described in 12.1.4.2Debug and Tuning Method.

12.1.6.3 USB Call Gain Control

A GAIN_DEVICE_USB is added for independent tuning for USB device Phone Call. For detail information please check section **Error! Reference source not found. Error! Reference source not found.**.

12.1.7 Debug Method

Debug command are list in below table, it can be enabled by setting property. Be noted that the command are bitwise, so you can enable multiple debug command at the same time.

Command:

adb root

adb setprop usbsph.debug “USB_DBG_TYPE”

```
enum USB_DBG_TYPE {
    USB_DBG_ASSERT_AT_STOP = 0x1 << 0,
    USB_DBG_BUFFER_LEVEL = 0x1 << 1,
    USB_DBG_ECHO_REF_ALIGN = 0x1 << 2,
    USB_DBG_USE_DL_ONLY = 0x1 << 3,
    USB_DBG_USB_UL_ADDITIONAL_DATA_TEST = 0x1 << 4,
    USB_DBG_ECHO_USE_SW = 0x1 << 5,
    USB_DBG_DL_TIME_PROFILE = 0x1 << 6,
    USB_DBG_UL_TIME_PROFILE = 0x1 << 7,
    USB_DBG_DL_DISABLE_THROTTLE = 0x1 << 8,
    USB_DBG_UL_DISABLE_THROTTLE = 0x1 << 9,
    USB_DBG_ALL = 0xFFFFFFFF,
};
```

An brief description of each command is provided below, please trace the driver code for detail usage.

Parameter Name	Description
USB_DBG_ASSERT_AT_STOP	Trigger assert when there is underflow during USB phone call.
USB_DBG_BUFFER_LEVEL	Print log about the data remained in the buffer.
USB_DBG_ECHO_REF_ALIGN	As described in 12.1.4.2 Debug and Tuning Method.
USB_DBG_USE_DL_ONLY	Use USB playback only when USB phone call, even the USB capture device is connected.
UL_ADDITIONAL_DATA_TEST	Check USB capture device data rate.
USB_DBG_ECHO_USE_SW	Echo reference path use software solution
USB_DBG_DL_TIME_PROFILE	Print log about time consumption in usb_call_dl thread.
USB_DBG_UL_TIME_PROFILE	Print log about time consumption in usb_call_ul thread.
USB_DBG_DL_DISABLE_THROTTE	Disable throttle control in usb_call_dl thread.
USB_DBG_UL_DISABLE_THROTTE	Disable throttle control in usb_call_ul thread.

13 AP Speech Driver Modifications

13.1 Modem Hardware Change

13.1.1 Only One Modem Left

In the previous chips, we could only establish the C2K phone call via modem 3. That is, AP side speech driver have to communicate with two different modems respectively by the network status. However, in the current new modem hardware architecture, we only use one modem to achieve different kinds of phone call.

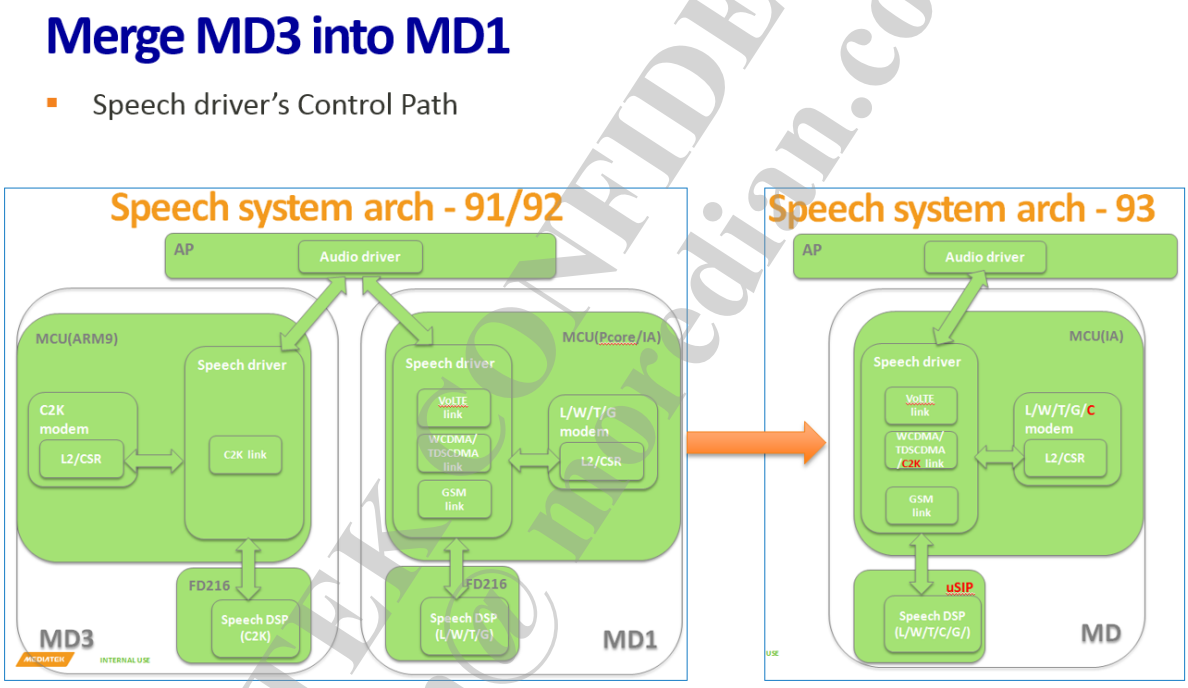


Figure 75. Merge MD3 to MD1

13.1.2 The Size of EMI Increased

The size of EMI (external memory interface) has increased from 20 Kbytes to 52 Kbytes, which means the AP and modem speech driver could use it to exchange more data and information, like speech parameters, record data, BGS data, VM/EPL data, modem alive flag, and so on.

13.2 AP Side Speech Driver Refactory

13.2.1 New Speech Driver and Speech Messenger Classes

For the modem hardware changes, AP side speech driver implement new classes:

- SpeechDriverNormal
  - Inherits from SpeechDriverInterface so that AudioALSASpeechPhoneCallController could still keep the same way to use speech driver.
  - Transfer each command like speech on/off, record on/off, gain, …, into speech message
- SpeechMessageQueue
  - There's a queue to manage the speech messages
  - When the head message is pop up to send to modem, the only thing that the other pushed messages in queue could do is just waiting until modem send head message's ack back.
- SpeechMessengerNormal
  - Transfer speech message into CCCI message format and write/read the CCCI messages to/from modem.
  - Implement the EMI read/write functions

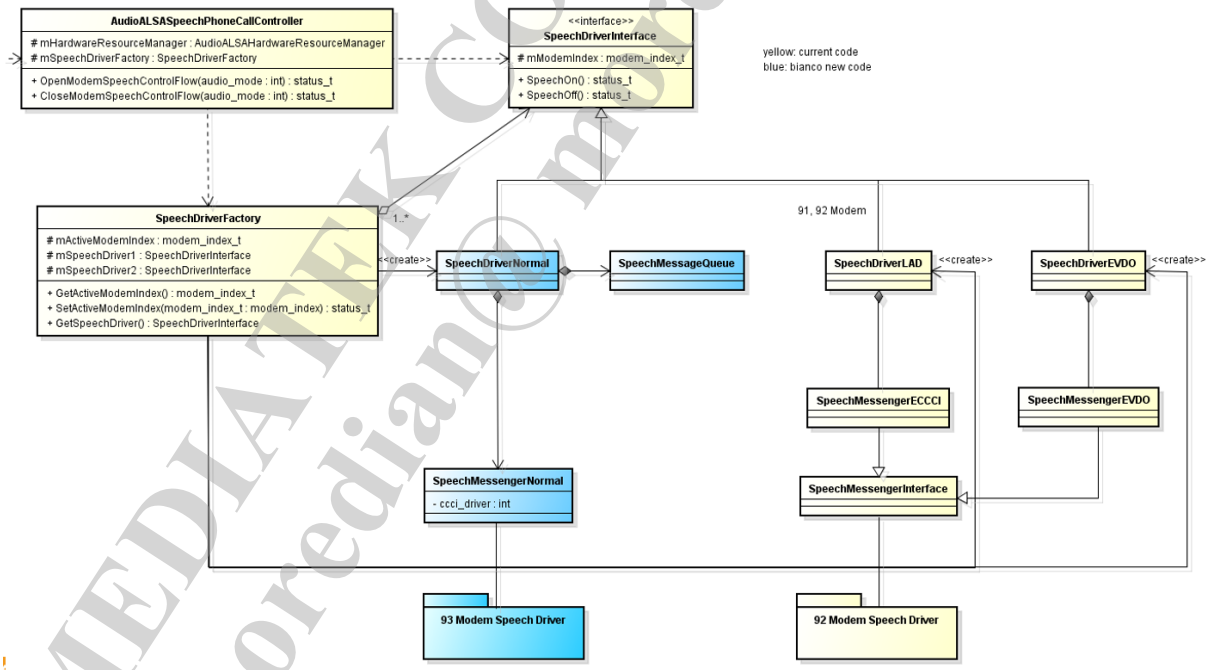


Figure 76. AP Side Speech Driver Class Diagram

13.2.2 File Path

- vendor/mediatek/proprietary/hardware/audio/common/speech_driver/SpeechDriverNormal.cpp
- vendor/mediatek/proprietary/hardware/audio/common/speech_driver/SpeechMessageQueue.cpp
- vendor/mediatek/proprietary/hardware/audio/common/speech_driver/SpeechMessengerNormal.cpp

13.3 More Dump Data Commands

- BGS
  - Before SRC
    - adb shell setprop persist.af.bgs_blisrc_dump_on 1
    - /sdcard/mtklog/audio_dump/BGS_before_Blisrc
  - Before send to modem
    - adb shell setprop persist.af.bgs_dump_on 1
    - /sdcard/mtklog/audio_dump/BGS
- PCM 2 Way
  - adb shell setprop persist.af.p2w_dump_on 1
  - /sdcard/mtklog/audio_dump/Play2Way
  - /sdcard/mtklog/audio_dump/Record2Way

14 Appendix

14.1 MTK MOL

MTK provides a forum named MTK on-line to share FAQ,eCourse and the important announcement

http://online.mediatek.com

You can type the keyword to search the related FAQ and eCourse

You can explores the document tree to enter the related audio FAQ and eCourse