MediaTek

L1Audio Design & Interface

MAUI Project

Documents Number:

Preliminary Information

Revision: 2.0.0

Release Date: Mar 20, 2007

Revision History

Revision	Date	Author	Comments
1.00	2005/9/21	Phil	First release
1.01	2005/09/27	JF Yeh	Add AAC and AMR/AWB design & interface
1.02	2005/10/7	Phil	Modify Bluetooth Interface
1.03	2005/10/25	KH Hung	Add Audio Post Process design & interface
1.04	2005/11/07	Murphy Chen	Add a new function and data structure description for AFE interface.
1.05	2005/12/03	Murphy Chen	Add AFE description for afe2.c and analog die VCALI setting.
1.06	2005/12/20	Adam Tseng	Add SBC encoder interface
1.07	2006/01/11	Adam Tseng	Add PCM4WAY/PCM2WAY interface
1.08	2006/01/25	KH Hung	Add Streaming PCM Playback & Record interface
1.09	2006/02/22	JF Yeh	Update Audio Played back design & interface
1.10	2006/04/04	Phil Hsieh	Remove SBC part and make it a standalone document.
1.11	2006/07/20	KH Hung	Add AudioPP Manager design & interface
1.12	2006/09/20	KH Hung	Update AudioPP Manager and the interface of Time Stretch and Reverb
1.13	2006/10/12	Guyger Fan	Add AGC Configurations
1.14	2006/11/06	YS Lin	Revise SND interface
1.15	2006/11/10	KH Hung	Update AudioPP Manager and Add concurrence table
1.16	2006/12/06	YS Lin	Update L1SP interface
1.17	2006/12/28	KH Hung	Revise Audio Post-Processing Manager
1.18	2007/01/02	KH Hung	Extract Audio Post-Processing Manager from L1Audio Desgn & Interface Document. APM is described in “Audio Post-Processing Interface”.
1.19	2007/02/15	JF Yeh	Revise and reorganize media playback and record interface
1.20	2007/03/16	Phil Hsieh	Replace “played back” with “playback”
1.21	2007/03/19	JF Yeh	Remove “Media Playback and Record Interface” and “Poc Audio Interface”
2.0.0	2007/03/20	Phil Hsieh	Reorganize document structure

Table of Content

1 Introduction

1.1 Concept

L1Audio software module is the audio driver for speech control, media file playback, sound recording, and volume control. L1Audio software module includes L1Audio task, L1audio HISR, D2C LISR part, AFE Manager, and AM(Audio Manager). Please refer to the diagram below.

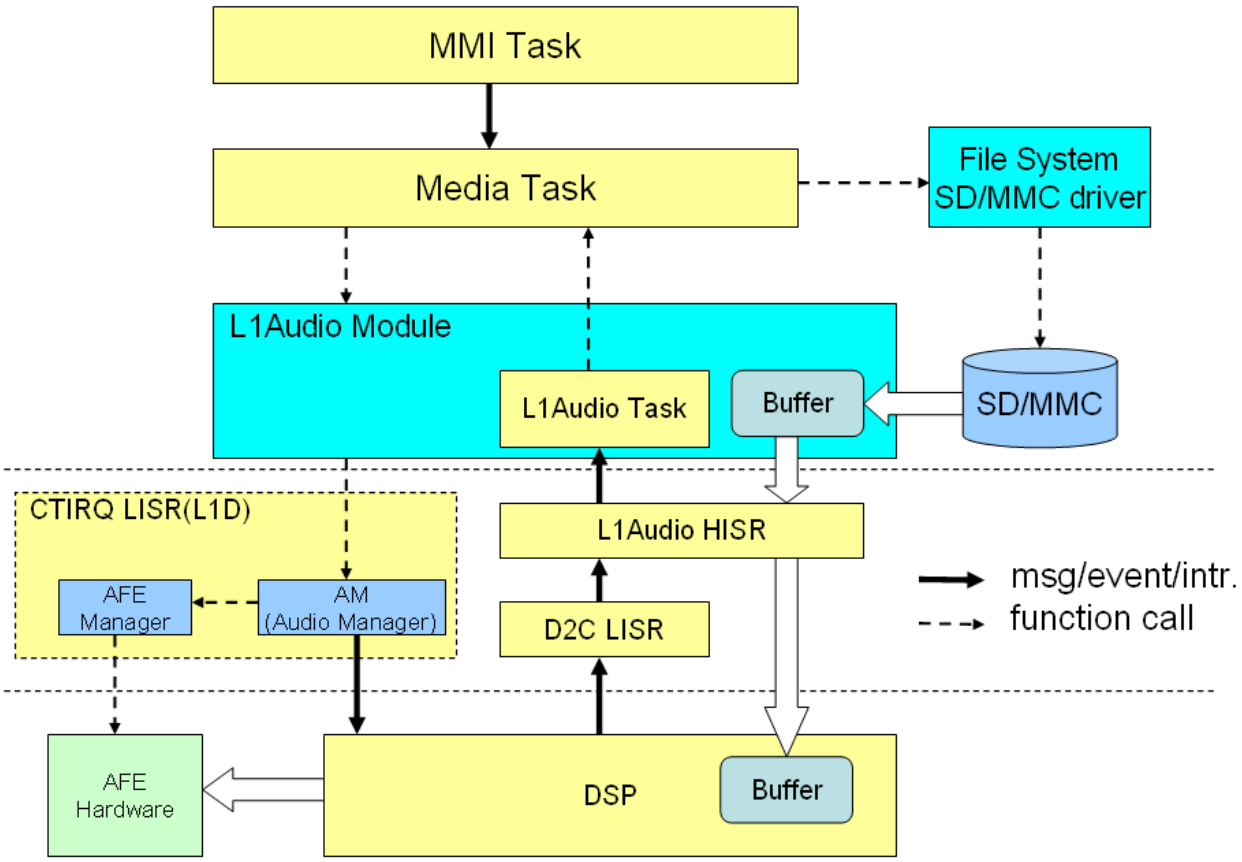


Figure 1 L1Audio Module

This document describs the implementation of L1Audio module and the interface between Media task and L1Audio. The communication between these two modules is by function calls. Media task calls the functions provided by L1Audio module to invoke certain service, such as “Turn on speech” or “Start playing a MIDI file”.
L1 Audio module provides versatile audio functions. They can be used to turning on/off speech, melody, tone, keytone, audio file(including MP3, PCM, GSM FR/EFR/HR) playback… etc. Refer to the diagram below.

In this architecture, the DSP and hardware AFE control commands is done in CTIRQ LISR context. CTIRQ LISR is mainly used by L1. It is because when turning on/off certain audio function, some parameters passing to the DSP SHERIF need to be synchronized with the audio block (20ms interval), and only L1 keeps the exact frame number of current time. So L1 can pass the parameters correctly for L1Audio module. After an audio function is turned on, the data interchange between L1Audio and the DSP is by direct access to the DSP SHERIF. And each of the audio functions has its own hand-shaking approach with a corresponding DSP task.

1.2 Header File

Before using any functions, constants, or macros of L1Audio module, a header file has to be included. Insert the following line before using any L1Audio functions.

#include “l1audio.h”

2 AFE Manager

2.1 Concept

The AFE Manager is used to control the hardware AFE part. All the interface functions are supposed to be used only by AM(audio manager), which is designed for managing both DSP and all the hardware components. Refer to the block diagram below. So do not used AFE functions in MMI or Media task, otherwise it will not work properly.

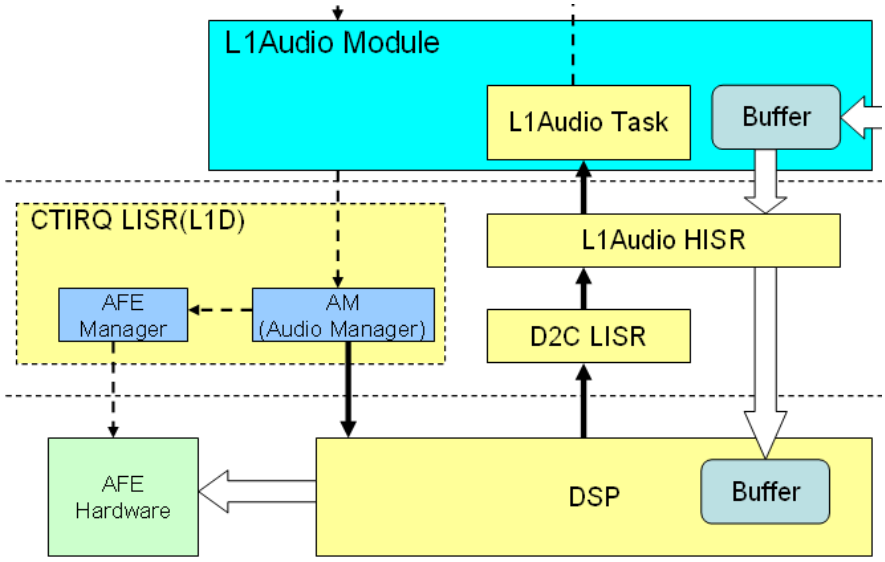


Figure 2 The Role of AFE Manager

The source code for hardware AFE control can be referred at

mcu\customer\audio\project_name\afe.c	Customization part (for MT6205B, this file is the main control part)
mcu\l1audio\afe2.c	Main control part (for MT6205B, this file doesn't exist)

2.2 Audio Front End Configuration

2.2.1 MT6205B Audio Front End Configuration

Two of possible audio hardware configurations for MT6205B are shown in the following diagrams.

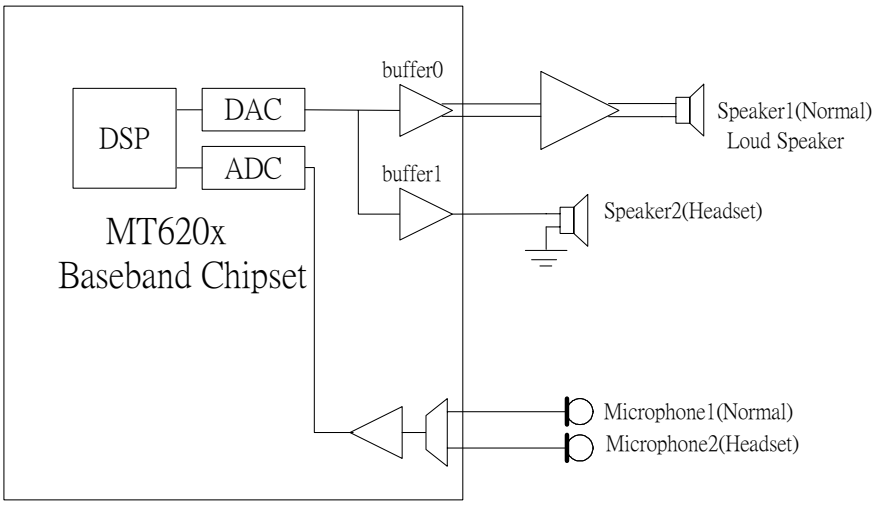


Figure 3 Audio Front End Configuration 1

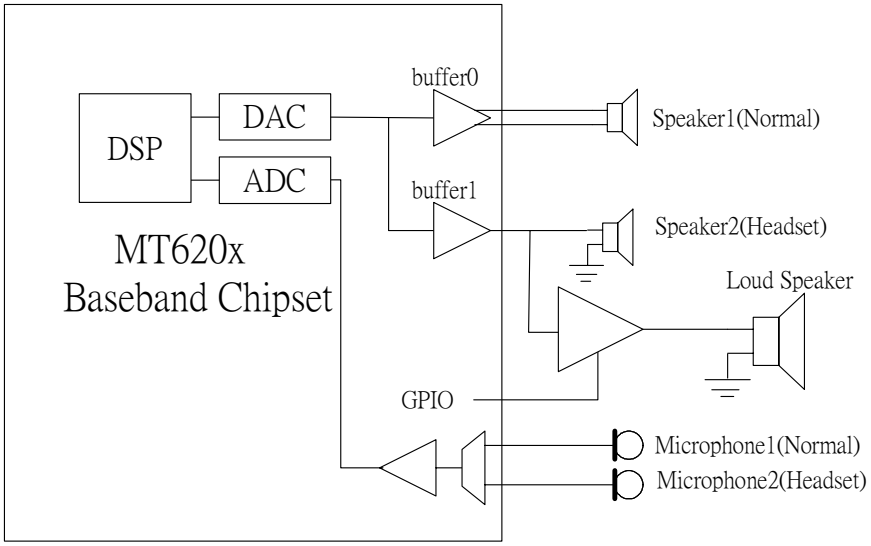


Figure 4 Audio Front End Configuration 2

In this schematic, there are two amplifiers that can be configured with different gains for different speakers. These two amplifiers can be controlled individually. At the input side, two input sources can be selected by a multiplexer, and there is only one input amplifier for these two input sources.

In the second configuration, a loud speaker is introduced for playing loud ring-tones or melody. This loud speaker needs an external audio amplifier to amplify the signal and supply enough power. The external audio amplifier is controlled by a general-purpose-output signal and the gain is fixed.
The AFE manager code for MT6205B AFE control can be referred at mcu\customer\audio\project_name\afe.c

2.2.2 MT6217/18/19 Audio Front End Configuration

Compared with MT6205B, the AFE of MT6217/18/19 adds stereo amplifiers, and FM radio input path.
It is more powerful for multimedia application.

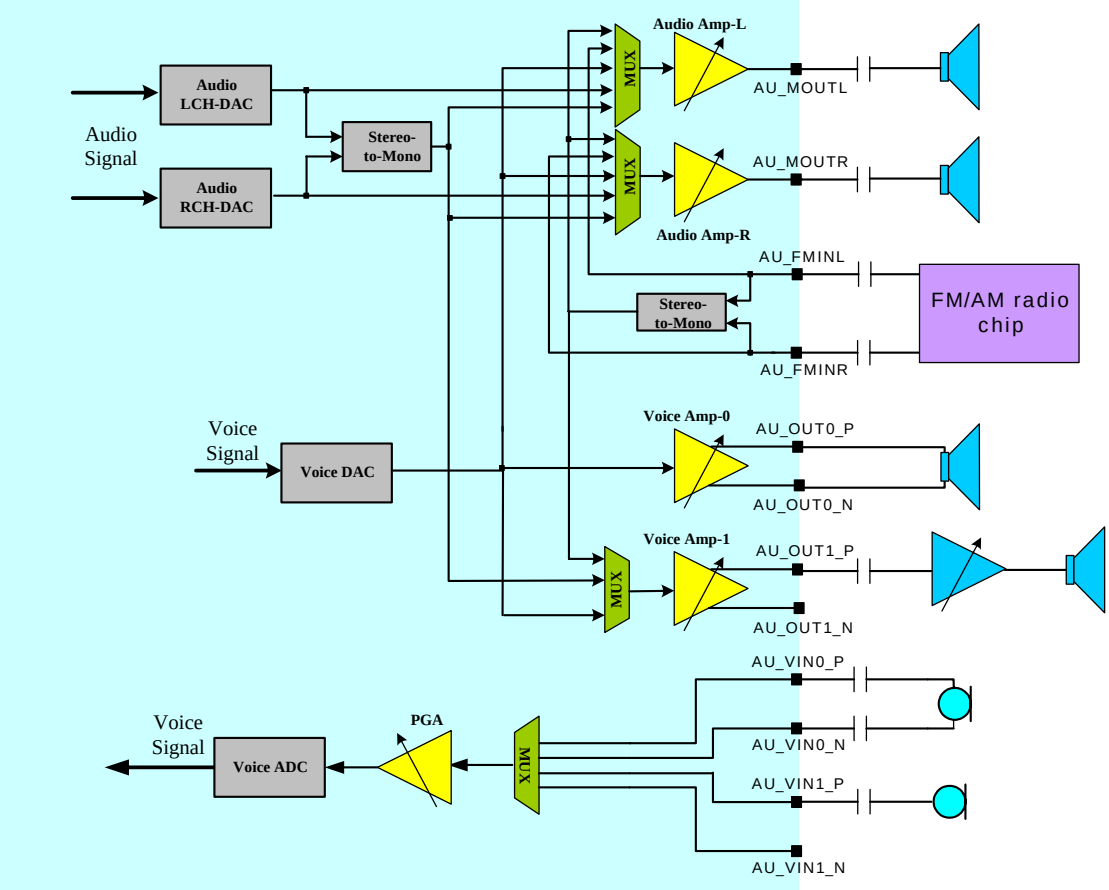


Figure 5 Audio Front End Configuration of MT6217/18/19

2.2.3 MT6226/27 Audio Front End Configuration

Compared with the AFE of MT6217/18/19, the one of MT6226/27 removes Voice Amp-1, which is usually redundant in real application. The mono loudspeaker can always be connected to one of the L or R channel of the stereo output and we can configure to output L+R signals to it.

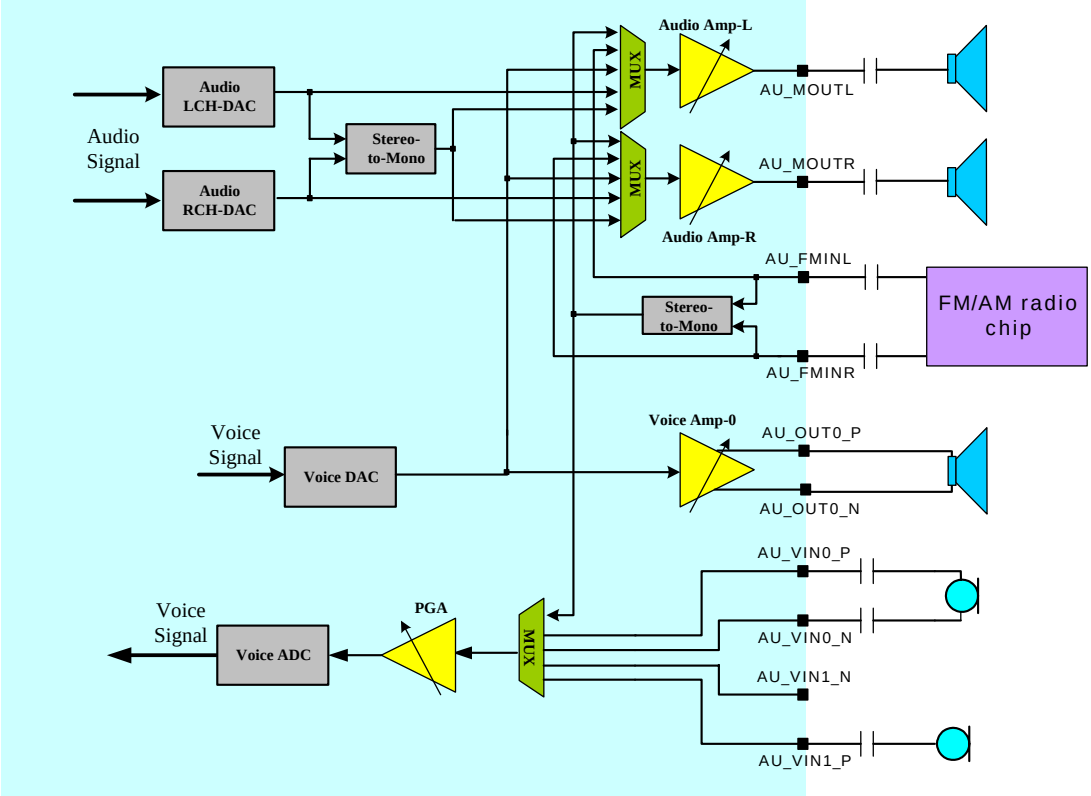


Figure 6 Audio Front End Configuration of MT6226/27

2.3 Speaker and Microphone Definition

In L1Audio software module, the following LOGICAL speakers/microphones are defined for MMI/Media usage.

Parameter	Description
L1SP_MICROPHONE1	the normal microphone on the handset
L1SP_MICROPHONE2	the microphone on the headset/earphone
L1SP_SPEAKER1	the normal speaker for speech.
L1SP_SPEAKER2	the headset/earphone speaker
L1SP_LOUD_SPEAKER	the loud speaker for ringtone melody

The definition of these logical devices is at mcu\custom\audio\project_name\audcoeff.c.
Usually look like:

```
const unsigned char  L1SP_MICROPHONE1  = L1SP_LNA_0;
const unsigned char  L1SP_MICROPHONE2  = L1SP_LNA_1;
const unsigned char  L1SP_SPEAKER1     = L1SP_BUFFER_0;
const unsigned char  L1SP_SPEAKER2     = L1SP_BUFFER_ST;
const unsigned char  L1SP_LOUD_SPEAKER = L1SP_BUFFER_ST|L1SP_BUFFER_EXT;
```

The logical devices are defined with some PHYSICAL hardware components.
The names for the physical hardware components are defined in mcu\interface\l1audio\l1audio.h

```
#define  L1SP_BUFFER_0       0x01    /* NOTE: Don't use buffer definition directly   */
#define  L1SP_BUFFER_1       0x02    /*       Use speaker definition below          */
#define  L1SP_BUFFER_ST      0x04
#define  L1SP_BUFFER_EXT     0x08
#define  L1SP_BUFFER_EXT_G   0x10
#define  L1SP_STEREO2MONO    0x20    /* Do not use this term for speaker definition */
#define  L1SP_BUFFER_ST_M    (L1SP_BUFFER_ST|L1SP_STEREO2MONO)

#define  L1SP_LNA_0          0
#define  L1SP_LNA_1          1
#define  L1SP_LNA_FMRR       3
```

Parameter	Description
L1SP_BUFFER_0	The voice buffer 0(Voice Amp-0 in the diagram of the previous section)
L1SP_BUFFER_1	The voice buffer 1(Voice Amp-1 in the diagram of the previous section)
L1SP_BUFFER_ST	The stereo buffers
L1SP_BUFFER_EXT	The external amplifier
L1SP_BUFFER_EXT_G	Currently not used(defined fo gain control for the external amplifier)
L1SP_STEREO2MONO	The stereo-to-mono converter in the diagram of the previous section
L1SP_BUFFER_ST_M	The stereo buffers with mono output
L1SP_LNA_0	The first microphone input path for the normal microphone
L1SP_LNA_1	The second microphone input path for the headset microphone
L1SP_LNA_FMRR	The input path for FM radio recording. Can only be used in MT6226/27 and later chipset platform.

2.4 Volume Setting

In “mcu\custom\audio\project_name\AFE.C” or “mcu\l1audio\AFE2.C”, two functions are used to set the output/input volumes:

void AFE_SetMicrophoneVolume(kal_uint8 mic_volume)

void AFE_SetOutputVolume(kal_uint8 aud_func, kal_uint8 volume1, kal_uint8 volume2)

The gain values are used to be set into specific hardware PGAs. No digital gain is used here.

The hardware setting for the gain control may have 16 or 32 steps, while each step may have 2dB or 3dB difference.

However ,for future portability, the volumes are NORMALIZED to be in kal_unit8 type, i.e. value in 0~255.

For example,

AFE_SetOutputVolume(L1SP_SPEECH, 80, 80)

The value 80 should be divided by 16 and the result is 5. 5 is the value that is written into hardware PGA.

Another example:

AFE_SetMicrophoneVolume(112)

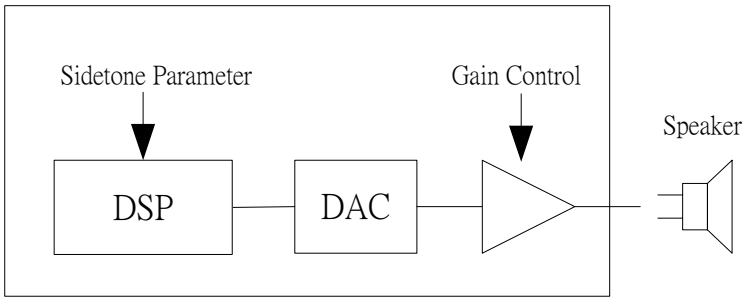
The value 112 is divided by 8 because the input LNA has 32 steps gain control.

112/8= 14, 14 is the value written into the LNA.

A special value for AFE_SetOutputVolume() is 0. Whenever the volume is set as 0, the amplifier will not be open and results in no audible sound.

2.5 Sidetone Auto-compensation

While dealing with sidetone control, one problem is encountered. See the diagram below.



We can see that the sidetone parameter is given to the DSP, however, the output amplifier is used for volume control. So, if we get a louder volume, a fixed sidetone parameter will make the sidetone to be louder accordingly.
A compensation table should be defined to solve this problem. Since the volume control amplifier has 16 steps, each of which has a 2.0dB difference to the adjacent steps, we need a table with 16 fields. See the following code.

```
const static uint16   SIDE_TONE_TABLE[] = {
   26943, 21400, 17000, 13503,
   10726,  8520,  6768,  5376,
    4270,  3392,  2694,  2140,
    1700,  1350,  1070,   850,
};
```

The louder the volume is, the smaller the sidetone parameter will be. The gain between two parameters can be calculated:

Gain = 20*log(21400 / 26943)
= -2.0 dB

So, the sidetone is compensated and maintained to have fixed volume.

2.6 AFE Manager Source Code

The source code for hardware AFE control can be referred at

mcu\customer\audio\project_name\afe.c	Customization part (for MT6205B, this file is the main control part)
mcu\l1audio\afe2.c	Main control part (for MT6205B, this file doesn't exist)

2.6.1 Interface Functions

The prototypes are listed as follows

Prototype	Description
void AFE_Init(void)	The initialization function which executs once when system bootup.
void AFE_TurnOn8K(void)	Turn on the hardware 8K clock tick. The 8K clock is for voice output(with 8K/sec sample rate) and this clock is also used to tick the DSP for voice/speech processing.
void AFE_TurnOff8K(void)	Turn off the hardware 8K clock.
void AFE_TurnOnAudioClock(kal_uint8 clock)	Turn on the hardware audio clock(32K/44.1K/48K). The audio clock is for MP3/AAC/WAV/MIDI playback.
void AFE_TurnOffAudioClock(void)	Turn off the hardware audio clock.
void AFE_TurnOnDAI(void)	Enable hardware DAI mode
void AFE_TurnOffDAI(void)	Disable hardware DAI mode
void AFE_TurnOnBluetooth(kal_uint16 param)	Turn on the hardware PCM interface for Bluetooth speech data transmission.
void AFE_TurnOffBluetooth(void)	Turn off the hardware PCM interface
void AFE_SetInputSource(kal_uint8 src)	Select the microphone input for the multiplexer in front of the LNA and ADC.
kal_uint8 AFE_GetInputSource(void)	Return the value that is assigned to AFE_SetInputSource.
void AFE_SetMicrophoneVolume(kal_uint8 mic_volume)	Set the volume(gain) of the microphone
kal_uint8 AFE_GetMicrophoneVolume(void)	Get the microphone gain.
void AFE_SetSidetoneVolume(kal_uint8 sidetone)	Set the volume of sidetone.
kal_uint8 AFE_GetSidetoneVolume(void)	Get the volume of sidetone
void AFE_SetSidetone(kal_bool ec)	Enable or disable sidetone.
void AFE_SetOutputDevice(kal_uint8 aud_func, kal_uint8 device)	Set the output device of a certain audio function
kal_uint8 AFE_GetOutputDevice(kal_uint8 aud_func)	Get the output device of a certain audio function
void AFE_SetOutputVolume(kal_uint8 aud_func, kal_uint8 volume1, kal_uint8 volume2)	Set the output volume of a certain audio function
void AFE_GetOutputVolume(kal_uint8 aud_func, kal_uint8 *volume1, kal_uint8 *volume2)	Get the output volume of a certain audio function
void AFE_TurnOnSpeaker(kal_uint8 aud_func)	Turn on speaker for a specific audio function
void AFE_TurnOffSpeaker(kal_uint8 aud_func)	Turn off speaker
void AFE_MuteSpeaker(kal_uint8 aud_func)	Mute speaker
void AFE_TurnOnMicrophone(kal_uint8 aud_func)	Turn on microphone for a specific audio function
void AFE_TurnOffMicrophone(kal_uint8 aud_func)	Turn off microphone

void AFE_MuteMicrophone(kal_bool mute)	Mute the microphone
void AFE_TurnOnExtAmplifier(void)	Force to turn on the external amplifier
void AFE_TurnOffExtAmplifier(void)	Turn off the external amplifier
void AFE_Manager(void)	The AFE Manager main function. this function runs under CTIRQ LISR context. All the AFE settings are done here.
kal_uint8 AFE_TurnOnMicBias(void)	Turn on microphone bias voltage (for accessary device detection
void AFE_TurnOffMicBias(void)	Turn off microphone bias voltage

2.6.2 Internal Static Functions

Prototype	Description
static void UpdateSidetone(kal_int8 vol)	Update sidetone value(with auto compensation algorithm)
static void UpdateVGain(kal_uint8 aud_func)	Update the gain of the voice amplifier(BUF0/BUF1)
static void UpdateAGain(kal_uint8 aud_func)	Update the gain of the stereo audio amplifier
static void UpdateVAPDN(kal_uint8 aud_func, kal_uint8 device)	Update the value of VAPDN register(refer to the MT62XX chipset datasheet, “Analog Front End section”). This register is used to turn on/off microphone and amplifiers(BUF0 and BUF1)
static void SearchSpkFlag(kal_int16 *v_lowest, kal_int16 *a_lowest)	This function is used to search for the highest prority audio function that is currently using the speaker.

2.6.3 What is “Audio Function”

“Audio Function” is defined “mcu\interface\l1audio\l1audio.h” and is used INTERNALLY inside l1audio software module.

```
#define  L1SP_KEYTONE        0
#define  L1SP_TONE           1
#define  L1SP_SPEECH         2
#define  L1SP_SND_EFFECT     3
#define  L1SP_VMP            4
#define  L1SP_VMR            5
#define  L1SP_VR             6
#define  L1SP_MEDIA          7
#define  L1SP_DAI            8
#define  L1SP_FM_RADIO       9
#define  L1SP_A_MEDIA        10
#define  L1SP_PCM_EX         11
```

This term is used to classify certain audio applications. An audio function with lower definition value is assumed to have higher proirity than those with higher values.

Whenever two audio functions are running concurrently, the resource that is occupied by the original low priority audio function should be freed and reassigned to the high priority one. After the higher priority audio function ends, the resource is returned.
For example, key tone playback in speech mode. Keytone has higher prioroty than speech. When a user press a keytone in speech mode, the keytone function works and preempt the speaker resource from “speech function”. When it ends, the speaker is returned to speech function.

2.6.4 AFE data structure

```
struct {
  /// Speaker usage flag
  /// bit n : aud function n
  /// [Set|Clear] by AFE_Turn[On|Off]Speaker
  kal_uint16     sp_flag;
  /// FIR filter usage flag
  /// bin n: aud function n
  /// [Set|Clear] by AFE_Turn[On|Off]FIR
  kal_uint16     fir_flag;
  /// Microphone usage flag
  /// bin n: aud function n
  /// [Set|Clear] by AFE_Turn[On|Off]Microphone
  kal_uint16     mic_flag;
  /// L1SP_LNA_0, L1SP_LNA_1, or L1SP_LNA_FMRR
  kal_uint8      mic_src;
  kal_uint8      mic_volume;
  kal_bool       mic_mute;
  kal_uint8      sidetone_volume;
  /// sidetone disable flag
  kal_bool       sidetone_disable;
  struct   {
    /// L1SP_SPEAKER1, L1SP_SPEAKER2, or L1SP_LOUDSPEAKER
    /// Set by AFE_SetOutputDevice
    kal_uint8   out_dev;
    kal_uint8   volume;
    kal_uint8   volume_st;
  } aud[L1SP_MAX_AUDIO];
  RINGBUFFER_T(DelayCmd,8)  regq;
  kal_bool       refresh;
  kal_bool       accessory_flag;
  /// for controlling external amplifier
  kal_bool       gpio_lock;
  kal_bool       ext_op_on;
  kal_int16      ext_op_delay;
} afe;
```

2.6.5 Customization

In afe2.c under custom folder, there is 2 functions: AFE_Initialize and AFE_SwitchExtAmplifier.

AFE_Initialize sets AFE_VAC_CON0::VDSEND for single-ended configuration control for voice buffer 0 and 1. It should match external circuit layout.

AFE_SwitchExtAmplifier turns on/off external amplifier.

2.6.6 Analog Die

AFE_VAC_CON0::VCALI determines the biasing current in analog die, it is implemented in the function Set_AFE_VAC_CON0_VCALI in afe.c. The value is set according to which analog die is used.

3 AM(Audio Manager) module

3.1 Concept

Refer to the block diagram below. AM is used mainly to control the DSP. It also commands AFE manager to turn on/off the corresponding speaker amplifiers or microphone input device.
The main function of AM is running under the context of CTIRQ LISR, i.e. the highest priority ISR that is triggerred every 4.615ms periodically in the system.

For speech synchronization, AM also gets the exact frame number from L1D and set the proper time/delay parameters for the DSP to do the synchronization between channel codec and speech codec.

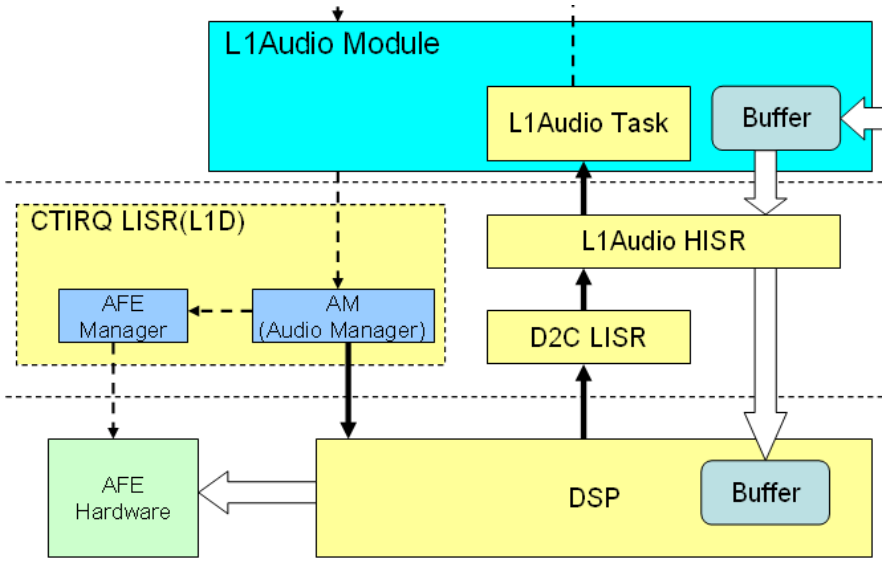


Figure 7 The Position of AM(Audio Manager)

3.2 Enqueue a Function to Run in CTIRQ LISR

For running a function in CTIRQ LISR, a command queue is introduced. Refer to the diagram below.

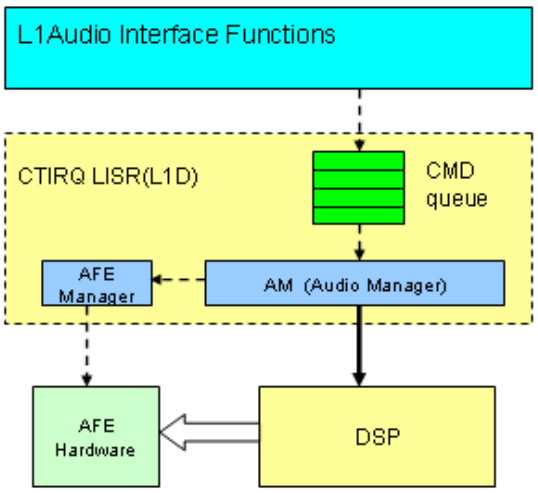


Figure 8 The Command Queue for Giving Command AM

Every 4.615ms, whenevery the CTIRQ is triggerred, AM is called by L1D. AM will check if there is any command inside the command queue. If the command queue is not empty, a command is dequeued and executed.
Note that since the CTIRQ LISR cannot be pended too long, only one command will be done at this moment. Others will be delayed to the next CTIRQ arrival. This approach is for spreading the loading over several CTIRQs and is assumed to be no harm to the audio functions and communication functions.

3.3 AM (Audio Manager) Source Code

The source code can be referred at mcu\l1audio\am.c

3.3.1 Interface Functions

The prototypes are listed as follows. Note that all these interface functions should only be used by l1audio software module. None of them are interface functions for MMI/Media task.

Prototype	Description
bool AM_AudioManager(bool handover, uint16 speech_mode)	AM main function. This function runs under the context of CTIRQ, the highest priority LISR, for easily keeping synchronization with the DSP
void AM_WriteFirCoeffs(const int16 in_coeff[30], const int16 out_coeff[30])	Write speech input/output compensation filter coefficients(FIR filter)
void AM_SpeechOn(void)	Turn on speech
void AM_SpeechOff(void)	Turn off speech
bool AM_IsSpeechOn(void)	Check if speech mode is enabled
void AM_SetDAIMode (uint8 mode)	Set DAI mode
bool AM_IsDAIMode(void)	Check if DAI mode is enabled

void AM_VMRecordOn(uint16 speech_mode)	Enable DSP Voice Memo recording function. The recorded file format is a proprietary "*.VM" format.
void AM_VMRecordOff(void)	Disable DSP Voice Memo recording function
void AM_VMPlaybackOn(uint16 control, uint32 style)	Enable DSP Voice Memo playback function
void AM_VMPlaybackOff(uint32 style)	Disable DSP Voice Memo playback function
void AM_KeyToneOn(void)	Enable DSP keytone synthesis.(DTMF)
void AM_KeyToneOff(void)	Turn off DSP keytone synthesis.
void AM_ToneOn(void)	Turn on the DSP tone synthesis function(Same as keytone interface)
void AM_ToneOff(void)	Turn off the DSP tone synthesis function
void AM_PCM8K_PlaybackOn(void)	Turn on PCM 8K playback channel(mono)
void AM_PCM8K_PlaybackOff(bool wait)	Turn off PCM 8K playback
void AM_PCM8K_RecordOn(void)	Turn on PCM 8K recording function.
void AM_PCM8K_RecordOff(bool wait)	Turn off PCM 8K recording function
void AM_AudioPlaybackOn(uint16 asp_type, uint16 asp_fs)	Turn on PCM playback channel(variable sampling rate from 8K to 48K, can be stereo or mono)
void AM_AudioPlaybackOff(bool wait)	Turn of PCM playback channel
int16 AM_IsAudioPlaybackOn(void)	Check if the audio PCM playback function is enabled.
void AM_MuteULSpeech(bool mute)	Mute uplink speech channel
void AM_Init(void)	Initialization function running at the very beginning when system bootup.

4 Speech Interface

4.1 Functions

L1 Audio module provides some speech related functions to select input and output device, do volume control, and enter/leave speech mode.

4.1.1 L1SP_SetOutputDevice

void L1SP_ SetOutputDevice(uint8 device)

Set the output device of speech-related function.

Parameters:

Parameter	Description
device	The output device (choose one among L1SP_SPEAKER1, L1SP_SPEAKER2, L1SP_LOUDSPEAKER)

Note: L1SP_SPEAKER1 means the normal speaker, L1SP_SPEAKER2 means the headset speaker, and L1SP_LOUDSPEAKER is used for handsfree phone application

Return value:
None

Example:
L1SP_SetOutputDevice (L1SP_SPEAKER1);

4.1.2 L1SP_SetOutputVolume

void L1SP_SetOutputVolume(uint8 volume1, int8 digital_gain_index)

Set the output volumes of speech-related function.

Parameters:

Parameter	Description
volume1	The gain of the output buffer 0/1 (0-255, 2dB per 16 steps).
digital_gain_index	The digital gain, 0 (0dB) to -128 (-64dB), 0.5dB per step)

Return value:
None

Example:
L1SP_SetOutputVolume(96, 80);

4.1.3 L1SP_SetInputSource

void L1SP_SetInputSource(uint8 src)

This function is for setting input source.

Parameters:

Parameter	Description
src	The input source (choose one between L1SP_MICROPHONE1, L1SP_MICROPHONE2)

Return value:
None

Example:
L1SP_SetInputSource(L1SP_MICROPHONE1);

4.1.4 L1SP_SetMicrophoneVolume

void L1SP_SetMicrophoneVolume(uint8 mic_volume)

This function is for setting microphone volume.

Parameters:

Parameter	Description
mic_volume	The gain of the microphone amplifier (0-255, 2dB per 16 steps).

Return value:
None

Example:
L1SP_SetMicrophoneVolume(80);

4.1.5 L1SP_SetSidetoneVolume

void L1SP_SetSidetoneVolume(uint8 sidetone_volume)

This function is for setting sidetone volume.

Parameters:

Parameter	Description
sidetone_volume	sidetone volume (0-255, 2dB per 16 steps).

Return value:
None

Example:
L1SP_SetMicrophoneVolume(80);

4.1.6 L1SP_MuteMicrophone

void L1SP_MuteMicrophone (bool mute);

This function is used to mute the microphone.

Parameters:

Parameter	Description
mute	**true** -> mute the microphone. **false** -> resume the microphone

Return value:
None

Example:
L1SP_MuteMicrophone (**true**);

4.1.7 L1SP_Speech_On

void L1SP_Speech_On(void)

This function is used to control L1Audio module and the DSP to be in speech mode when getting an incoming call or originating an outgoing call.

Parameters:
None

Return value:
None

Example:
None

4.1.8 L1SP_Speech_Off

void L1SP_Speech_Off(void)

This function is used to leave speech mode when the call is terminated.

Parameters:
None

Return value:
None

Example:
None

4.1.9 L1SP_SetSpeechVolumeLevel

void L1SP_SetSpeechVolumeLevel(kal_uint8 level)

This function indicates the current output volume level.

Parameters:

Parameter	Description
level	The current output volume level from 0 to 6. It is used to choose the speech enhancement algorithm.

Return value:
None

Example:
None

4.1.10 L1SP_LoadCommonSpeechPara

void L1SP_LoadCommonSpeechPara(uint16 c_para[NUM_COMMON_PARAS])

This function loads the common parameters of speech enhancement in all speech modes.
The default parameter values are defined in audcoeff.c.

Parameters:

Parameter	Description
c_para	The common parameters for speech enhancement in all speech mode

Return value:
None

Example:
None

4.1.11 L1SP_SetSpeechMode

void L1SP_SetSpeechMode(uint8 mode, uint16 m_para[NUM_MODE_PARAS])

This function indicates the speech mode used to choose different speech enhancement algorithm.
Currently there are 6 modes, defined in l1audio.h:

```
#define SPH_MODE_NORMAL         0
#define SPH_MODE_EARPHONE       1
#define SPH_MODE_LOUDSPK        2
#define SPH_MODE_BT_EARPHONE    3
#define SPH_MODE_BT_CORDLESS    4
#define SPH_MODE_BT_CARKIT      5
```

When the speech mode is switched between Bluetooth (`SPH_MODE_BT_EARPHONE`, `SPH_MODE_BT_CORDLESS` and `SPH_MODE_BT_CARKIT`) and local path (`SPH_MODE_NORMAL`, `SPH_MODE_EARPHONE` and `SPH_MODE_LOUDSPK`), this function will also configure Bluetooth path.
The default parameter values are defined in audcoeff.c.

Parameters:

Parameter	Description
mode	Indicates the current speech mode. It is used to choose different speech enhancement algorithm.
m_para	The parameters for speech enhancement in the indicated speech mode

Return value:
None

Example:
None

4.1.12 L1SP_EnableSpeechEnhancement

void L1SP_EnableSpeechEnhancement(bool enable)

This function is used to turn on/off speech enhancement. The speech enhancement is turned on in default.

Parameters:

Parameter	Description
enable	true -> turn on speech enhancement false -> turn off speech enhancement

Return value:
None

Example:
None

4.1.13 L1SP_Set_DAI_Mode

void L1SP_Set_DAI_Mode(uint8 mode)

This function is used to enter and leave DAI mode, and called by L1 task only.

Parameters:

Parameter	Description
mode	DAI mode mode = 0 -> Normal operation mode = 1 -> DAI mode 1. Test of speech decoder/DTX function(downlink) mode = 2 -> DAI mode 2. Test of speech encoder/DTX function(uplink) mode = 4 -> DAI mode 3. Test of acoustic devices and A/D & D/A

Return value:
None

Example:
/* Enter DAI test mode */
L1SP_Set_DAI_Mode(4);

/* Leave DAI test mode */
L1SP_Set_DAI_Mode(0);

4.1.14 L1SP_Write_Audio_Coefficients

void L1SP_Write_Audio_Coefficients(int16 in_coeff[30], int16 out_coeff[30])

This function is used to set the coefficients for the DSP uplink and downlink compensation FIR filters. And, it is called only by L1 when system initialization.

Parameters:

Parameter	Description
in_coeff	An array containing the uplink FIR coefficients. Unused fields should be filled with zeros
out_coeff	An array containing the downlink FIR coefficients. Unused fields should be filled with zeros

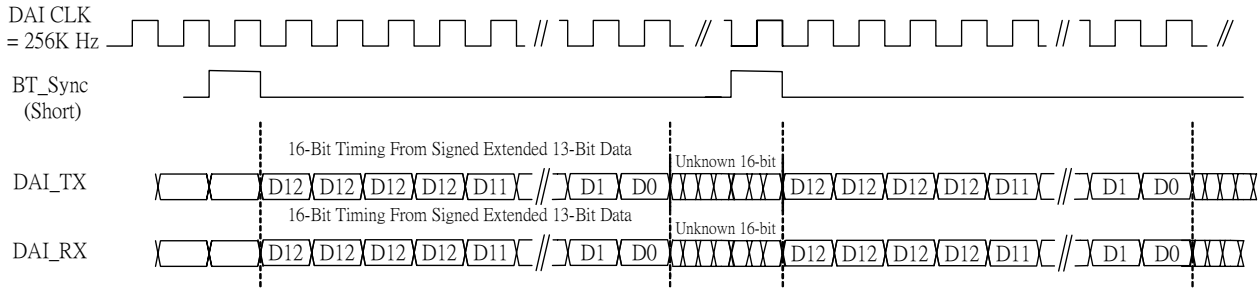
Return value:
None

Example:
```
const int16 Audio_Input_Coeff[30] =
{    0x7FFF, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000,
     0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000,
     0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000
};

const int16 Audio_Output_Coeff[30] =
{    0x7FFF, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000,
     0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000,
     0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000
};

AM_WriteAudioCoeff( Audio_Input_Coeff, Audio_Output_Coeff );
```

4.2 Bluetooth Speech Interface

4.2.1 Hardware PCM Interface Timing

The speech data is generated by the DSP and outputed to the hardware PCM interface.
The timing chart is shown below.



Timing of BT Earphone Mode Configuration

4.2.2 Software Interface Functions

The Bluetooth speech control is integrated into L1SP_SetSpeechMode.

5 Key Tone Interface

5.1 Concept

This is a specific function for key-tone playback.
The key-tone function cannot be overlapped, i.e. whenever a key-tone is playing, any other key-tone playing commands will be ignored automatically.

5.2 Functions

5.2.1 KT_SetOutputDevice

void KT_SetOutputDevice(uint8 device)

Set the output device of the key-tone function.

Parameters:

Parameter	Description
device	The output device (choose one among L1SP_SPEAKER1, L1SP_SPEAKER2, L1SP_LOUDSPEAKER)

Note: L1SP_SPEAKER1 means the normal speaker, L1SP_SPEAKER2 means the headset speaker, and L1SP_LOUDSPEAKER is used for handsfree phone application

Return value:
None

Example:
KT_SetOutputDevice (L1SP_SPEAKER1);

5.2.2 KT_SetOutputVolume

void KT_SetOutputVolume(uint8 volume1, uint8 volume2)

Set the output volumes of the key-tone function.

Parameters:

Parameter	Description
volume1	The gain of the output buffer 0/1 (0-255, 2dB per 16 steps).
volume2	The gain of the stereo buffer (0-255, 2dB per 16 steps).

Return value:
None

Example:
KT_SetOutputVolume (96, 96);

5.2.3 KT_Play

void KT_Play(uint16 freq1, uint16 freq2, uint16 duration)

This function is used to play key-tone.

Parameters:

Parameter	Description
freq1	First frequency of the key-tone
freq2	Second frequency of the key-tone
duration	Duration of the key-tone, in milli-second, 0 for continues tone.

Return value:
None

Example 1:
```
KT_Play( 425, 510,  0 );          /* Start play key-tone */
…
KT_Stop();                        /* Stop key-tone immediately */
```

Example 2:
```
KT_Play( 425, 510,  100 );        /* Play a key-tone for 100 ms */
```

5.2.4 KT_Stop

void KT_Stop(void)

Stop key-tone immediately.

Parameters:
None

Return value:
None

Example:
```
KT_Stop();
```

6 Tone Interface

6.1 Concept

This is a generic tone playing function.
The tone interface is a flexible, and programmable interface which can be configured to play tone in several operation modes, listed below.
It is suggested not to use this tone function for key-tone playing. The key-tone function described in the previous chapter has higher precedence than the generic tone function and can interrupt it whenever the user presses a key. The tone function is suggested to be for in-call alerting only. Additionally, these two interfaces (key-tone and the generic tone) provide respective output device and volume control functions.

6.1.1 Continuous Tone

Continuous tone operation mode is used in key tone playing.

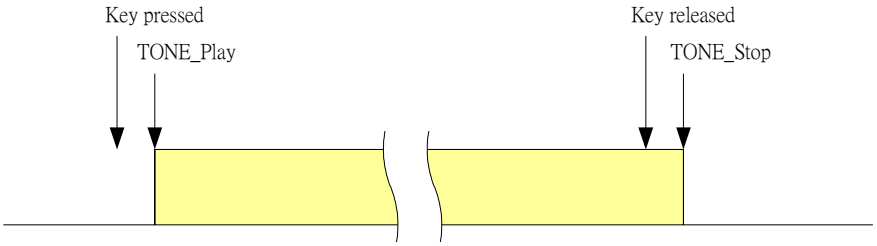


Figure 9 Continuous Tone

6.1.2 Programmed Tones

Programmed tones operation mode is very convenient for playing those tones with predefined frequencies and durations.

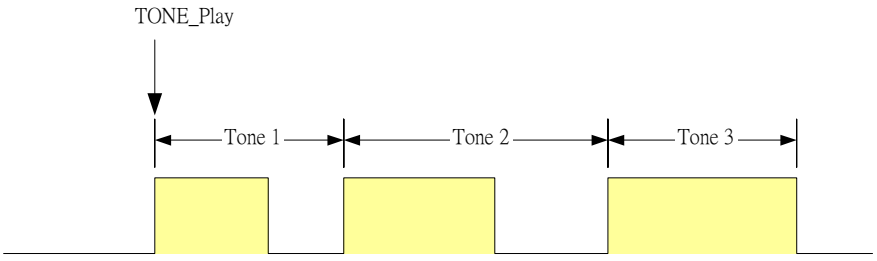


Figure 10 Programmed Tones

6.1.3 Programmed Tones with Repeats

This operation mode is the extension of the previous one. After playing the last tone in a tone table, the first tone is fetched and played. The sequence repeats until MMI calls L1SP_TONE_Stop function to stop the operation.

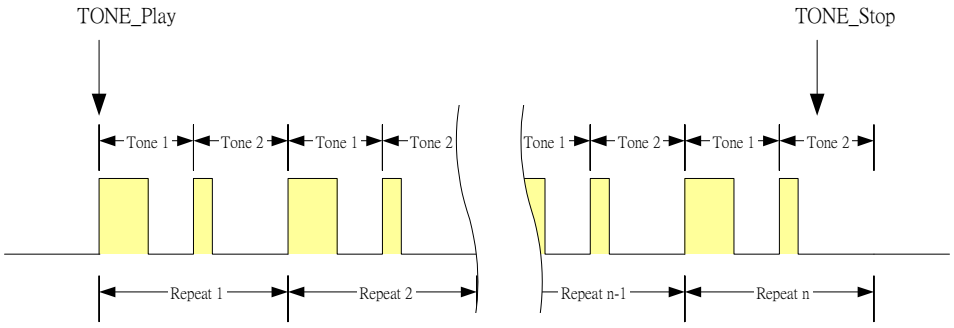


Figure 11 Programmed Tones with Repeats

6.2 Type Definitions

6.2.1 L1SP_Tones

Definition:

```
typedef struct {
  uint16   freq1;          /* First frequency                                  */
  uint16   freq2;          /* Second frequency                                 */
  uint16   on_duration;    /* Tone on duation, in ms unit, 0 for continuous tone */
  uint16   off_duration;   /* Tone off duation, in ms unit, 0 for end of playing */
  int8     next_tone;      /* Index of the next tone                           */
} L1SP_Tones;
```

This structure contains the necessary information for playing a tone. It includes two frequencies and on/off durations. This structure can be used to define a tone table that can be viewed as a program of tone playing.

Members:

Members	Description
freq1	The first frequency of the tone.
freq2	The second frequency of the tone.
on_duration	ON duration of the tone (in millisecond unit). Give 0 for a continuous tone, no matter what off_duration and next_tone is given.
off_duration	OFF duration of the tone (in millisecond unit) Give 0 for End-of-Playing. Tone playing ends when getting a zero off_duration, no matter what next_tone is given.
next_tone	The index of the next tone in a tone table. The index starts from 0 to N-1, where N is the total amount of tones in a tone table.

6.3 Functions

6.3.1 TONE_SetOutputDevice

void TONE_SetOutputDevice(uint8 device)

Set the output device of the tone function.

Parameters:

Parameter	Description
device	The output device (choose one among L1SP_SPEAKER1, L1SP_SPEAKER2)

Return value:
None

Example:
TONE_SetOutputDevice(L1SP_SPEAKER1);

6.3.2 TONE_SetOutputVolume

void TONE_SetOutputVolume(uint8 volume)

Set the output volumes of the tone function.

Parameters:

Parameter	Description
volume	The gain of the output amplifier (0-255, 2dB per 16 steps).

Return value:
None

Example:
TONE_SetOutputVolume (96);

6.3.3 TONE_Play

void TONE_Play(const L1SP_Tones *tonelist)

This function is used to play tones.

Parameters:

Parameter	Description
tonelist	An array of tone (tone table). TONE_Play function plays the tones in sequence by following the next_tone member of each tone.

Return value:
None

Example 1 (continuous tone):

```
static const L1SP_Tones tone1[] =  { { 425,   0,   0,   0,   0 }   };          // Note 0, continuous(because on_duration=0)
TONE_Play( tone1 );
```

Example 2 (programmed tones):

```
static const L1SP_Tones tone2[] = {         { 425,   0, 200, 200,   1 },          // Note 0, next_tone=1
                                            { 425,   0, 200, 200,   2 },          // Note 1, next_tone=2
                                            { 425,   0, 200,   0,   0 }   };      // Note 2, end(because off_duration=0)
TONE_Play( tone2 );
```

Example 3 (programmed tones with repeats):

```
static const L1SP_Tones tone3[] = {         { 425,   0, 200, 200,   1 },          // Note 0, next_tone=1
                                            { 425,   0, 200, 200,   2 },          // Note 1, next_tone=2
                                            { 425,   0, 200, 200,   0 }   };      // Note 2, next_tone=0
TONE_Play( tone3 );
```

6.3.4 TONE_Stop

void TONE_Stop(void)

Stop tone playing immediately.

Parameters:
None

Return value:
None

Example:
TONE_Stop();

7 SND interface

7.1 Concept

The SND interface is used to play sound effect in idle/ dedicated mode. The sound effect is associated by a FSAL instance. The supported sound file format are 8/16-bit linear PCM, G711 A/u-law and DVI-ADPCM in MS RIFF format with 8k sample rate.

The state transition table is shown below: the master should follow the state transition to use the interface.

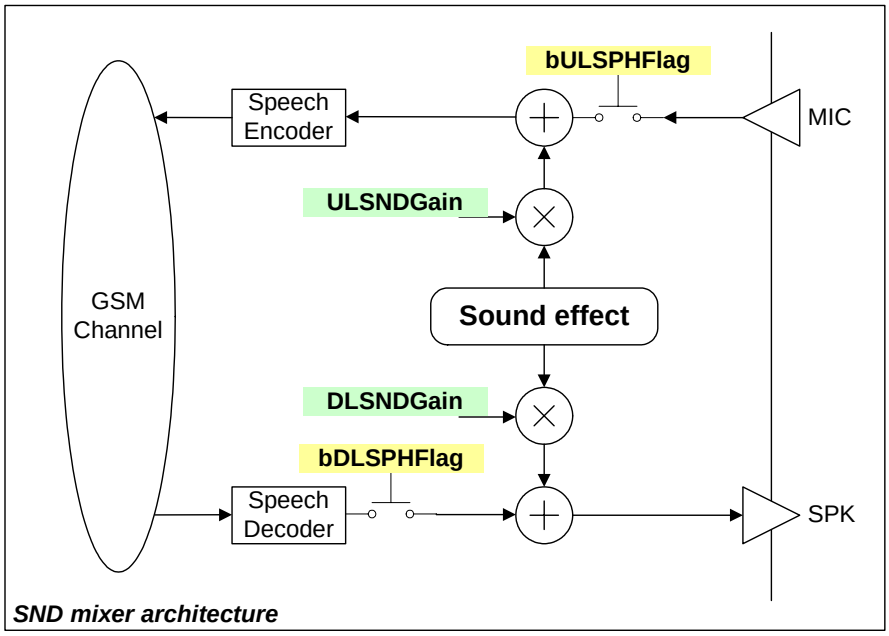
State transition	IDLE	OPENED	PLAYING	STOPPED
SND_GetFormat	IDLE	OPENED	PLAYING	STOPPED
SND_SetOutputDevice	IDLE	OPENED	PLAYING	STOPPED
SND_Open	OPENED			
SND_Play		PLAYING		PLAYING
SND_Stop			STOPPED	STOPPED
SND_Close				IDLE
SND_ConfigULMixer		OPENED	PLAYING	STOPPED
SND_ConfigDLMixer		OPENED	PLAYING	STOPPED

Table 1 SND state transition table

In idle mode, the SND interface can be used to play sound file. Pause/ resume operation is not supported.
In dedicated mode, the SND interface provides the three individual operation modes at both uplink and downlink path:

- Voice
- Sound effect with configurable gain
- Voice + sound effect with configurable gain

The sound effect mixer architecture and four parameters are defined as follows to support the above features.



SND mixer architecture

Figure 1. SND mixer architecture

The setting of the four parameters in idle/ dedicated mode are shown below. (X means no effect)
Idle mode:

bULSPHFlag	ULSNDGain	bDLSPHFlag	DLSNDGain	Description
X	X	X	0 ~ 7	User hears SND if DLSNDGain is not zero.

Dedicated mode: (setting gain to 0 mutes the sound effect and makes the user hears SPH only)

bULSPHFlag	ULSNDGain	bDLSPHFlag	DLSNDGain	Description
TRUE	1 ~ 7	TRUE	1 ~ 7	Both of UL and DL hear SPH + SND
TRUE		FALSE		UL hears SPH + SND, DL hears SND
FALSE		TRUE		UL hears SND, DL hears SPH + SND
FALSE		FALSE		Both of UL and DL hear SND only

7.1.1 Type definition

7.1.1.1 SND_Param

Definition:

```
typedef struct {
  STFSAL        *pstFSAL;
  kal_int16      repeats;  /* 0 -> endless */
  Media_Format  format;
} Snd_Param;
```

Members:

Member	Description
pstFSAL	The pointer to the FSAL instance of the sound.
repeats	The repeat times the sound should be played. (0 implies endless)
format	The sound format.

7.1.2 SND function

7.1.2.1 SND_GetFormat

Media_Status SND_GetFormat(STFSAL *pstFSAL, Media_Format *pFormat);

The function is used to check if the sound format of a FSAL instance is supported by this SND interface.

Parameters:

Parameter	Description
pstFSAL	The pointer to the FSAL instance
pFormat	The pointer to the sound format of the FSAL instance. It is valid only when the function return

Return value:

Return **MEDIA_SUCCESS** if this SND interface supports the sound format of the FSAL instance. Return **MEDIA_BAD_FORMAT** otherwise.

Example:

```
STFSAL fsalSoundFile;
Media_Format soundFormat;

eFSALRet = FSAL_Open( &fsalSoundFile, (void *)path, FSAL_READ );
…
If ( SND_GetFormat( fsalSoundFile, &soundFormat ) == MEDIA_SUCCESS )
…
```

7.1.2.2 SND_SetOutputDevice

void SND_SetOutputDevice(kal_uint8 device);

The function sets the output device only when sound effect is played at idle mode.

Parameters:

Parameter	Description
device	The output device (L1SP_SPEAKER1, L1SP_SPEAKER2, L1SP_LOUD_SPEAKER)

Return value:

None

Example:

```
SND_SetOutputDevice(L1SP_LOUD_SPEAKER);
```

7.1.2.3 SND_Open

Media_Handle *SND_Open(void(*handler)(Media_Handle *, Media_Event), Snd_Param **param);

The function opens a handle for background sound playback. It should be invoked before any other operation.

Parameters:

Parameter	Description
handler	A callback function used for handling **MEDIA_END** and **MEDIA_REPEATED** events.
param	Specific parameter for the background sound playback.

Return value:

Return the media handle.

Example:

```
#define SND_MIX_DL  1
#define SND_MIX_UL  2
STFSAL fsalSoundFile;
Media_Handle *pstMediaHandle;
```

```
Snd_Param *pstSndParam;

eFSALRet = FSAL_Open( &fsalSoundFile, (void *)path, FSAL_READ );
…
pstMediaHandle = SND_Open( snd_callback, &pstSndParam);
pstSndParam->pstFSAL = &fsalSoundFile;
pstSndParam->repeats = 0;
…
SND_ConfigULMixer( pstMediaHandle, KAL_TRUE, 100 );
SND_ConfigDLMixer( pstMediaHandle, KAL_TRUE, 10 );
SND_Play( pstMediaHandle );
…
SND_ConfigULMixer( pstMediaHandle, KAL_TRUE, 20 );
SND_ConfigDLMixer( pstMediaHandle, KAL_FALSE, 20 );
…
SND_Stop( pstMediaHandle );
SND_Close( pstMediaHandle );
```

7.1.2.4 SND_Close

Media_Status *SND_Close(Media_Handle *hdl);

The function closes the media handle.

Parameters:

Parameter	Description
hdl	The media handle

Return value:
Return **MEDIA_SUCCESS** when the operation is complete.

Example:
See the example of SND_Open and SND_Stop.

7.1.2.5 SND_Play

Media_Status *SND_Play(Media_Handle *hdl);

The function starts the background sound playback of the media handle.

Parameters:

Parameter	Description
hdl	The media handle

Return value:

Return Value	Description
MEDIA_SUCCESS	Background sound has been played.

MEDIA_REENTRY	Reentrance has occurred. Operation aborted.
MEDIA_NOT_INITIALIZED	The media handle is invalid. Operation aborted.
MEDIA_BAD_FORMAT	Bad sound format. Operation aborted.

Example:
See the example of SND_Open.

7.1.2.6 SND_Stop

void *SND_Stop(Media_Handle *hdl);

The function stops the background sound playback of the media handle.

Parameters:

Parameter	Description
hdl	The media handle

Return value:
None

Example:
1. See the example of SND_Open.
2. Reference design of the callback function handling **MEDIA_END** and **MEDIA_REPEATED** events.

```
void snd_callback( Media_Handle *handle, Media_Event event )
{
  switch( event )
  {
    case MEDIA_END:
      SND_Stop( handle );
      SND_Close( handle );
      break;
    case MEDIA_REPEATED:
      break;
    default:
      ASSERT( 0 );
  }
}
```

7.1.2.7 SND_ConfigULMixer

void SND_ConfigULMixer(Media_Handle *hdl, kal_bool bULSPHFlag, kal_int8 ULSNDGain);

The function sets the playback mode. The function can be invoked after the media handle is opened and during the media handle is playing.
Note: in idle mode, the bULSPHFlag is forced to turn off automatically, so there will be no effect when you turns on bULSPHFlag.

Parameters:

Parameter	Description
hdl	The media handle
bULSPHFlag	Mixer switch for uplink speech
ULSNDGain	Sound effect gain for uplink mixer (0 ~ 100). Note 1: The default value is 100(maximum volume). Note 2: The volume level 0 will mute the sound effect.

Return value:
None

Example:
See the example of SND_Open.

7.1.2.8 SND_ConfigDLMixer

void SND_ConfigDLMixer(Media_Handle *hdl, kal_bool bDLSPHFlag, kal_int8 DLSNDGain);

The function sets the playback mode. The function can be invoked after the media handle is opened and during the media handle is playing.
Note: in idle mode, the bDLSPHFlag is forced to turn off automatically, so there will be no effect when you turns on bDLSPHFlag.

Parameters:

Parameter	Description
hdl	The media handle
bDLSPHFlag	Mixer switch for downlink speech
DLSNDGain	Sound effect gain for downlink mixer (0 ~ 100). Note 1: The default value is 100(maximum volume). Note 2: The volume level 0 will mute the sound effect.

Return value:
None

Example:
See the example of SND_Open.

8 PCM4WAY/PCM2WAY Interface

Audio front-end (AFE) is connected with DSP. For PCM playback, MCU send PCM data to DSP; and for PCM recording , MCU get PCM data from DSP. PCM4WAY/PCM2WAY interface provides a way that exchanges PCM data to and fro with DSP concurrently in both idle mode (PCM2WAY) and in dedicated mode (PCM4WAY).

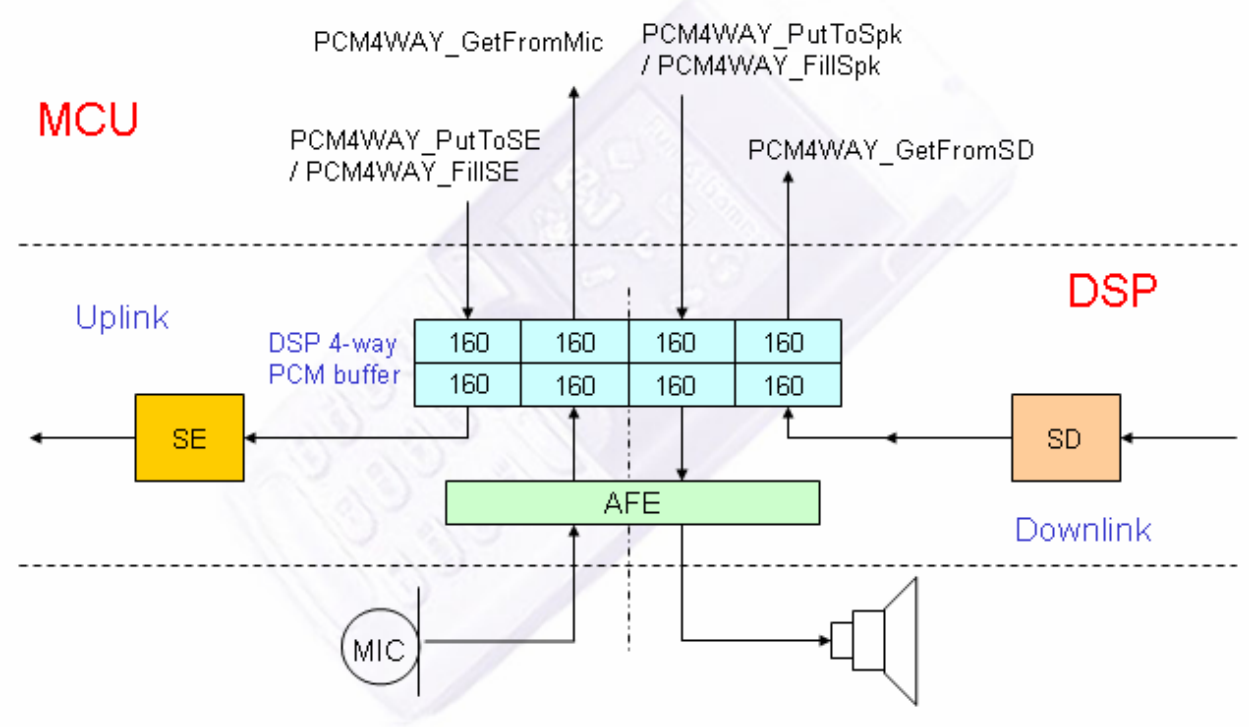
The PCM data are at the rate of speech data , 16-bit 8K Hz, and are used in speech-relative applications such as CTM (using PCM4WAY) or VOIP(using PCM2WAY).

To use PCM2WAY/PCM4WAY interface in code, we must include “pcm4way.h”.

8.1 Concept

8.1.1 PCM4WAY interaction between MCU and DSP

PCM4WAY is operated in dedicated mode. The interaction between MCU and DSP through PCM4WAY interface is shown below:



In DSP side, when PCM4WAY service is started through PCM4WAY_Start,

(1) There is a 4-way PCM buffer ,consisting of 8 160-word buffers, inside DSP to exchange PCM data between MCU and AFE. The PCM buffer is operated in ping-pong mode so that each different data moving is through different 160-word PCM buffer.

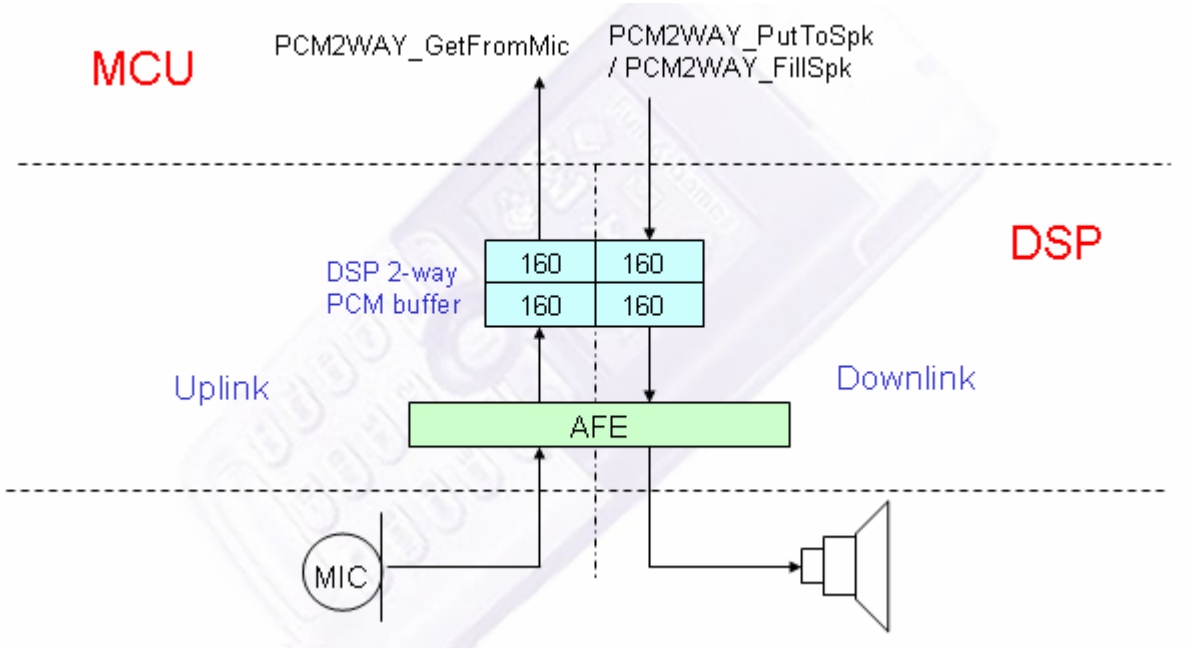
(2) DSP will issue interrupt to inform MCU of the timing to exchange data with DSP.

In MCU side,

(1) Use PCM4WAY_Start to start the PCM4WAY service. In calling PCM4WAY_Start , it must register a handler which will be call-backed each time when L1Audio process the DSP interrupt. (Note : the registered handler is run under the context of L1Audio_HISR.)
(2) Inside the registered handler, for up-link path we could
  (i) use PCM4WAY_GetFromMic to get data from DSP and store to some 160-word PCM buffer.
  (ii) use PCM4WAY_PutToSE to get data from some 160-word PCM buffer and send to DSP or use PCM4WAY_FillSE to fill one specific value to the DSP 160-word buffer.
(3) Inside the registered handler, for down-link path we could
  (i) use PCM4WAY_GetFromSD to get data from DSP and store to some 160-word PCM buffer.
  (ii) use PCM4WAY_PutToSpk to get data from some 160-word PCM buffer and send to DSP or use PCM4WAY_FillSpk to fill one specific value to the DSP 160-word buffer.
(4) Use PCM4WAY_Stop to stop the PCM4WAY service.

8.1.2 PCM2WAY interaction between MCU and DSP

PCM2WAY is operated in idle mode. The interaction between MCU and DSP through PCM2WAY interface is shown below:



In DSP side, when PCM2WAY service is started through PCM2WAY_Start,
(1) There is a 2-way PCM buffer ,consisting of 4 160-word buffers, inside DSP to exchange PCM data between MCU and AFE. The PCM buffer is operated in ping-pong mode so that each different data moving is through different 160-word PCM buffer.
(2) DSP will issue interrupt to inform MCU of the timing to exchange data with DSP.

In MCU side,

(1) Use PCM2WAY_Start to start the PCM2WAY service. In calling PCM2WAY_Start , it must register a handler which will be call-backed each time when L1Audio process the DSP interrupt. (Note : the registered handler is run under the context of L1Audio_HISR.)
(2) Inside the registered handler, we could
  (i) use PCM2WAY_GetFromMic to get data from DSP and store to some 160-word PCM buffer.
  (ii) use PCM2WAY_PutToSpk to get data from some 160-word PCM buffer and send to DSP or use PCM2WAY_FillSpk to fill one specific value to the DSP 160-word buffer.
(3) Use PCM2WAY_Stop to stop the PCM2WAY service.

8.2 PCM4WAY Functions

8.2.1 PCM4WAY_Start

void PCM4WAY_Start(void (*pcm4way_hdlr)(void));

This function is called to enable PCM4WAY service.

Parameters:

Parameter	Description
pcm4way_hdlr	Pointer to PCM4WAY registered handler which will be callbacked in L1Audio_HISR .

Return value:
None.

Example:
```
#include "pcm4way.h"
static uint16 PCM_BUF[160];
static uint16 PCM_UL_BUF[8][160];
static uint16 PCM_DL_BUF[8][160];
static uint32 pcm_buf_rw;

void pcm4way_hisrHdl()
{
  uint32 I;
  uint16 *buf1, *buf2;

  buf1 = (uint16*)PCM_UL_BUF[pcm_buf_rw];
  buf2 = (uint16*)PCM_BUF;
  for(I = 160; I > 0 ; I--)
    *buf2++ = (*buf1++) >> 2;
  PCM4WAY_GetFromMic((uint16*)PCM_UL_BUF[pcm_buf_rw]);
  buf1 = (uint16*)PCM_UL_BUF[pcm_buf_rw];
  buf2 = (uint16*)PCM_BUF;
  for(I = 160; I > 0 ; I--)
    (*buf2++) += (*buf1++) >> 2 * 3;
  PCM4WAY_PutToSE((const uint16*)PCM_BUF);
```

```
  buf1 = (uint16*)PCM_DL_BUF[pcm_buf_rw];
  buf2 = (uint16*)PCM_DL_BUF;
  for(I = 160; I > 0 ; I--)
    *buf2++ = (*buf1++) >> 2;
  PCM4WAY_GetFromSD((uint16*)PCM_DL_BUF[pcm_buf_rw]);
  buf1 = (uint16*)PCM_DL_BUF[pcm_buf_rw];
  buf2 = (uint16*)PCM_BUF;
  for(I = 160; I > 0 ; I--)
    (*buf2++) += (*buf1++) >> 2 * 3;
  PCM4WAY_PutToSpk((const uint16*)PCM_BUF);

  pcm_buf_rw ++;
  pcm_buf_rw &= 0x7;
}

void Start_Echo(void)
{
        memset( (kal_uint8 *)PCM_UL_BUF, 0x0, 2560 );
        memset( (kal_uint8 *)PCM_DL_BUF, 0x0, 2560 );
        pcm_buf_rw = 0;
        PCM4WAY_Start(pcm4way_hisrHdl);
}

void Stop_Echo(void)
{
        PCM4WAY_Stop();
}
```

8.2.2 PCM4WAY_Stop

void PCM4WAY_Stop(void);

This function is called to disable PCM4WAY service.

Parameters:
None.

Return value:
None.

Example:
See example 15.2.1.

8.2.3 PCM4WAY_GetFromMic

void PCM4WAY_GetFromMic(uint16 *ul_buf);

This function is used to get PCM data from DSP and store to some 160-word buffer. The PCM data are up-link speech data that recorded by AFE.

Parameters:

Parameter	Description
ul_buf	Pointer to 160-word buffer that stores the up-link data got from DSP

Return value:
None

Example:
See example 15.2.1

8.2.4 PCM4WAY_PutToSE

void PCM4WAY_PutToSE(const uint16 *ul_data);

This function is used to get PCM data from some 160-word buffer and send to DSP to be encoded in speech encoder for up-link speech transmission.

Parameters:

Parameter	Description
ul_data	Pointer to 160-word buffer that stores the PCM data to be sent to DSP

Return value:
None

Example:
See example 15.2.1

8.2.5 PCM4WAY_FillSE

void PCM4WAY_FillSE(uint16 value);

This function is used to fill one specific value to the DSP 160-word buffer to be encoded in speech encoder for up-link speech transmission. It would make a silence-like effect.

Parameters:

Parameter	Description
Value	The specific value to be filled in the DSP 160-word buffer

Return value:
None.

Example:

```
#include "pcm4way.h"

void pcm4way_hisrHdl()
{
```

```
    PCM4WAY_FillSE(0);
    PCM4WAY_FillSpk(0);
}

void Start_Mute(void)
{
     PCM4WAY_Start(pcm4way_hisrHdl);
}

void Stop_Mute(void)
{
     PCM4WAY_Stop();
}
```

8.2.6 PCM4WAY_GetFromSD

void PCM4WAY_GetFromSD(uint16 *dl_buf);

This function is used to get PCM data from DSP and store to some 160-word buffer. The PCM data are down-link speech data from the DSP speech decoder.

Parameters:

Parameter	Description
dl_buf	Pointer to 160-word buffer that stores the down-link data got from DSP

Return value:
None.

Example:
See example 15.2.1

8.2.7 PCM4WAY_PutToSpk

void PCM4WAY_PutToSpk(const uint16 *dl_data);

This function is used to get PCM data from some 160-word buffer and send to DSP to be played in the AFE.

Parameters:

Parameter	Description
dl_data	Pointer to 160-word buffer that stores the PCM data to be sent to DSP

Return value:
None.

Example:
See example 15.2.1

8.2.8 PCM4WAY_FillSpk

void PCM4WAY_FillSpk(uint16 value);

This function is used to fill one specific value to the DSP 160-word buffer to be played in the AFE. It would make a silence-like effect.

Parameters:

Parameter	Description
Value	The specific value to be filled in the DSP 160-word buffer

Return value:
None.

Example:
See example 15.2.5

8.3 PCM2WAY Functions

8.3.1 PCM2WAY_Start

void PCM2WAY_Start(void (*pcm2way_hdlr)(void));

This function is called to enable PCM2WAY service.

Parameters:

Parameter	Description
pcm2way_hdlr	Pointer to PCM2WAY registered handler which will be callbacked in L1Audio_HISR .

Return value:
None.

Example:

```
#include "pcm4way.h"
static uint16 PCM_BUF[160];
static uint16 PCM_UL_BUF[8][160];
static uint32 pcm_buf_rw;
typedef enum {
  NORMAL_LOOPBACK,
  ECHO_LOOPBACK,
  MUTE_SPK
} Test_Func;
static Test_Func test_func;

void pcm2way_hisrHdl()
{
```

```
  uint32 I;
  uint16 *buf1, *buf2;

  buf1 = (uint16*)PCM_UL_BUF[pcm_buf_rw];
  buf2 = (uint16*)PCM_BUF;
  for(I = 160; I > 0 ; I--)
    *buf2++ = (*buf1++) >> 2;
  PCM2WAY_GetFromMic((uint16*)PCM_UL_BUF[pcm_buf_rw]);

  If(test_func == NORMAL_LOOPBACK)
     PCM2WAY_PutToSpk((uint16*)PCM_UL_BUF[pcm_buf_rw]);
  else if(test_func == ECHO_LOOPBACK)
  {
     buf1 = (uint16*)PCM_UL_BUF[pcm_buf_rw];
     buf2 = (uint16*)PCM_BUF;
     for(I = 160; I > 0 ; I--)
        (*buf2++) += (*buf1++) >> 2 * 3;
     PCM2WAY_PutToSpk((const uint16*)PCM_BUF);
  }
  else if(test_func == MUTE_SPK)
     PCM2WAY_FillSpk( 0);
  pcm_buf_rw ++;
  pcm_buf_rw &= 0x7;
}

void  Start_Test(void)
{
    memset( (kal_uint8 *)PCM_UL_BUF, 0x0, 2560 );
    pcm_buf_rw = 0;
    test_func = NORMAL_LOOPBACK;
    PCM2WAY_Start(pcm2way_hisrHdl);
}

void  Stop_Test(void)
{
   PCM2WAY_Stop();
}

void Change_Test_Func(Test_Func new_test)
{
    uint32 savedMask;
    ASSERT(new_test == NORMAL_LOOPBACK || new_test == ECHO_LOOPBACK || new_test == MUTE_SPK);
    savedMask = SaveAndSetIRQMask();
    test_func = new_test;
    RestoreIRQMask( savedMask );
}
```

8.3.2 PCM2WAY_Stop

void PCM2WAY_Stop(void);

This function is called to disable PCM2WAY service.

Parameters:
None.

Return value:
None.

Example:
See example 15.3.1.

8.3.3 PCM2WAY_GetFromMic

void PCM2WAY_GetFromMic(uint16 *ul_buf);

This function is used to get PCM data from DSP and store to some 160-word buffer. The PCM data are recorded by AFE.

Parameters:

Parameter	Description
ul_buf	Pointer to 160-word buffer that stores the data got from DSP

Return value:
None.

Example:
See example 15.3.1

8.3.4 PCM2WAY_PutToSpk

void PCM2WAY_PutToSpk(const uint16 *dl_data);

This function is used to get PCM data from some 160-word buffer and send to DSP to be played in the AFE.

Parameters:

Parameter	Description
dl_data	Pointer to 160-word buffer that stores the PCM data to be sent to DSP

Return value:
None

Example:
See example 15.3.1

8.3.5 PCM2WAY_FillSpk

void PCM2WAY_FillSpk(uint16 value);

This function is used to fill one specific value to the DSP 160-word buffer to be played in the AFE. It would make a silence-like effect.

Parameters:

Parameter	Description
Value	The specific value to be filled in the DSP 160-word buffer

Return value:
None.

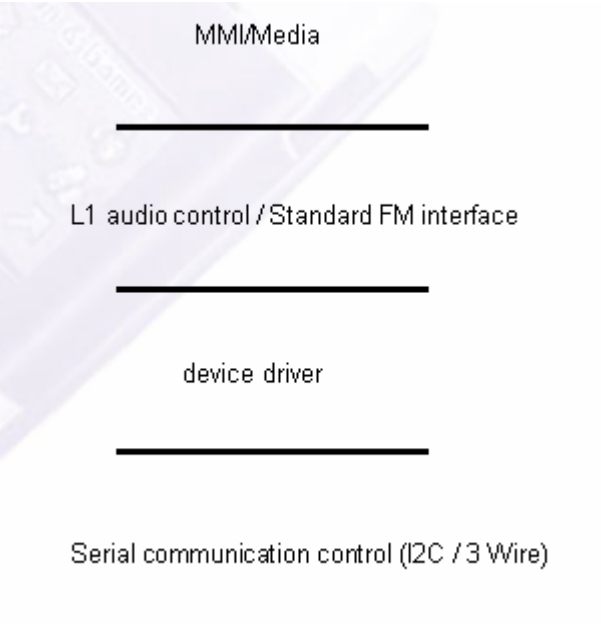
Example:
See example 15.3.1

9 FM Radio driver interface

The TEA5767HN is a single-chip electronically tuned FM stereo radio for low-voltage application with fully integrated IF selectivity and demodulation. This chapter is mainly to describe a set of function that could be used to control TEA5767HN directly.

9.1 FM driver partition

The FM driver partition is shown as follows.



(1) L1 audio control: including AFE control, key-tone disabled control and L1 FM radio interfaces.
(2) Device driver: Device-specific code to implement FM radio interface on specific FM chip.
(3) Serial communication control: GPIO control (I2C/3-wire protocol)
We introduce standard FM interface next section.

9.2 Functions

9.2.1 FMR_PowerOn

void FMR_PowerOn(void)

Before you operate FM Radio module, this function should be called firstly to configure AFE and reset FM chip.

Parameters:

Parameter	Description
none	None

Return value:
None

Example:

```
    FMR_PowerOn( );      /* Turn on FM Radio function */
    FMR_SetFreq(907);   /* Set tuned frequency to 90.7MHz */
…
    FMR_PowerOff( )      /* Turn off FM Radio function */
```

9.2.2 FMR_PowerOff

void FMR_PowerOff(void)

This function could be used to turn off FM Radio module.

Parameters:

Parameter	Description
none	none

Return value:
None

Example:
See the example of FMR_PowerOn

9.2.3 FMR_SetFreq

void FMR_SetFreq(int16 curf)

This function could be used to set wanted frequency. It will convert current tuned frequency to PLL word and set it in PLL tuning system.

Parameters:

Parameter	Description
Curf	The tuned frequency. The range is limited from 875 to 1080.

Return value:
None

Example:
See the example of FMR_PowerOn

9.2.4 FMR_GetSignalLevel

uint8 FMR_GetSignalLevel(int16 curf)

Get the RF input level for a specific FM station.

Parameters:

Parameter	Description
Curf	Wanted frequency: 875 - 1080 (87.5 MHz - 108.0 MHz)

Return value:
RF input level, signal level range is 0 ~ 15

Example:

```
uint8 signalLevel
…
signalLevel = FMR_GetSignalLevel(1001);
```

9.2.5 FMR_ValidStop

uint8 FMR_ValidStop(int16 freq, int8 signalvl, bool is_step_up)

This function will perform Valid Stop Check algorithm to confirm if this is a proper station.

Parameters:

Parameter	Description
freq	Frequency, the range is from 875 to 1080
signalvl	Stop signal level, the range is from 0 to 15
Is_step_up	Search direction, 1: up, 0: down

Return value:
Return true if the current frequency is a valid station, return false otherwise.

Example:

```
#define SEARCH_UP   1
Int16 frequency,i;
Int8 level;
frequency = 875;
level = 5;
    FMR_PowerOn( );     /* Turn on FM Radio function */

    For(I=0;I<206;I++){
      if(FMR_ValidStop(frequency+i, level, SEARCH_UP)){
        FMR_SetFreq(frequency+i);
        break;
       }
    }
    …
    …
    …
    FMR_PowerOff( )     /* Turn off FM Radio function */
```

9.2.6 FMR_Mute

void FMR_Mute(uint8 mute)

Mute the FM input.

Parameters:

Parameter	Description
mute	1: mute, 0: un-mute

Return value:
None

Example:
FMR_Mute(1);

9.2.7 FMR_SetOutputDevice

void FMR_SetOutputDevice(uint8 device)

Set output device

Parameters:

Parameter	Description
device	Output device

Return value:
None

Example:
FMR_SetOutputDevice(L1SP_SPEAKER_ST);

9.2.8 FMR_SetOutputVolume

void FMR_SetOutputVolume(uint8 volume1, uint8 volume2)

Set output volume

Parameters:

Parameter	Description
Volume1	The gain of the output buffer 0 (0-255, 2dB per 16 steps).
Voulme2	The gain of the output buffer 1 (0-255, 2dB per 16 steps).

Return value:
None

Example:
FMR_SetOutputVolume(160, 160);

9.2.9 FMR_IsChipValid

bool FMR_IsChipValid(void)

FMR_IsChipValid could be called to confirm if FM chip dose exist.

Parameters:

Parameter	Description
none	

Return value:
Return true if FM chip is valid, return false otherwise.

Example:

```
If(FMR_IsChipValid())
{
  …
  //    Enable FM radio menu
  …
}
```

9.2.10 FMR_IsActive

bool FMR_IsActive(void)

FMR_IsActive could be called to confirm whether FM radio is active.

Parameters:

Parameter	Description
None	

Return value:
Return true if FM ratio is active, return false otherwise.

Example:

```
If(FMR_IsActive())
{
  …
  //    … do something
  …
}
```

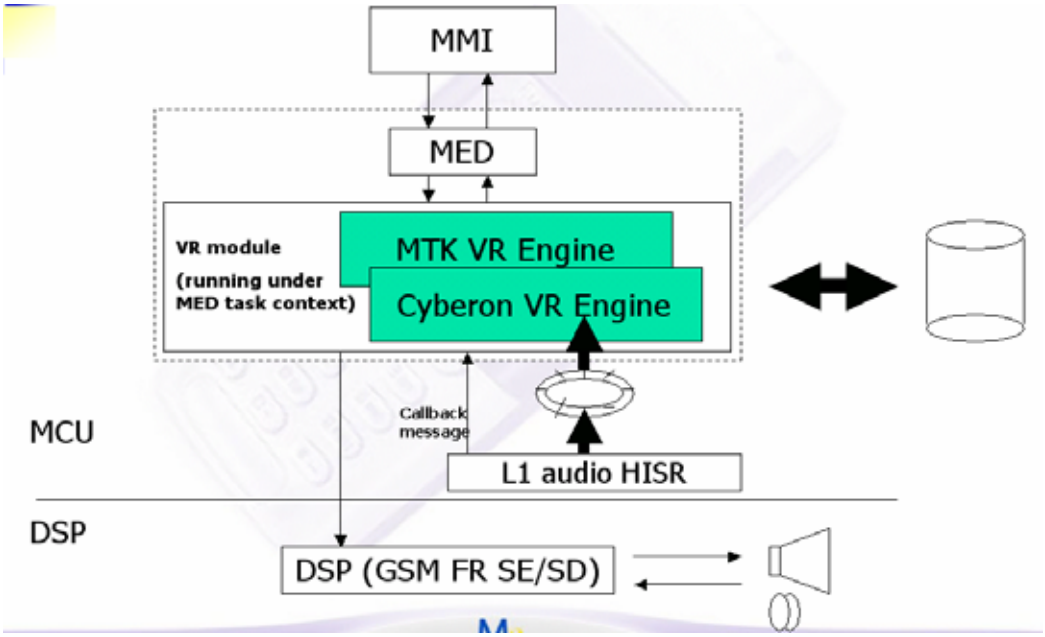
10 SD VR interface

L1 audio provides a serious interface for speaker dependent Voice Recognition.

10.1 Concept

10.1.1 SD VR architecture

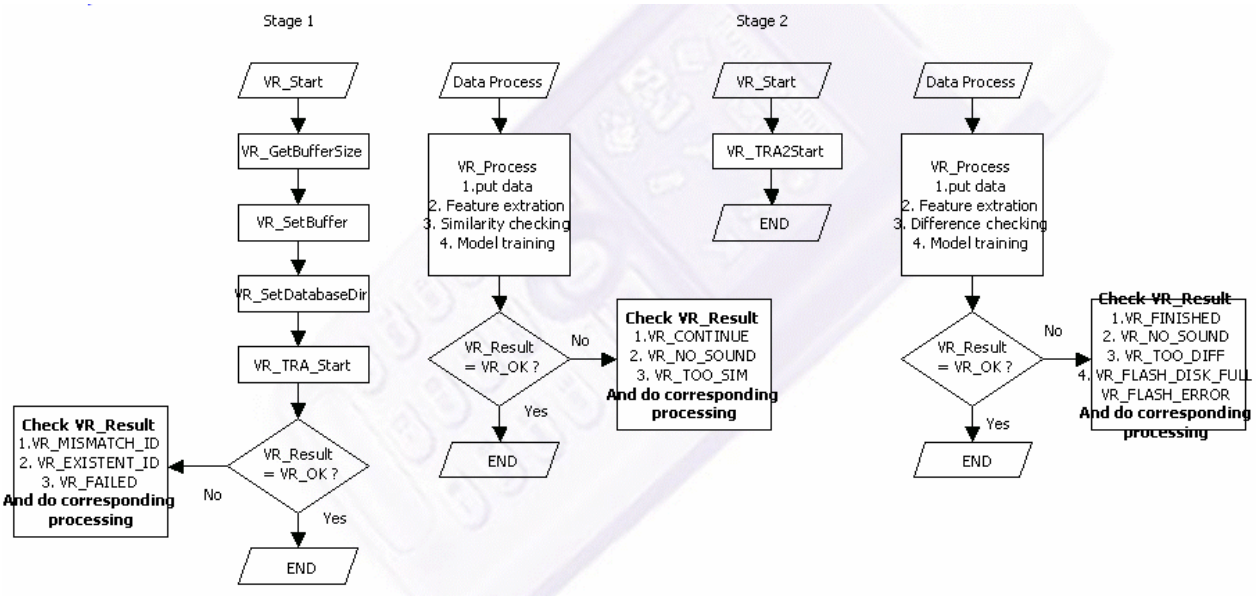
The method to perform the VR task of what the ways enable speech input propagation, VR process and flash access and how they are likely to be used is to be depicted in the following plot. The VR related routines like model training, recognition and flash access are all arranged on MED task context. MED task should perform all action via interfaces and handle callback message with VR_Process API. Whenever DSP issued an interrupt to MCU (per 20ms), L1 audio HISR will send a message to MED task via callback function. And, MED task will continue VR process (call VR_Process interface) that gets data from ring buffer to VR engine to process. During VR kernel process, VR engine will access flash automatically to read/write model and voice data.



10.1.2 VR flowchart

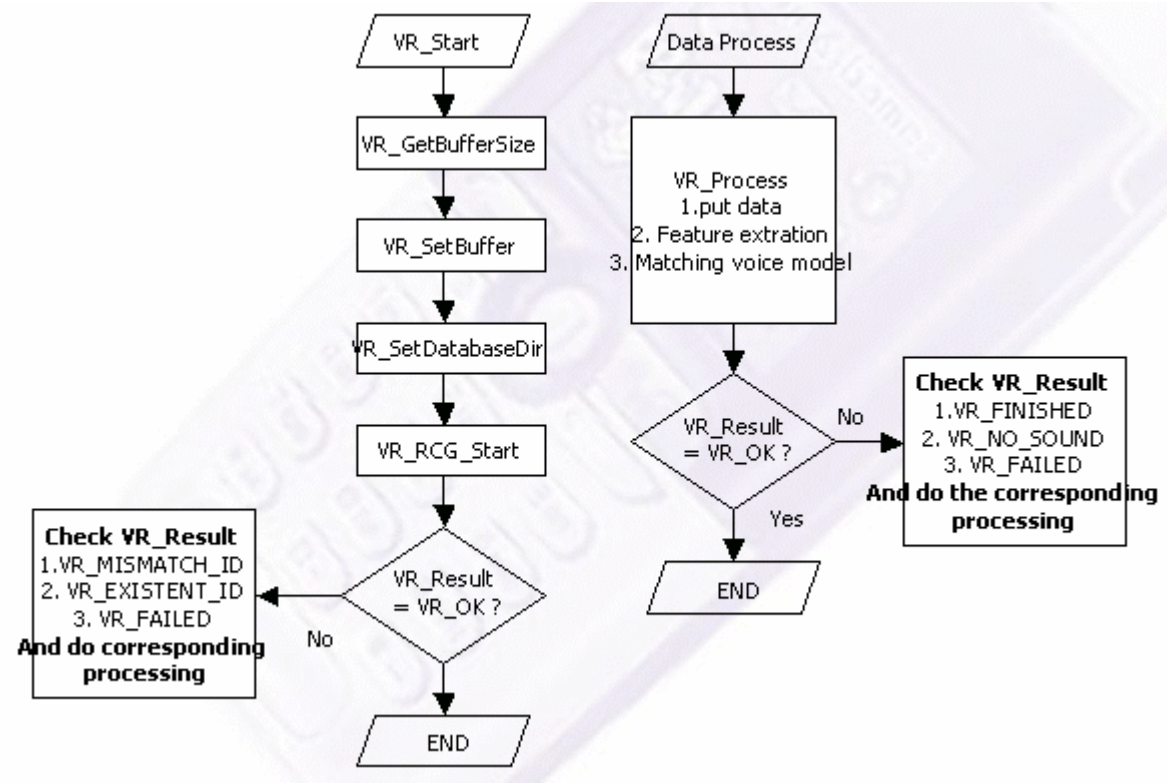
10.1.2.1 Training

The following figures show what are legal calling sequences completing the training process.



10.1.2.2 Recognition

The following figures show what are legal calling sequences completing the recognition process.



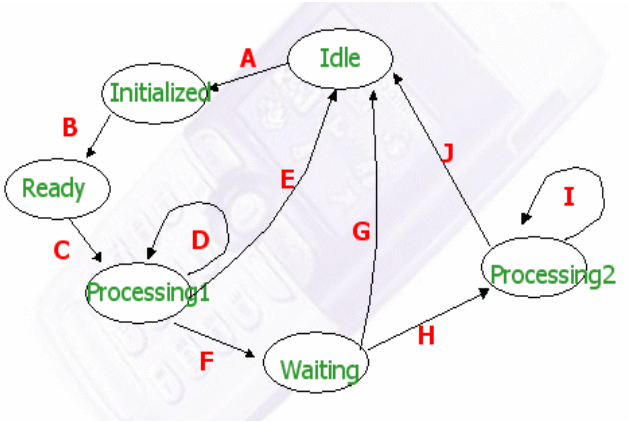
10.1.3 VR state diagram

State definition:

VR_STATE_IDLE	0x0
VR_STATE_T_INITIALIZED	0x1
VR_STATE_T_READY	0x2
VR_STATE_T_PROCESSING1	0x3
VR_STATE_T_WAITING	0x4
VR_STATE_T_PROCESSING2	0x5
VR_STATE_R_INITIATED	0x81
VR_STATE_R_READY	0x82
VR_STATE_R_PROCESSING	0x83

10.1.3.1 Training

The following figure is training state diagram.



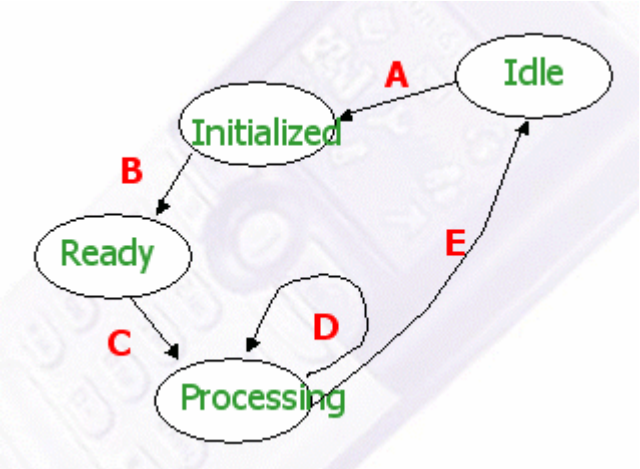
VR task status		Description
VR_STATE_IDLE	0x0	Idle state
VR_STATE_T_INITIALIZED	0x1	Initialized
VR_STATE_T_READY	0x2	Ready to get data from input buffer
VR_STATE_T_PROCESSING1	0x3	1st stage of training process session
VR_STATE_T_WAITING	0x4	Waiting state.
VR_STATE_PROCESSING2	0x5	2nd stage of training process session.

Action	Description
A	Get needed run-time memory size and set run-time memory. VR_GetBufferSize() is called to inquire needed buffer size. And, VR_SetBuffer() is called to set buffer for VR engine.
B	Set Database directory. (VR_SetDatabaseDir() is called)
C	VR_TRA_Start() is called to initialize the VR engine and register a space for training a speech. And, GSM-FR recording is turned on.
D	VR_Porcess() is called via callback message. 1. Get speech for the use of extracting feature. 2. Process the captured speech data (similarity checking and model training).
E	DSP is turned off and VR engine is reset.

	Possible case: 1. User abort (VR_Stop is called) 2. Processing failed. (Return VR_NO_SOUND / VR_TOO_SIM / VR_MISMATCH_ID/VR_EXISTENT_ID)
F	DSP is turned off. Need to process one more time (Return VR_CONTINUE)
G	User Abort (VR_Stop is called)
H	1. DSP is turned on and VR engine will register a space for training a speech. (VR_TRA2Start() is called)
I	(VR_Process() is called via callback message) 1. Get speech for the use of extracting feature. 2. Process the captured speech data (difference checking and model training).
J	DSP is turned off and VR engine is reset. Possible case: 1. User abort (VR_Stop() is called) 2. Processing finished. (Return VR_FINISHED) 3. Processing failed. (Return VR_TOO_DIFF/ VR_NO_SOUND / VR_FLASH_DISK_FULL / VR_FLASH_ERROR)

10.1.3.2 Recognition

The following figure is recognition state diagram.



VR task status		Description
VR_STATE_IDLE	0x0	Idle state
VR_STATE_R_INITIALIZED	0x81	Initialized
VR_STATE_R_READY	0x82	Ready to get data from input buffer
VR_STATE_R_PROCESSING	0x83	Recognition process session

Action	Description
A	Get needed run-time memory size and set run-time memory. VR_GetBufferSize() is called to inquire needed buffer size. And, VR_SetBuffer() is called to set buffer for VR engine.
B	Set Database directory.

	(VR_SetDatabaseDir() is called)
C	VR_RCG_Start() is called to initialize the VR engine and register a space for recognizing a speech. And, GSM-FR recording is turned on.
D	VR_Porcess() is called via callback message. 1. Get speech for the use of extracting feature. 2. It matches the best voice model trained.
E	DSP is turned off and VR engine is reset. Possible case: 1. User abort (VR_Stop() is called) 2. Processing finished. (Return VR_FINISHED) 3. Processing failed. (Return VR_FAILED / VR_NO_SOUND / VR_MISMATCH_ID / VR_EXISTENT_ID / VR_SDMODEL_NOT_READY)

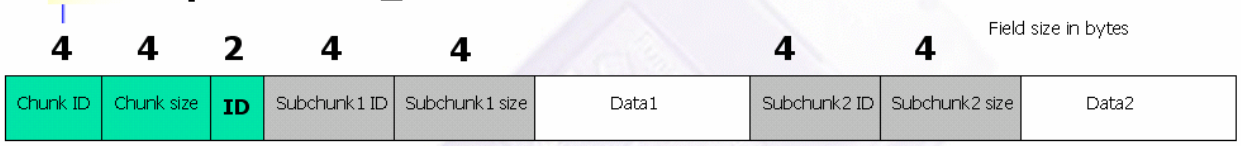
10.1.4 VR file format

When VR training is successful and a voice tag is well feature extraction succeeds, the results is saved automatically. The VR file format is shown as follows:

This format structure is used to defined all VR file data, and is described as following table in detail.
All defined VR file must contain a main chunk and two sub chunks.

The file is named as “MTKVR_XXXX.vrd”. (XXXX is the ID number that is represented as hexadecimal number in ASCII form)
For instance, the file name is “MTKVR_0014.vrd” and ID of this file is 20.

Example: MTKVR_0001.vrd



Chunk ID	The “MTKV”(for MTK) or “CYBV” (for Cyberon) chunk descriptor. Contains the letters “MTKV" or “CYBV” in ASCII form (0x4D544B56 or 0x43594256 big-endian form).
Chunk size	The size in bytes of the all information in the VR file not including the size of the Chunk ID and size (8 bytes, little-endian form).
ID	2 bytes, Voice tag ID , range: 0 ~ 65535.
Sub-chunk1 ID	The “VOCE” sub-chunk1 descriptor Contains the letters “VOCE" in ASCII form (0x564F4345 big-endian form).
Sub-chunk1 size	Voice data size in bytes
Sub-chunk2 ID	The “MODL” sub-chunk2 descriptor Contains the letters “MODL" in ASCII form (0x4D4F444C big-endian form).
Sub-chunk2 size	Model size in bytes
Data1/Data2	Voice data (GSM-FR) / HMM model or pattern

10.2 Type Definition

Define:

```
typedef enum
{
        VR_FINISHED,
        VR_OK,
        VR_FAILED,
        VR_CONTINUE,
        VR_NO_SOUND,
        VR_TOO_SIM,
        VR_TOO_DIFF,
        VR_SDMODEL_NOT_READY,
        VR_MISMATCH_ID,
        VR_EXISTENT_ID,
        VR_FLASH_DISK_FULL,
        VR_FLASH_ERROR
} VR_Result;
```

This enumeration defines the return type of VR interfaces.

Member	Description
VR_FINISHED	VR task is complete and successful.
VR_OK	If the function succeeds, it returns VR_OK. For interface VR_Process(), the VR processing is not complete yet.
VR_FAILED	If the function fails, the return value is VR_FAILED.
VR_CONTINUE	Need to call VR_TRA2Start() to start 2nd stage of training.
VR_NO_SOUND	Only for training, no sound is heard.
VR_TOO_SIM	Only for training, some id in context is too similar with the current voice.
VR_TOO_DIFF	Only for training, voices are too different.
VR_SDMODEL_NOT_READY	Only for recognition, SD model is not trained ready.
VR_MISMATCH_ID	Name list IDs mismatch with database is detected.
VR_EXISTENT_ID	The trained ID number is equal to any entry in name list.
VR_FLASH_DISK_FULL	Not enough contiguous clusters are found in file system.
VR_FLASH_ERROR	Other flash access error.

10.3 Functions

10.3.1 VR_GetParameters

VR_Result VR_GetParameters(int16 *SimThrld, int16 *DiffThrld, int16 *RejThrld)

Get the current VR parameters setting.

Parameters:

Parameter	Description
SimThrld	Similarity checking threshold for VR training. SimThrld should be in range of {-5000, 3000}, and default value is 0. The lower SimThrld, the harder similarity check could pass.

DiffThrld	Difference checking threshold for VR training. DiffThrld should be in range of {-3000, 3000}, and default value is 0. The higher DiffThrld, the harder difference check could pass.
RejThrld	Rejection threshold for VR recognition. RejThrld should be in range of {-3000, 3000}, and default value is 0. The higher RejThrld, the harder rejection test could pass.

Return value:
If the operation is complete, return VR_OK.
Otherwise, return VR_FAILED.

Example:
Int16 SimThrld, DiffThrld, RejThrld;

VR_GetParameters(&SimThrld, &DiffThrld, &RejThrld);
…
VR_SetParemeters (100, 100, 100);

10.3.2 VR_SetParameters

VR_Result VR_SetParemeters (int16 SimThrld, int16 DiffThrld, int16 RejThrld)

Set the current VR parameters setting.

Parameters:
Refer to VR_GetParameters.

Return value:
If the operation is complete, return VR_OK.
Otherwise, return VR_FAILED.

Example:
See example of VR_GetParameters.

10.3.3 VR_SetDatabaseDir

Void VR_SetDatabaseDir(uint8 *Dir)

Set database directory for flash access.
*Please **do not** assign the same directory for different ID groups.*

Parameters:

Parameter	Description
Dir	Pointer to the database directory for flash access.

Return value:
None.

Example:

uint8 DatabaseDir[] = "c:\\audio\\VR_NameList1";

```
uint8 *WorkingSpace;
uint32 buf_size;

buf_size = VR_GetBufferSize(void);
…
/* Allocate a memory in size of buf_size */
…
VR_SetBuffer(WorkingSpace);
VR_SetDatabaseDir(DatabaseDir);
```

10.3.4 VR_GetBufferSize

uint32 VR_GetBufferSize(void)

This function is to inquire the needed run-time memory size for VR engine.

Parameters:
None.

Return value:
Return run-time memory needed size in byte.

Example:
See example of VR_SetDatabaseDir.

10.3.5 VR_SetBuffer

Void VR_SetBuffer(uint8 *RTmemory)

Assign a run-time memory for VR task.

Parameters:

Parameter	Description
RTmemory	Pointer to the run-time memory to be used.

Return value:
None.

Example:
See example of VR_SetDatabaseDir.

10.3.6 VR_TRA_Start

VR_Result VR_TRA_Start(uint16 wID, uint16 *mdIDList, uint16 *mdIDListLen,
void (*vr_handler)(void *parameter))

Start VR training with assigned ID.

1. Initialize VR engine for VR training.

2. Check name list validation. If mismatch, return correct name list.
3. Turn on DSP.
4. Register a space for training a speech.
5. Get speech for the use of extracting feature.

Parameters:

Parameter	Description
WID	Current voice ID to be saved.
MdIDList	Pointer to name list to be returned.
MdIDListLen	Pointer to the length of name list to be returned.
vr_handler	A callback function for handling VR process (send a message to activate MED task).

Return value:
If the operation is complete, return VR_OK.
If the name list IDs mismatch with database, return VR_MISMATCH_ID.
If the parameter wID is equal to any entry in name list, return VR_EXISTENT_ID.
Otherwise, return VR_FAILED.
Example:
See example of VR_Process.

10.3.7 VR_TRA2Start

VR_Result VR_TRA2Start(void)

Start 2nd stage of VR training.

Parameters:
None.

Return value:
If the operation is complete, return VR_OK.
Otherwise, return VR_FAILED.

Example:
See example of VR_Process.

10.3.8 VR_RCG_Start

VR_Result VR_RCG_Start(uint16 *mdIDList, uint16 *mdIDListLen,
void (*vr_handler)(void *parameter))

Start VR recognition with assigned ID.

1. Initialize VR engine for VR recognition.
2. Check name list validation. If mismatch, return correct name list.
3. Turn on DSP.
4. Register a space for training a speech.

5. Get speech for the use of extracting feature.

Parameters:

Parameter	Description
MdIDList	Pointer to name list to be returned if needed.
MdIDListLen	Pointer to the length of name list to be returned if needed.
vr_handler	A callback function for handling VR process (send a message to activate MED task).

Return value:

If the operation is complete, return VR_OK.

If the name list IDs mismatch with database, return VR_MISMATCH_ID,

Otherwise, return VR_FAILED.

Example:

See example of VR_Process.

10.3.9 VR_Process

VR_Result VR_Process(uint16 *RcgID)

This function gets speech frame for the usage of extracting feature. In training session, it will process the captured speech data like similarity checking, difference checking and model training. In recognition session, it will process the captured speech data and match the best voice model trained
This function should be triggered by callback function of VR_TRA_Start() or VR_RCG_Start().

Parameters:

Parameter	Description
RcgID	Pointer to the recognition result to be returned.

Return value:

Member	Description
VR_OK	If the function succeeds, it returns VR_OK.
VR_FAILED	It the function fails, it returns VR_FAILED.
VR_FINISHED	If the VR training/recognition finished, it returns VR_FINISHED.
VR_CONTINUE	Need to start second stage of VR training. (call VR_TRA2Start())
VR_NO_SOUND	No sound is heard when VR training/recognition
VR_TOO_SIM	Some id in context is too similar with the current voice when VR training
VR_TOO_DIFF	Voices are too different when VR training
VR_SDMODEL_NOT_READY	SD model is not trained ready when VR recognition
VR_FLASH_DISK_FULL	Not enough contiguous clusters are found in file system when VR training.
VR_FLASH_ERROR	Other flash access error when VR training.

Example:

```
/* VR training */
Uint16 namelist[20];
Uint16 namelistLen;
uint8 DatabaseDir[] = "c:\\audio\\VR";
uint8 *WorkingSpace;
uint32 buf_size;
```

```
VR_Result vr_result;
uint 16 RcgID;
uint16 wID = 1;
…
…
buf_size = VR_GetBufferSize(void);
…
/* Allocate a memory in size of buf_size */
…
VR_SetBuffer(WorkingSpace);
VR_SetDatabaseDir(DatabaseDir);

vr_result = VR_TRA_Start(wID, namelist, &namelistLen,  Callback((void*)0));
if(vr_result is equal to VR_FAILED/ VR_MISMATCH_ID/ VR_EXISTENT_ID) … /* return error message */…

…

/* Callback message received */
Switch(message){
…
   case VR_Process_MSG:

            switch (VR_Process(&RcgID)){
                  case  VR_OK:
                        /* do nothing */
                       break;
                  case  VR_NO_SOUND:
                        /* return error message */
                       break;
                  case  VR_TOO_SIM:
                        /* return error message */
                        break;
                  case  VR_TOO_DIFF:
                        /* return error message */
                        break;
                  case  VR_FLASH_DISK_FULL:
                        /* return error message */
                        break;
                  case  VR_FLASH_ERROR:
                        /* return error message*/
                       break;
                  case  VR_FINISHED:
                        /* VR training is complete */
                        …
                        /* Example: A beep now confirms the recording, which is saved automatically. */
                        break;
                  case  VR_CONTINUE:
                          vr_result = VR_TRA2Start();
                           if(vr_result == VR_FAILED) … … ; /* return error message */
                           break;
                   default:
                          ASSERT(false);
            }
            break;
…
}
```

Example:

```
/* VR recognition */
…
…
vr_result = VR_RCG_Start(namelist, &namelistLen,  Callback((void*)0));
if(vr_result is equal to VR_FAILED/ VR_MISMATCH_ID) … /* return error message */…
…
…
…
/* Callback message received */
Switch(message){
…
   case VR_Process_MSG:

            switch (VR_Process(&RcgID)){
                  case  VR_OK:
                         /* do nothing */
                        break;
                  case  VR_FAILED:
                         /* return error message */
                        break;
                  case  VR_NO_SOUND:
                         /* return error message */
                        break;
                  case  VR_FINISHED:
                         /* VR recognition is complete */
                         …
                         …
                        /* example: Voice tag playback */
                        /* The number is dialed. (Make MT call) */
                         break;
                   default:
                          ASSERT(false);
            }
            break;
…
}
```

10.3.10 VR_Stop

void VR_Stop(void)

VR_Stop will stop and abort VR task immediately.

Parameters:

None.

Return value:

None.

Example:

```
…
VR_Stop();
…
```

11 SI VR Interface

For speaker independent voice recognition, L1Audio provides a VRSI engine to implement the functionalities. VRSI engine is an integrated speaker independent voice recognition engine. It includes the following functions:

<1> Adding tags: register texts to automatically generate recognition patterns
<2> Training tags: generate recognition patterns by user's speech
<3> Voice command recognition: recognition under grammar
<4> Digit recognition: recognize only digits' patterns
<5> Digit adaptation: improve digits' recognition patterns by user's speech
<6> the Playback of tag: play existed recognition patterns
<7> TTS playback: register texts to generate audio output data

11.1 Concept

11.1.1 VRSI Architecture

VRSI engine is supposed to run in Media task. Media Task implements a SI VR task to communicate message protocol with MMI and passes relative jobs to VRSI engine by calling VRSI APIs. The SI VR task should register a callback handler to VRSI engine to let VRSI engine send events during process. Relative audio activities are implemented by L1Audio functions and L1Audio HISR.

The architecture of VRSI engine could be shown as below:

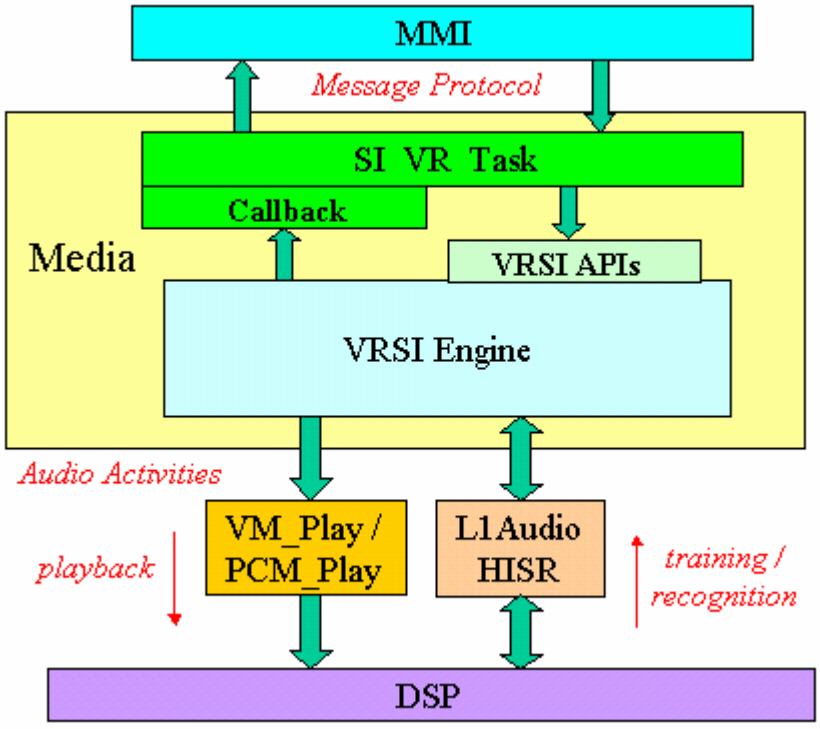


Figure 12 VRSI Engine Architecture

11.1.2 VRSI State Diagram

Since VRSI engine could implement several functions, its overall state diagram is complicated. For explanation, we will divide the overall state diagram into several functional state diagrams.

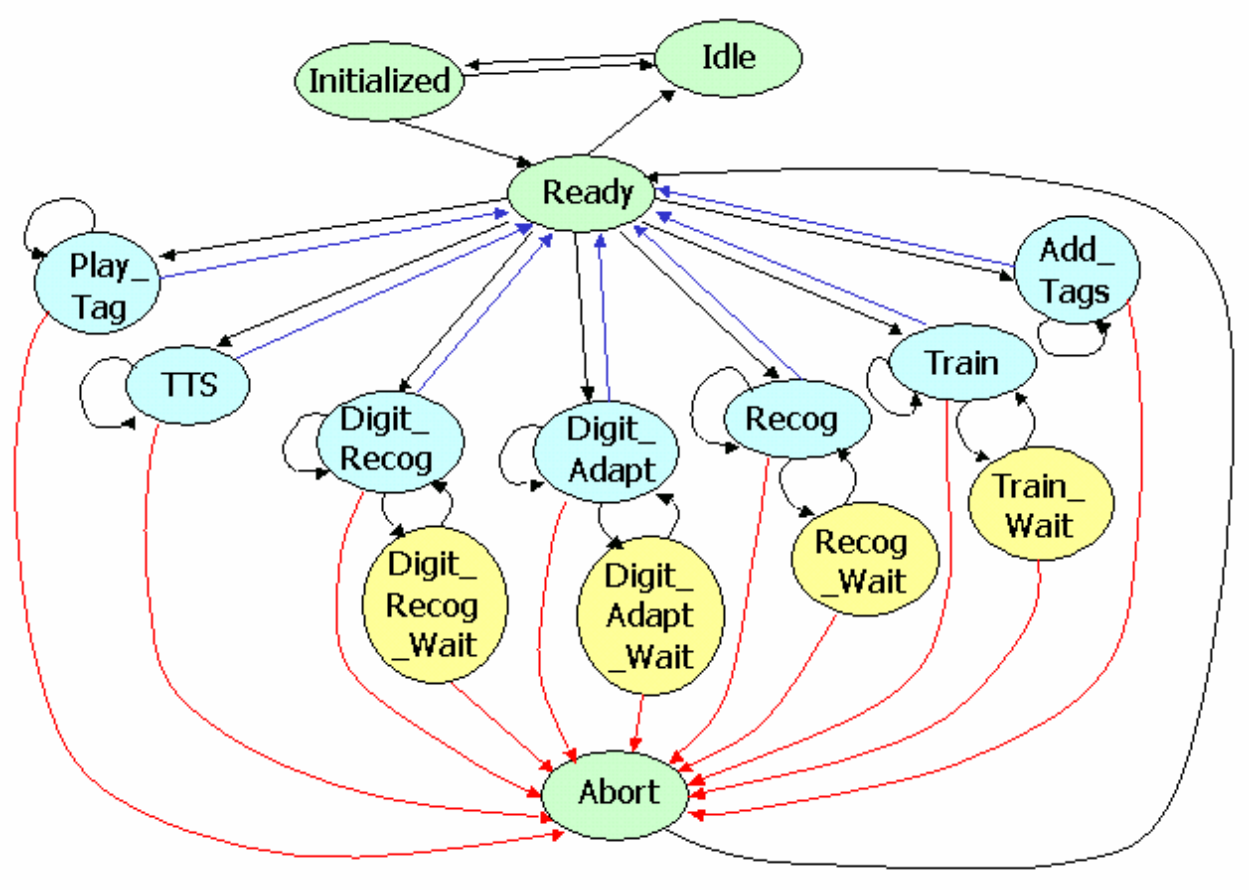
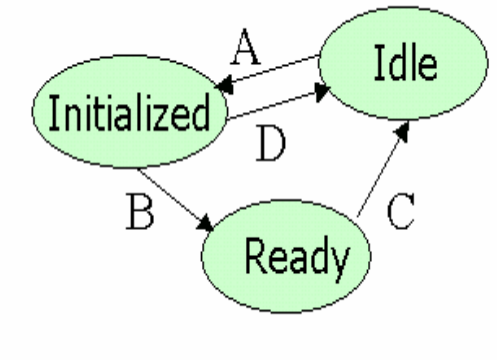


Figure 13 VRSI Engine State Diagram

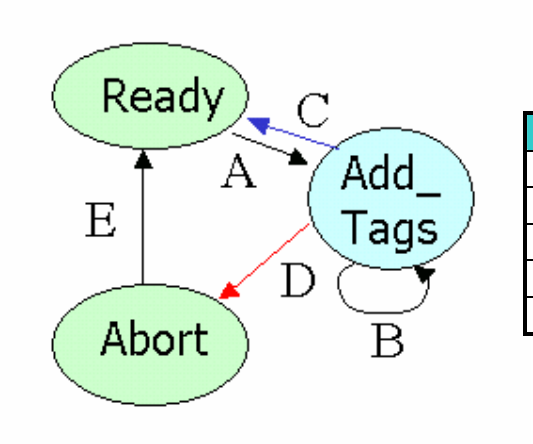
11.1.2.1 Initialization & Termination



Action	Description
A	Call VRSI_Init
B	Call VRSI_SetMem to set the working memory with size of VRSI_GetMemRequest
C , D	Call VRSI_Close

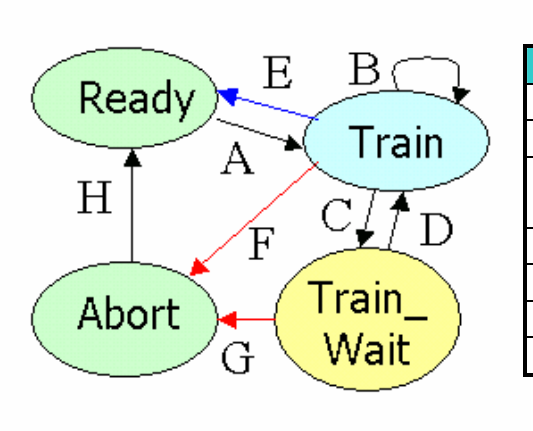
Before any VRSI function is executed, the VRSI engine must be initialized to the ready state through action A and B. For termination of VRSI engine, it must be in ready state or initialized state and terminated by calling VRSI_Close API.

11.1.2.2 Adding Tags



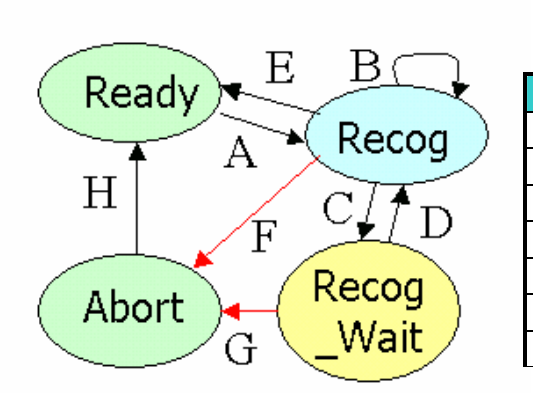
Action	Description
A	Call VRSI_Add_Tags
B	VRSI_PROCESS event
C	VRSI_ADD_TAGS_OK or VRSI_ERROR event
D	Call VRSI_Stop
E	VRSI engine finish the stop action

11.1.2.3 Training A Tag



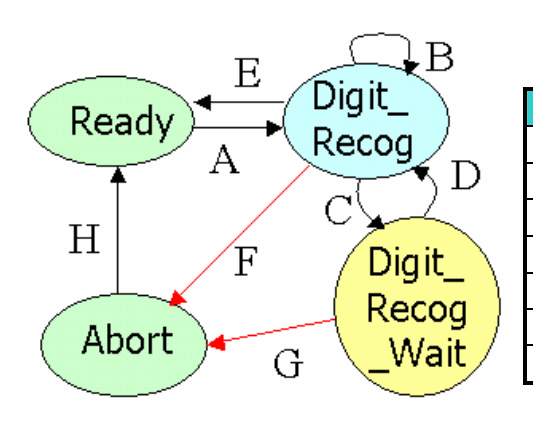
Action	Description
A	Call VRSI_Train_Tag
B	VRSI_PROCESS event
C	VRSI_TRAIN_TAG_1ST_MPR or VRSI_TRAIN_TAG_2ND_MPR event
D	Call VRSI_MMI_Confirmed
E	VRSI_TRAIN_TAG_OK or VRSI_ERROR event
F, G	Call VRSI_Stop
H	VRSI engine finish the stop action

11.1.2.4 Voice Command Recognition



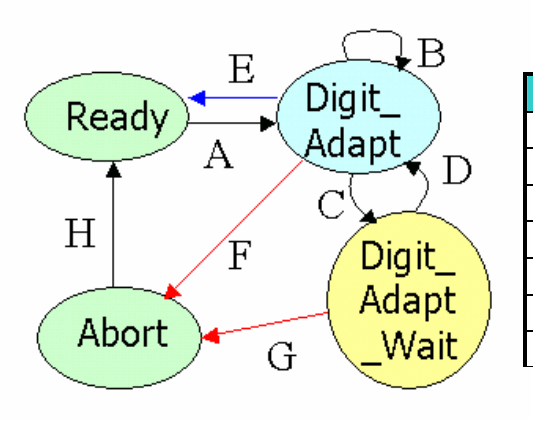
Action	Description
A	Call VRSI_Recog
B	VRSI_PROCESS event
C	VRSI_RECOG_MPR event
D	Call VRSI_MMI_Confirmed
E	VRSI_RECOG_OK or VRSI_ERROR event
F, G	Call VRSI_Stop
H	VRSI engine finish the stop action

11.1.2.5 Digit Recognition



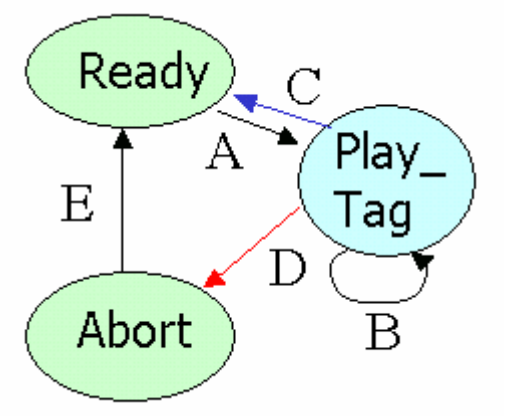
Action	Description
A	Call VRSI_Digit_Recog
B	VRSI_PROCESS event
C	VRSI_DIGIT_RECOG_MPR event
D	Call VRSI_MMI_Confirmed
E	VRSI_DIGIT_RECOG_OK or VRSI_ERROR event
F, G	Call VRSI_Stop
H	VRSI engine finish the stop action

11.1.2.6 Digit Adaptation



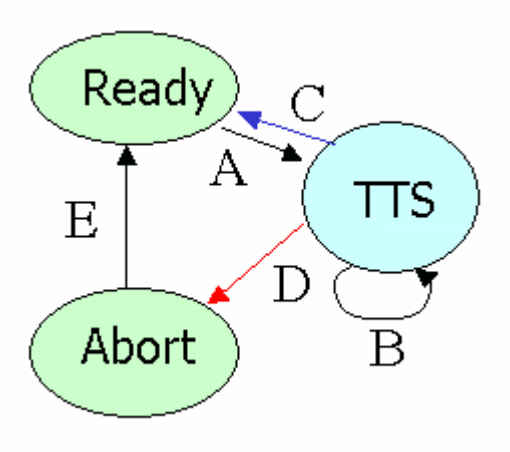
Action	Description
A	Call VRSI_Digit_Adapt
B	VRSI_PROCESS event
C	VRSI_DIGIT_ADAPT_MPR event
D	Call VRSI_MMI_Confirmed
E	VRSI_DIGIT_ADAPT_OK or VRSI_ERROR event
F, G	Call VRSI_Stop
H	VRSI engine finish the stop action

11.1.2.7 Playing A Tag



Action	Description
A	Call VRSI_Play_Tag
B	VRSI_PROCESS event
C	VRSI_PLAY_TAG_OK or VRSI_ERROR event
D	Call VRSI_Stop
E	VRSI engine finish the stop action

11.1.2.8 Playing TTS



Action	Description
A	Call VRSI_TTS_Play
B	VRSI_PROCESS event
C	VRSI_TTS_OK or VRSI_ERROR event
D	Call VRSI_Stop
E	VRSI engine finish the stop action

11.1.3 Tag Attributes

We mean a voice command pattern by a "tag". A tag has the following 4 attributes:

<1> TagName: the UNICODE symbolic representation

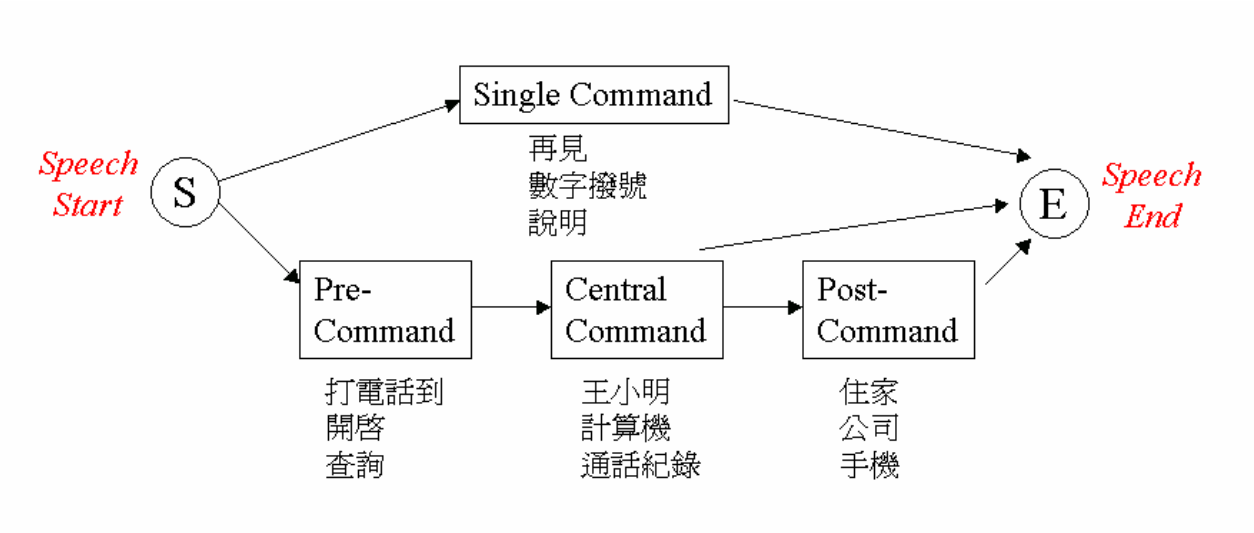
<2> TagLong: a value assigned by developer when designing grammars or by application when dynamically adding tags

<3> TagID: a unique value assigned by VRSI engine

<4> IsPeopleName: a flag indicating a people's name or not

11.1.4 Grammar

Grammar is the limitation of effective recognition patterns and recognition path during voice command recognition. There are two kinds of voice command recognition: single command and compound command. Single command is represented by SingleCmd and could be recognized individually. Compound command means the combination of pre-command(PreCmd) , central-command(CentralCmd) and post-command(PostCmd). The grammar is defined by the definition of SingleCmd , PreCmd, CentralCmd and PostCmd. Below is an example of grammar.



From the above grammar, examples of effective recognizable voice commands are ▲ 再見 ▲ 數字撥號 ▲ 打電話到王小明手機 ▲ 開啓計算機 ▲ 查詢通話紀錄 , etc.

But because of possible recognition errors, wrong combinations of commands, such as ▲ 打電話到計算機 ▲ 開啓王小明, may be recognized. These wrong combinations of commands must be rejected or handled by application.

11.1.4.1 Grammar Types

In the grammar definition, we could define just single command or just compound command and even both. In compound command, CentralCmd must be defined and PreCmd and PostCmd are not necessarily defined. When PreCmd is defined, it must be recognized; however, PostCmd is not "must be" recognized. PreCmd and PostCmd must be recognized along with CentralCmd; they couldn't be recognized individually.

There are 9 kinds of grammar types:

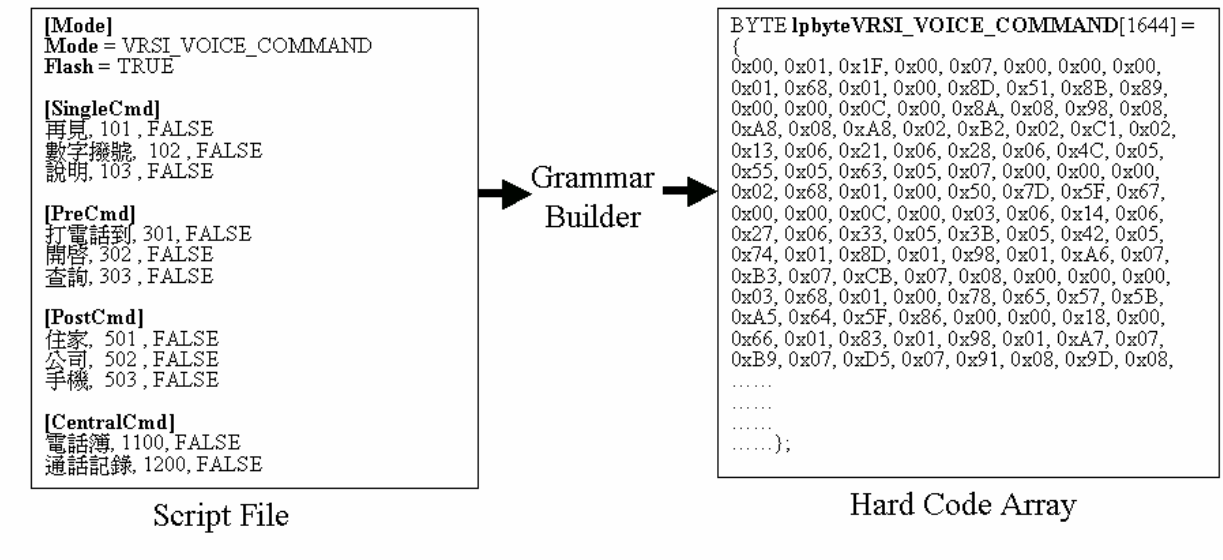
Type	Grammar	Effective Recognition Path
1	[SingleCmd] [PreCmd] [CentralCmd] [PostCmd]	P1 : SingleCmd P2 : PreCmd + CentralCmd + PostCmd P3 : PreCmd + CentralCmd
2	[SingleCmd]	P1 : SingleCmd
3	[SingleCmd] [CentralCmd]	P1 : SingleCmd P2 : CentralCmd
4	[SingleCmd] [PreCmd] [CentralCmd]	P1 : SingleCmd P2 : PreCmd + CentralCmd
5	[SingleCmd] [CentralCmd] [PostCmd]	P1 : SingleCmd P2 : CentralCmd P3 : CentralCmd + PostCmd
6	[CentralCmd]	P1 : CentralCmd

7	[PreCmd] [CentralCmd]	P1 : PreCmd + CentralCmd
8	[CentralCmd] [PostCmd]	P1 : CentralCmd P2 : CentralCmd + PostCmd
9	[PreCmd] [CentralCmd] [PostCmd]	P1 : PreCmd + CentralCmd + PostCmd P2 : PreCmd + CentralCmd

Application designer must think over what kind of grammar is most suitable for recognition application and carefully define command patterns of SingleCmd, PreCmd, CentralCmd and PostCmd.

11.1.4.2 The Creation of Grammar

To build a grammar model in VRSI engine, first we need write a script file to describe the grammar definition and then use PC grammar builder tool to generate a hard code array of grammar to embed into VRSI engine.



In grammar definition, all SingleCmd, PreCmd and PostCmd are pre-defined; As for CentralCmd, it could be either pre-defined or dynamically added by adding tags or training tags. In script file, [Mode] must be defined; the value of attribute “Mode” is the name of grammar. As for the attribute “Flash”, when Flash = TRUE, it means CentralCmd could be dynamically added; when Flash = FALSE, CentralCmd could not be dynamically added. Below [SingleCmd], [PreCmd], [PostCmd] or [CentralCmd] are the description of pre-defined command patterns (tags). Take “再見 , 101 , FALSE “ as a example, “再見” is the TagName of the tag, “101” is the TagLong of the tag and “FALSE” is the “IsPeopleName” of the tag.

11.2 Type Definition

11.2.1 VRSI_Status

Definition:

```
typedef enum {
    VRSI_OK,
    VRSI_FAIL
} VRSI_Status;
```

This enumeration defines the return type of VRSI functions. When the VRSI function return VRSI_FAIL, the Media Task could use VRSI_GetErrMsg to get the error message.

11.2.2 VRSI_Event

Definition:

```
typedef enum {
    VRSI_PROCESS,
    VRSI_RECOG_MPR,
    VRSI_RECOG_OK,
    VRSI_DIGIT_RECOG_MPR,
    VRSI_DIGIT_RECOG_OK,
    VRSI_DIGIT_ADAPT_MPR,
    VRSI_DIGIT_ADAPT_OK,
    VRSI_TRAIN_TAG_1ST_MPR,
    VRSI_TRAIN_TAG_2ND_MPR,
    VRSI_TRAIN_TAG_OK,
    VRSI_ADD_TAGS_OK,
    VRSI_TTS_OK,
    VRSI_PLAY_TAG_OK,
    VRSI_ERROR
} VRSI_Event;
```

This enumeration is used as the type of the event given to media handler.

Members:

Member	Description
VRSI_PROCESS	This event will be given when VRSI engine still doesn't get the final function result and needs further process. When Media Task gets this event, it just calls the VRSI_Process( ) function.
VRSI_RECOG_MPR	This event will be given at the start of VRSI recognition session. When Media Task gets this event, it needs to send MMI prompt request for the start of recognition session. When Media Task receives the MMI prompt confirmation, it must call VRSI_MMI_Confirmed to inform VRSI engine.
VRSI_RECOG_OK	Indication of recognition OK. When Media Task gets this event, it could call VRSI_ReadResult to get the recognition result.
VRSI_DIGIT_RECOG_MPR	This event will be given at the start of VRSI digit recognition session. When

	Media Task gets this event, it needs to send MMI prompt request for the start of digit recognition session. When Media Task receives the MMI prompt confirmation, it must call VRSI_MMI_Confirmed to inform VRSI engine.
VRSI_DIGIT_RECOG_OK	Indication of digit recognition OK. When Media Task gets this event, it could call VRSI_ReadResult to get the digit recognition result.
VRSI_DIGIT_ADAPT_MPR	This event will be given at the start of VRSI digit adaptation session. When Media Task gets this event, it needs to send MMI prompt request for the start of digit adaptation session. The adaptation digit string is read by VRSI_ReadPrompt. When Media Task receives the MMI prompt confirmation, it must call VRSI_MMI_Confirmed to inform VRSI engine.
VRSI_DIGIT_ADAPT_OK	Indication of digit adaptation OK.
VRSI_TRAIN_TAG_1ST_MPR	This event will be given at the start of the 1st VRSI training session. When Media Task gets this event, it needs to send MMI prompt request for the start of the 1st VRSI training session. When Media Task receives the MMI prompt confirmation, it must call VRSI_MMI_Confirmed to inform VRSI engine.
VRSI_TRAIN_TAG_2ND_MPR	This event will be given at the start of the 2nd VRSI training session. When Media Task gets this event, it needs to send MMI prompt request for the start of the 2nd VRSI training session. When Media Task receives the MMI prompt confirmation, it must call VRSI_MMI_Confirmed to inform VRSI engine.
VRSI_TRAIN_TAG_OK	Indication of training OK.
VRSI_ADD_TAGS_OK	Indication of adding tags OK. When Media Task gets this event, it could call VRSI_ReadResult to get the result of adding tags.
VRSI_TTS_OK	Indication of TTS OK.
VRSI_PLAY_TAG_OK	Indication of playing tag OK.
VRSI_ERROR	Indication of error. When Media Task gets this event, it could call VRSI_GetErrMsg to get the error message.

11.2.3 VRSI_ErrMsg

Definition:

```
typedef enum {
    VRSI_ERR_NO,
    VRSI_ERR_UNINITIALIZED,
    VRSI_ERR_STATE_UNMATCH,
    VRSI_ERR_BUSY,
    VRSI_ERR_INSUFFICENT_MEM,
    VRSI_ERR_TOO_MANY_TAGS,
    VRSI_ERR_WRONG_DATA_DIR,
    VRSI_ERR_WRONG_DATA_FILE,
    VRSI_ERR_LIBRARY_CORRUPT,
    VRSI_ERR_DB_FULL,
    VRSI_ERR_DB_ERROR,
    VRSI_ERR_NO_MATCH_TAG,
    VRSI_ERR_NO_SOUND,
```

```
    VRSI_ERR_LANG_NOT_SUPPORT,
    VRSI_ERR_BAD_GRAMMAR,
    VRSI_ERR_ADD_TAGS_FAIL,
    VRSI_ERR_TRAIN_TAG_FAIL,
    VRSI_ERR_RECOG_FAIL,
    VRSI_ERR_DIGIT_RECOG_FAIL,
    VRSI_ERR_DIGIT_ADAPT_FAIL,
    VRSI_ERR_TTS_TOO_LONG,
    VRSI_ERR_UNKNOWN
} VRSI_ErrMsg;
```

This enumeration defines the error messages of VRSI engine.

Members:

Member	Description
VRSI_ERR_NO	No error.
VRSI_ERR_UNINITIALIZED	VRSI engine is not initialized.
VRSI_ERR_STATE_UNMATCH	Some VRSI function is called at wrong VRSI state.
VRSI_ERR_BUSY	VRSI engine is busy. When one VRSI function is in progress, another one VRSI function can't be implemented.
VRSI_ERR_INSUFFICENT_MEM	The assigned working memory is not enough.
VRSI_ERR_TOO_MANY_TAGS	Set too many tags beyond the ability that VRSI engine can handle.
VRSI_ERR_WRONG_DATA_DIR	The given data file directory is wrong.
VRSI_ERR_WRONG_DATA_FILE	Can't read data file.
VRSI_ERR_LIBRARY_CORRUPT	VRSI database library is corrupted
VRSI_ERR_DB_FULL	VRSI data file is full.
VRSI_ERR_DB_ERROR	VRSI data file is corrupted.
VRSI_ERR_NO_MATCH_TAG	No available tag matches the input tagID.
VRSI_ERR_NO_SOUND	When recognition / training / adaptation, there is no speech input.
VRSI_ERR_LANG_NOT_SUPPORT	When digit recognition/adaptation, the assigned language is not supported.
VRSI_ERR_BAD_GRAMMAR	Assigned grammar can't work or is not existed.
VRSI_ERR_ADD_TAGS_FAIL	Can't successfully add tags.
VRSI_ERR_TRAIN_TAG_FAIL	Can't successfully train a tag.
VRSI_ERR_RECOG_FAIL	Can't successfully recognize voice commands.
VRSI_ERR_DIGIT_RECOG_FAIL	Can't successfully recognize digits.
VRSI_ERR_DIGIT_ADAPT_FAIL	Can't successfully adapt digits.
VRSI_ERR_TTS_TOO_LONG	The synthesized TTS playback data is over the engine data buffer.
VRSI_ERR_UNKNOWN	Unknown error

11.2.4 VRSI_Language

Definition:

```
typedef enum{
    VRSI_LANG_TAIWAN,
    VRSI_LANG_CHINA,
    VRSI_LANG_AMERICAN,
```

VRSI_LANG_BRITISH
} **VRSI_Tag_Param**;

Members:

Member	Description
VRSI_LANG_TAIWAN	Mandarin in Taiwan
VRSI_LANG_CHINA	Mandarin in China
VRSI_LANG_AMERICAN	American English
VRSI_LANG_BRITISH	British English

11.2.5 VRSI_Tag_Param

Definition:

```
typedef struct{
    kal_uint16 *pTagName;
    kal_int32 tagLong;
    kal_uint16 tagID;
    kal_uint8 isPeopleName;
} VRSI_Tag_Param;
```

Members:

Member	Description
pTagName	Pointer to the TagName of a tag
TagLong	The TagLong attribute of a tag
TagID	The TagID attribute of a tag
isPeopleName	The IsPeopleName attribute of a tag

11.2.6 VRSI_AddTags_Param

Definition:

```
typedef struct {
    kal_uint16 **ppTagName;
    kal_int32 *pTagLong;
    kal_uint8  *pIsPeopleName;
    kal_uint16 numTag;
} VRSI_AddTags_Param;
```

Members:

Member	Description
ppTagName	The pointer to the array of TagName attributes of “to-be-added” tags
pTagLong	The pointer to the array of TagLong attributes of “to-be-added” tags
pIsPeopleName	The pointer to the array of IsPeopleName attributes of “to-be-added” tags
numTag	The number of tags to be added into VRSI engine

11.2.7 VRSI_AddTags_Result

Definition:

```
typedef struct {
    kal_uint16 *pTagID
    kal_int32 *pTagLong;
    kal_uint16 numTag;
} VRSI_AddTags_Result;
```

Members:

Member	Description
PTagID	The pointer to the array of TagID attributes of “successfully-added” tags
pTagLong	The pointer to the array of TagLong attributes of “successfully-added” tags
numTag	The number of tags that have been successfully added into VRSI engine

11.2.8 VRSI_Recog_Result

Definition:

```
typedef struct {
    kal_uint16 *pTagID;
    kal_int32 *pTagLong;
    kal_uint16 **ppTagName;
    kal_uint16 numTag;
    kal_uint16 resType;
    kal_int16 preCmd;
    kal_int16 postCmd;
} VRSI_Recog_Result;
```

This is the data structure used to store recognition result when voice command recognition. It could be read by VRSI_ReadResult API.

Members:

Member	Description
PTagID	The pointer to the array of TagID attributes of recognized CentralCmds
pTagLong	The pointer to the array of TagLong attributes of recognized CentralCmds
ppTagName	The pointer to the array of TagName attributes of recognized CentralCmds
numTag	The number of recognized CentralCmds (greater than 2 when there are homonyms)
ResType	Recognition result type. (Please refer to the following explanation.)
PreCmd	TagLong of PreCmd or SingleCmd
postCmd	TagLong of PostCmd

[Explanation of Recognition Result Type]

From the grammar types, we could summarize 6 possible recognition results

R0 : No result (Recognition Fail)

R1 : SingleCmd

R2 : PreCmd + CentralCmd

R3 : PreCmd + CentralCmd + PostCmd
R4 : CentralCmd
R5 : CentralCmd + PostCmd

When voice command recognition, application designers must get the correct results according to the recognition result type. The following table lists the relationship between recognition result type values of VRSI_Recog_Result data

Type	resType	preCmd	postCmd	numTag	pTagID	pTagLong	ppTagName
R0	0	Non-sense	Non-sense	Non-sense	Non-sense	Non-sense	Non-sense
R1	1	TagLong of SingleCmd	Non-sense	Non-sense	Non-sense	Non-sense	Non-sense
R2	2	TagLong of PreCmd	Non-sense	Number of CentralCmds	TagIDs of CentralCmds	TagLongs of CentralCmds	TagNames of CentralCmds
R3	3	TagLong of PreCmd	TagLong of PostCmd	Number of CentralCmds	TagIDs of CentralCmds	TagLongs of CentralCmds	TagNames of CentralCmds
R4	4	Non-sense	Non-sense	Number of CentralCmds	TagIDs of CentralCmds	TagLongs of CentralCmds	TagNames of CentralCmds
R5	5	Non-sense	TagLong of PostCmd	Number of CentralCmds	TagIDs of CentralCmds	TagLongs of CentralCmds	TagNames of CentralCmds

11.2.9 VRSI_Digit_Recog_Result

Definition:

```
typedef struct {
   kal_uint16 *pDigits;
   kal_uint16 digitLen;
} VRSI_Digit_Recog_Result;
```

Members:

Member	Description
pDigits	The pointer to the array of recognized digits
digitLen	The length of the recognized digits

11.3 Functions

11.3.1 Initialization

In the initialization of VRSI engine, firstly, the Media Task needs to register a callback function for processing VRSI events; secondly, the Media Task asks for the working memory request according to the number of maximum recognition tags and the number of maximum training tags; thirdly, the Media Task sets the working memory.

11.3.1.1 VRSI_Init

VRSI_Status VRSI_Init(void (*vrsi_handler)(VRSI_Event event));

This function is used to initialize VRSI engine and register a callback handler for processing VRSI events.

Parameters:

Parameter	Description
vrsi_callback	A call-back function used for handling VRSI events. Please refer to VRSI_Event enumeration type for the event details.

Return value:
Return **VRSI_OK** if the operation is complete. Return **VRSI_FAIL** if something is wrong.

Example:

```
#define MAX_RECOG_TAGS  500
#define MAX_SD_TAGS  30
#define SI_DATA_DIRECTORY  D:\VRSI\
kal_uint32 bufLen;

void vrsiHandler(VRSI_Event event )
{
    switch( event ) {
        case VRSI_PROCESS:
            :
            break;
        case VRSI_ERROR:
            :
            break;
        ……..
}

VRSI_Init( vrsiHandler );
bufLen = VRSI_GetMemRequest (MAX_RECOG_TAGS, MAX_SD_TAGS);
// allocate working memory
VRSI_SetMem(bufPtr, bufLen, MAX_RECOG_TAGS, MAX_SD_TAGS, SI_DATA_DIRECTORY);
        :
```

11.3.1.2 VRSI_GetMemRequest

kal_uint32 VRSI_GetMemRequest(kal_uint16 maxRecogTag , kal_uint8 maxSDTag);

This function is used to get the working memory request for VRSI engine.

Parameters:

Parameter	Description
maxRecogTag	The number of maximum recognition tags
maxSDTag	The number of maximum SD training tags

Return value:
The size of the working memory request for VRSI engine.

Example:
Refer to the example of VRSI_Init

11.3.1.3 VRSI_SetMem

VRSI_Status VRSI_SetMem(kal_uint8 *memPtr , kal_uint32 memLen , kal_uint16 maxRecogTag , kal_uint8 maxSDTag , kal_uint8 *vrsiDir);

This function is used to set the working memory for VRSI engine.

Parameters:

Parameter	Description
memPtr	The working memory pointer required for VRSI engine
memLen	The size of VRSI working memory
maxRecogTag	The number of maximum recognition tags
maxSDTag	The number of maximum SD training tags
vrsiDir	VRSI data directory where VRSI data files are located

Return value:
Return **VRSI_OK** if the operation is complete. Return **VRSI_FAIL** if something is wrong.

Example:
Refer to the example of VRSI_Init

11.3.2 Voice Command Recognition

Voice command recognition is a recognition limited by assigned grammar. In the grammar description, there are some pre-defined command patterns and when this grammar allows for dynamically adding tags, we could call VRSI_Add_Tags or VRSI_Train_Tag to dynamically add tags. Recognition could be started by VRSI_Recog.

11.3.2.1 VRSI_Add_Tags

VRSI_Status VRSI_Add_Tags(const kal_uint8 *grammar, VRSI_AddTags_Param *param);

This function is used to dynamically add tags by passing tag information about TagLong, TagName, IsPeopleName to automatically generate command patterns in VRSI engine. All tag information of “to-be-added” tags are recorded in the parameter of VRSI_AddTags_Param data type.

Parameters:

Parameter	Description
grammar	Pointer to the assigned grammar
param	Pointer to the “VRSI_AddTags_Param” parameter that records tag information about “to-be-added” tags

Return value:
Return **VRSI_OK** if the operation is complete. Return **VRSI_FAIL** if something is wrong.

Example:

```
const kal_uint8 grammarName[]= {…….};   // grammar
        :
VRSI_AddTags_Param addtags_param;
VRSI_AddTags_Result  *addtags_result;
VRSI_ErrMsg error;
    :
// function entry
{assign addtags_param by the information of “to-be-added” tags}

VRSI_Add_Tags(grammarName, &addtags_param);

// message process
VRSI_Process();  // in case of VRSI_PROCESS event
    :
VRSI_ReadResult((void**)&addtags_result);  // in case  of  VRSI_ADD_TAGS_OK event
    :
error = VRSI_GetErrMsg();  // in case of VRSI_ERROR event
    :
```

11.3.2.2 VRSI_Train_Tag

VRSI_Status VRSI_Train_Tag (const kal_uint8 *grammar, VRSI_Tag_Param *param);

This function is used to train a specific tag by user’s speech.

Parameters:

Parameter	Description
Grammar	Pointer to the assigned grammar
Param	Pointer to the data structure of the trained tag

Return value:
Return **VRSI_OK** if the operation is complete. Return **VRSI_FAIL** if something is wrong.

Example:

```
const kal_uint8 grammarName[]= {…….};   // grammar
        :
VRSI_Tag_Param tag_param;
Kal_uint16 tagID;
VRSI_ErrMsg error;
      :
// function entry
{assign tag information}

VRSI_Train_Tag(grammarName, &tag_param);

// message process
VRSI_Process();  // in case of VRSI_PROCESS event
      :
{ Media send VRSI prompt request to MMI }  // in case of VRSI_TRAIN_TAG_1ST_MPR or
                                           // VRSI_TRAIN_TAG_2ND_MPR event
      :
VRSI_MMI_Confirmed(); // when Media receives MMI confirmation
      :
tagID = tag_param.tagID;   // in case  of  VRSI_TRAIN_TAG_OK event
// application records tagID of the trained tag
      :
error = VRSI_GetErrMsg();  // in case of VRSI_ERROR event
```

11.3.2.3 VRSI_Recog

```
VRSI_Status VRSI_Recog(const kal_uint8 *grammar );
```

This function is used to start voice command recognition.

Parameters:

Parameter	Description
Grammar	Pointer to the assigned grammar

Return value:

Return **VRSI_OK** if the operation is complete. Return **VRSI_FAIL** if something is wrong.

Example:

```
const kal_uint8 grammarName[]= {…….};   // grammar
        :
VRSI_Recog_Result  *recog_result;
VRSI_ErrMsg error;
      :
// function entry

VRSI_Recog(grammarName);
```

```
// message process
VRSI_Process();  // in case of VRSI_PROCESS event
        :
{ Media send VRSI prompt request to MMI }  // in case of VRSI_RECOG_MPR event
        :
VRSI_MMI_Confirmed(); // when Media receives MMI confirmation
        :
VRSI_ReadResult((void**)&recog_result);  // in case  of  VRSI_RECOG_OK event
// application judge the recognition result type and analyze the recognized commands to do further action
        :
error = VRSI_GetErrMsg();  // in case of VRSI_ERROR event
```

11.3.3 Digit Recognition & Adaptation

11.3.3.1 VRSI_Digit_Recog

VRSI_Status VRSI_Digit_Recog(VRSI_Language lang, kal_uint16 limit);

This function is used to start digit recognition.

Parameters:

Parameter	Description
lang	The language of the recognized speech
limit	The length limits of recognized digits. Assigned by bit mask. See the example.

Return value:
Return **VRSI_OK** if the operation is complete. Return **VRSI_FAIL** if something is wrong.

Example:

```
#define MASK_DIGIT_08 0x80
#define MASK_DIGIT_09 0x100
#deinfe MASK_DIGIT_10 0x200
            :
VRSI_Digit_Recog_Result  *recog_result;
VRSI_ErrMsg error;
        :
// function entry
VRSI_Digit_Recog(VRSI_LANG_TAIWAN, MASK_DIGIT_08| MASK_DIGIT_09| MASK_DIGIT_10);
                                                    // the length limits are 8, 9 and 10

// message process
VRSI_Process();  // in case of VRSI_PROCESS event
        :
{ Media send VRSI prompt request to MMI }  // in case of VRSI_DIGIT_RECOG_MPR event
        :
VRSI_MMI_Confirmed(); // when Media receives MMI confirmation
       :
```

```
VRSI_ReadResult((void**)&recog_result);  // in case  of  VRSI_DIGIT_RECOG_OK event
        :
error = VRSI_GetErrMsg();  // in case of VRSI_ERROR event
        :
```

11.3.3.2 VRSI_Digit_Adapt

```
VRSI_Status VRSI_Digit_Adapt( VRSI_Language  lang );
```

This function is used to start digit adaptation.

Parameters:

Parameter	Description
lang	The language of the recognized speech

Return value:
Return **VRSI_OK** if the operation is complete. Return **VRSI_FAIL** if something is wrong.

Example:

```
kal_uint8 promptLen;
kal_uint16 *prompt;
VRSI_ErrMsg error;
        :
// function entry
VRSI_Digit_Adapt(VRSI_LANG_TAIWAN);

// message process
VRSI_Process();  // in case of VRSI_PROCESS event
        :
VRSI_ReadPrompt(&promptLen, &prompt); // in case of VRSI_DIGIT_ADAPT_MPR event
{ Media send VRSI prompt request to MMI }
        :
VRSI_MMI_Confirmed(); // when Media receives MMI confirmation
        :
error = VRSI_GetErrMsg();  // in case of VRSI_ERROR event
        :
```

11.3.4 Playback

11.3.4.1 VRSI_Play_Tag

```
VRSI_Status VRSI_Play_Tag( const kal_uint8 grammar,  kal_uint16 tagID , void*  param);
```

This function is used to play an existed tag.

Parameters:

Parameter	Description

grammar	Pointer to the assigned grammar
tagID	TagID of the tag to be played
param	MEDIA_VMP_AS_SPEECH or MEDIA_VMP_AS_RINGTONE

Return value:
Return **VRSI_OK** if the operation is complete. Return **VRSI_FAIL** if something is wrong.

Example:
```
const kal_uint8 grammarName[]= {…….};   // grammar
VRSI_ErrMsg error;
      :
// function entry
VRSI_Play_Tag(grammarName ,100, MEDIA_VMP_AS_SPEECH);
      :
// message process
VRSI_Process();  // in case of VRSI_PROCESS event
      :
      :
error = VRSI_GetErrMsg();  // in case of VRSI_ERROR event
     :
```

11.3.4.2 VRSI_TTS_Play

VRSI_Status VRSI_TTS_Play(VRSI_Language lang, kal_uint16 *pText);

This function is used to play arbitrary text.

Parameters:

Parameter	Description
lang	The language of the playback speech
pText	Pointer to the text content

Return value:
Return **VRSI_OK** if the operation is complete. Return **VRSI_FAIL** if something is wrong.

Example:
```
UNICODE playText[] = “請問你要確認或取消”
VRSI_ErrMsg error;
      :
// function entry
VRSI_TTS_Play(VRSI_LANG_TAIWAN, playText);
      :
// message process
VRSI_Process();  // in case of VRSI_PROCESS event
      :
VRSI_Play_Process(); // in case of VRSI_PLAY_REQUEST event
      :
```

```
error = VRSI_GetErrMsg();  // in case of VRSI_ERROR event
        :
```

11.3.5 Message Process

When Media Task starts some VRSI function, VRSI engine will send events to Media Task by callback handler. These functions are used in Media Task to deal with the received VRSI events.

11.3.5.1 VRSI_Process

VRSI_Status VRSI_Process(void);

This function is called in case of VRSI_PROCESS event.

Return value:
Return **VRSI_OK** if the operation is complete. Return **VRSI_FAIL** if something is wrong.

Example:
Refer to the example of VRSI_Recog

11.3.5.2 VRSI_ReadPrompt

void VRSI_ReadPrompt (kal_uint8 *promptLen, kal_uint16 **prompt);

This function is used to get the prompt content of VRSI prompt request when needed.

Parameters:

Parameter	Description
promptLen	The pointer to the length of VRSI prompt
prompt	The pointer to the content of VRSI prompt

Return value:
No return value. However , if no prompt , *prompt would be NULL.

Example:
Refer to the example of VRSI_Digit_Adapt

11.3.5.3 VRSI_MMI_Confirmed

void VRSI_MMI_Confirmed(void);

This function is used to inform VRSI engine that MMI has confirmed the previous prompt request.

Return value:
None.

Example:
Refer to the example of VRSI_Recog.

11.3.5.4 VRSI_ReadResult

void VRSI_ReadResult(void **result);

This function is used to read result when adding tags, voice command recognition and digit recognition.

Parameters:

Parameter	Description
result	Pointer to the result data

Return value:
No return value. However , if no result , *result would be NULL.

Example:
Refer to the example of VRSI_Add_Tags, VRSI_Recog and VRSI_Digit_Recog.

11.3.5.5 VRSI_GetErrorMsg

VRSI_ErrMsg VRSI_GetErrMsg(void);

This function is used to get the error message when VRSI APIs return VR_FAIL or when process VR_ERROR event.

Return value:
The error message.

Example:
Refer to the example of VRSI_Recog.

11.3.6 Database Management

We could build many grammars to change recognition sessions for different recognition purposes. Since tags could be dynamically added for some grammars, the management of tags should be taken care by application developer. The following functions are used to maintain the dynamic tags in the view of specific grammar or in the view of the whole VRSI engine.

11.3.6.1 VRSI_Gram_GetTagNum

VRSI_Status VRSI_Gram_GetTagNum(const kal_uint8 *grammar , kal_int32 *tagNum);

This function is used to get the number of dynamic tags of the assigned grammar.

Parameters:

Parameter	Description
grammar	Pointer to the assigned grammar
tagNum	Address to store the returned number

Return value:
Return **VRSI_OK** if the operation is complete. Return **VRSI_FAIL** if something is wrong.

Example:

```
const kal_uint8 grammarName[]= {…….};   // grammar
#define MAX_DYNAMIC_TAGS 500  //  maintain by application to limit the maximum dynamic tags when adding tags
                              //  or training tags
        :
kal_int32 tagNum, readTagNum, notMatchTagNum;
kal_uint16 buffer[MAX_DYNAMIC_TAGS];       // the allocation of buffers is just simple for convenience of explanation
kal_uint16 delBuffer[MAX_DYNAMIC_TAGS];  // application should take care the usage of buffers by itself
kal_bool bExist;

// Database synchronization of the dynamic tags in the specific grammar
VRSI_Gram_GetTagNum(grammarName, &tagNum);

VRSI_Gram_ReadTagIDs(grammarName, buffer, tagNum, & readTagNum);
// Check if readTagNum == tagNum
// Check if the kept TagID values in application match the read TagID values from VRSI engine
// If TagID of tag X exists in application, but not exist in VRSI engine, then delete tag X in application
VRSI_TagExist(grammarName , some TagID kept in application,& bExist );
If(!bExist)
{ delete tag in application }
// If TagID of tag Y exists in VRSI engine, but not exist in application, then delete tag Y in VRSI engine.
// store the TagID of tagY into delBuffer[] , notMatchTagNum

// after check all TagID values in application and in VRSI engine
VRSI_DeleteTags(grammarName , notMatchTagNum, delBuffer, &tagNum, buffer) ;
```

11.3.6.2 VRSI_Gram_ReadTagIDs

VRSI_Status VRSI_Gram_ReadTagIDs(const kal_uint8 *grammar, kal_uint16 *pTagID , kal_int32 bufferSize, kal_int32 *readTagNum);

This function is used to read TagID attributes of dynamic tags of the assigned grammar in an amount of limited buffer size.

Parameters:

Parameter	Description
grammar	Pointer to the assigned grammar
PTagID	Pointer to the buffer that stores the read TagIDs
bufferSize	Size of the buffer that stores the read TagIDs
readTagNum	Address to store the number of read dynamic tags

Return value:

Return **VRSI_OK** if the operation is complete. Return **VRSI_FAIL** if something is wrong.

Example:

Refer to the example of VRSI_Gram_GetTagNum

11.3.6.3 VRSI_Gram_DeleteAllTags

VRSI_Status VRSI_Gram_DeleteAllTags(const kal_uint8 *grammar);

This function is used to delete all dynamic tags of the specific grammar.

Parameters:

Parameter	Description
grammar	Pointer to the specific grammar

Return value:
Return **VRSI_OK** if the operation is complete. Return **VRSI_FAIL** if something is wrong.

Example:

```
const kal_uint8 grammarName[]= {…….};   // grammar
        :
if( user want to reset the grammar) ]
    VRSI_Gram_DeleteAllTags(grammarName);
```

11.3.6.4 VRSI_DeleteTags

VRSI_Status VRSI_DeleteTags(const kal_uint8 *grammar, kal_int32 tagNum, const kal_uint16 *pTagID, kal_int32 *delTagNum, kal_uint16 *pDelTagID);

This function is used to delete dynamic tags with assigned TagID values by application. VRSI engine will report the truly deleted dynamic tags with deleted TagID values.

Parameters:

Parameter	Description
Grammar	Pointer to the specific grammar
TagNum	The number of the dynamic tags to be deleted
PTagID	Array of the TagID values of the dynamic tags to be deleted
DelTagNum	Address to store the number of the dynamic tags that have been deleted
PDelTagID	Array of the TagID values of the dynamic tags that have been deleted

Return value:
Return **VRSI_OK** if the operation is complete. Return **VRSI_FAIL** if something is wrong.

Example:
Refer to the example of VRSI_Gram_GetTagNum

11.3.6.5 VRSI_TagExist

VRSI_Status VRSI_TagExist(const kal_uint8 *grammar,kal_uint16 tagID , kal_bool *bExist);

This function is used to query VRSI engine about the existence of the tag with the assigned TagID value.

Parameters:

Parameter	Description
grammar	Pointer to the specific grammar
tagID	TagID value of the queried tag
bExist	Pointer to the existed status : true or false.

Return value:
Return **VRSI_OK** if the operation is complete. Return **VRSI_FAIL** if something is wrong.

Example:
Refer to the example of VRSI_Gram_GetTagNum

11.3.6.6 VRSI_ReadTags

VRSI_Status VRSI_ReadTags(const kal_uint8 *grammar, kal_int32 tagNum, const kal_uint16 *pTagID, kal_uint16 **ppTagName, kal_int32 *pTagLong);

When tagIDs are known, we could use this function to get the information about tagName and tagLong

Parameters:

Parameter	Description
grammar	Pointer to the specific grammar
tagNum	The number of tags to be queried
pTagID	Pointer to the array of TagID values of tags to be queried
ppTagName	Pointer to the array of returned TagName contents of tags to be queried. If the TagID is not valid, the TagName is NULL.
pTagLong	Pointer to the array of returned TagLong values of tags to be queried. If the returned TagName is NULL, the TagLong is not effective.

Return value:
Return **VRSI_OK** if the operation is complete. Return **VRSI_FAIL** if something is wrong.

Example:

```
const kal_uint8 grammarName[]= {…….};   // grammar
// When application wants to check if tag information in application matches tag information in VRSI engine
#define SIZE_OF_QUERY_BUFFER 100
kal_uint16 bufTagID[SIZE_OF_QUERY_BUFFER];
kal_int32 bufTagLong[SIZE_OF_QUERY_BUFFER];
kal_uint16 *bufTagName[SIZE_OF_QUERY_BUFFER];
kal_int32 tagNum;

// application store TagID values of tags to be queried into bufTagID and calculate tagNum
// notice if the tags to be queried > SIZE_OF_QUERY_BUFFER,
// separate the queried tags into several steps, or augment  SIZE_OF_QUERY_BUFFER
VRSI_ReadTags(grammarName , tagNum, bufTagID, bufTagName, bufTagLong);
        :
```

11.3.7 Termination

When a VRSI function is in progress, Media Task could call VRSI_Stop to stop it; when the VRSI engine is just stopped, another VRSI function could be run afterwards. When Media Task want to release the working memory of VRSI engine, the Media Task should first call VRSI_Stop to stop the VRSI engine and then call VRSI_Close to end it. When the VRSI engine is ended, if Media Task wants to use any VRSI function, it should do the initialization again.

11.3.7.1 VRSI_Stop

VRSI_Status VRSI_Stop(void);

This function is used to stop any VRSI function in progress.

Return value:
Return **VRSI_OK** if the operation is complete. Return **VRSI_FAIL** if something is wrong.

Example:

```
        :
VRSI_Digit_Adapt(VRSI_LANG_TAIWAN);
        :
VRSI_Stop();  // user stop for some reason
        :
VRSI_Digit_Recog(VRSI_LANG_TAIWAN, 0x380) ;
        :
VRSI_Stop(); // user stop for some reason
        :
VRSI_Close(); // user want to leave SI VR application
```

11.3.7.2 VRSI_Close

VRSI_Status VRSI_Close(void);

This function is used to end the VRSI engine. The working memory allocated for VRSI engine could be reused after VRSI_Close.

Return value:
Return **VRSI_OK** if the operation is complete. Return **VRSI_FAIL** if something is wrong.

Example:
Refer to the example of VRSI_Stop.

Index of Figures

Index of Tables