Realtek VoIP Solution

VoIP Programming Guide

Rev. 0.5-draft
Jan. 27 2011

Realtek Semiconductor Corp.
No. 2, Industry E. Rd. IX, Science-Based Industrial Park, Hsinchu 300, Taiwan
Tel: +886-3-5780211 Fax: +886-3-5776047
www.realtek.com.tw

COPYRIGHT

DISCLAIMER

Realtek provides this document "as is", without warranty of any kind, neither expressed nor implied, including, but not limited to, the particular purpose. Realtek may make improvements and/or changes in this document or in the product described in this document at any time. This document could include technical inaccuracies or typographical errors.

TRADEMARKS

Realtek is a trademark of Realtek Semiconductor Corporation. Other names mentioned in this document are trademarks/registered trademarks of their respective owners.

USING THIS DOCUMENT

This document provides is intended for the software engineer's reference and provides detailed programming information.

Though every effort has been made to ensure that this document is current and accurate, more information may have become available subsequent to the production of this guide. In that event, please contact your Realtek representative for additional information that may help in the development process.

REVISION HISTORY

Revision	Release Date	Summary
0.1	2009/04/9	First release (preliminary version)
0.2	2009/08/04	Second release

0.3	2009/10/22	Add new section 6.5 FXS-Pulse Dial Detection Add new section 6.8 FXO Pulse Dial Generation Add new section 6.11 FXS/FXO Line Status Add new API rtk_Get_Session_Statistics() in section 7.3. Add new API rtk_SetFaxModemDet(),rtk_GetCEDToneDetect() in section 7.5.
0.4	2010/03/08 2010/04/16 2010/04/23 2010/04/23 2010/05/19 2010/05/21 2010/05/24	Add new section 6.12 FXS/FXO DTMF Detection Rename section 6.1 to FXS-line, and add new API in this section Add V.152 parameters description in section 13.1 Add jitter buffer optimization factor zero Add new API rtk_Gen_MDMF_FSK_CID(), rtk_Set_FSK_CLID_Para(), rtk_Set_FSK_CLID_Para() in section 6.4.4 Add new section 11.2 Caller ID Generation Example 2 Add new section 12.13 to 12.15 for enumeration of FSK Caller ID Add new section 13.8 Data Structure for FSK Caller ID Remove the API rtk_Set_CID_FSK_GEN_MODE() in section 6.4.4 Add new mode for DTMF Caller ID in section 6.4.2 Add new example code in section 11.1 Add new API rtk_LimitMaxRfc2833DtmfDuration() in section 7.4.1 Add new example code in section 11.3 To correct the wrong API name from rtk_Set_SLIC_Impedance() to rtk_Set_Impedance() in section 6.1.1
0.5	2010/6/28 2010/10/18 2011/01/04 2011/01/24 2011/01/27	Add RTP payload for Redundant Audio Data description in 7.3.1 Add vlan API and example Add busy state return value for API rtk_Gen_FSK_CID(),rtk_Gen_MDMF_FSK_CID(), and rtk_Gen_FSK_VMWI(). Add new API rtk_SetRFC2833TxConfig() and update section 11.3 example. Add new parameters for API rtk_Set_Pulse_Digit_Det() Add pulse dial sample code in section 11.11

删除: 122

1. Overview

The Realtek VoIP SDK provides comprehensive voice codec support including G.711, G.729, G.723, G.726, iLBC and GSM. All these codecs are tested and measured with Digital Line Speech Analyzer. The wideband codec, for example G.722, is being developed and will be released in no time. The Table 1 shows the voice codecs information of RTK VoIP SDK.

格式化: 字型: (英文) Arial

刪除: Table 1

Codec Name	Bitrate	Fidelity	Avg. Quality (PESQ-LQ)[1]
G.711 ulaw	64kbps	Narrowband	4.33
G.711 alaw	64kbps	Narrowband	4.33
G.729/AB	8kbps	Narrowband	3.90
G.723 5.3k	5.3kbps	Narrowband	3.67
G.723 6.3K	6.5kbps	Narrowband	3.82
G.726 32K	32kbps	Narrowband	4.09
G.726 16K	16kbps	Narrowband	2.94
G.726 24K	24kbps	Narrowband	3.72
G.726 40K	40kbps	Narrowband	4.19
GSM-FR	13.2bps	Narrowband	3.72
ILBC[2]	20ms	Narrowband	3.87
G.722	64kbps	Wideband	4.36

Table 1. VoIP Codec for VoIP SDK

The VoIP call features can be accomplished by using the powerful VoIP Interface APIs. The VoIP Interfaces APIs provide various functions to control the DSP and network interface. The Table 2 shows the call features supported in RTK VoIP SDK.

Type of Feature	Call Feature
REGISTER	Multi-Proxy REGISTER
	Single-Proxy REGISTER
Non-REGISTER	Direct IP CALL

[1] The value of PESQ-LQ depends on platform. The above figure refers to the test report of RTL8186

[2,3] This is new codec which is available depending on platform.

Incoming Call	Answer call
	Drop call
Outing Call	Cancel Call
	Drop Call
Call Waiting	Call waiting tone
	Call waiting caller id
	Alternate call after call waiting
Hold	Hold/Resume call
	Consultation call
Call Forwarding	Immediate Forward
	Busy Forward
	No Answer Forward
Transfer Call	Blind Transfer call
	Attended Transfer call
Conference	3-way Conference
Miscellaneous	Speed Dial
	Auto Dial
	Off-hook Alarm
	Flash Hook Time
	Hot Line
	DND
	Off-hook Password-FXS
	Off-hook Password-FXO
	PSTN Routing Prefix
	Reject Direct IP Call
NAT Traversal	Outbound Proxy
	Stun
VoIP Security	SRTP
	SIP Over TLS

Table 2 Call features for VoIP SDK

2. System Architecture

The Realtek VoIP architecture includes several components. These components can be classified into two categories, the user space components and kernel space components. The user space component, for example sipua, can control the kernel space components through I/O control and system call. However, the developer should know about the detail of operation of kernel space components. This is too complex and time consuming for user to understand the detail of the operations. Owning this, VoIP interface APIs are developed. Infect, these APIs are conceptual layer for developers who can just concern about the input parameters and return values of the APIs. Most of the VoIP features can be achieved by calling the VoIP interface APIs. The Figure 1 shows the main components of VoIP architecture.

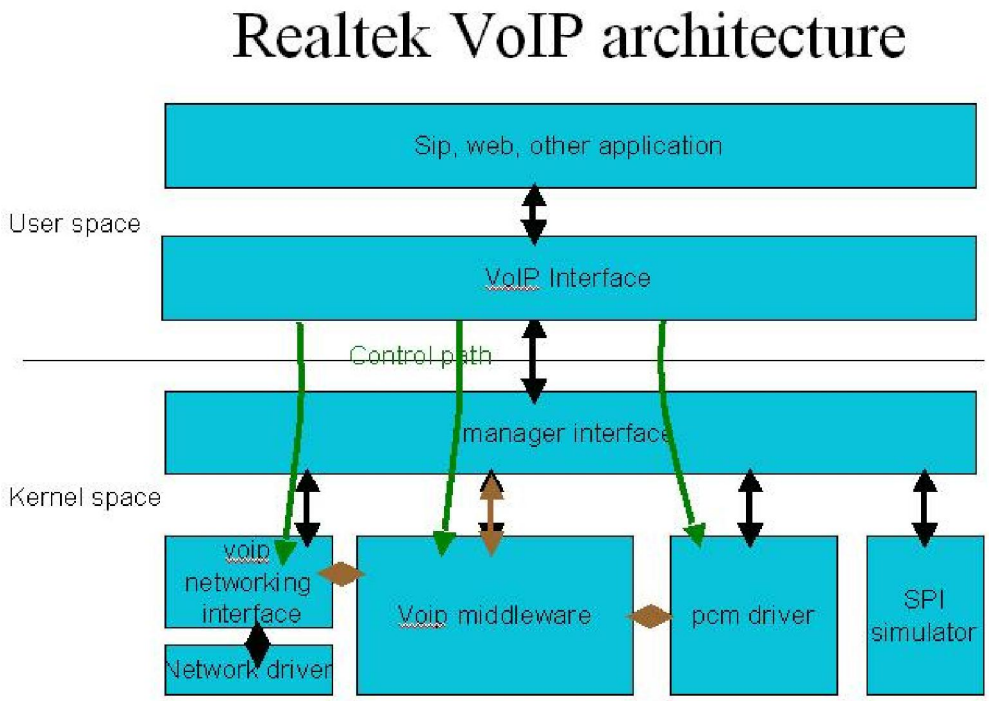

Figure 1 VoIP Architecture

Application: The sipua generates SIP message according to RFC3261. As a caller, it sends the SIP request and receives SIP response over TCP/IP stacks. On the other hands, it receives SIP request and sends response to caller as it acts as callee. The other application for VoIP is web.

Infact main purpose of the web is the interface that allows user to input the VoIP parameters of the device. The parameters then are saved in flash memory.

VoIP interface: The user space application, sipua for example, controls the kernel space components through VoIP interface APIs. Including VLAN, DSP, RTP and phone functions, the VoIP interface APIs is the basic components for setting up the VoIP function.

Manager interface: This interface accepting the VoIP interface command controls the VoIP kernel components.

VoIP network interface: This interface accepts the data of RTP set by VoIP interface. There may be different network interfaces for different platform. For example, the 8186 platform includes static IP, DHCP, PPPoE, PPTP and L2TP.

Network driver: The RTP packet different form SIP packet is sent by kernel space module. So the VoIP packets will be set as higher priority in queue. The network driver sends the VoIP data through wire.

VoIP middeware: This is the core component for VoIP SDK. This DSP toolbox includes codec, jitter buffer, AEC, LEC, tone detection, dtmf detection etc.

PCM driver: This module accepts the data of VoIP middleware and converts the data for generating the VoIP packet.

SPI simulator: This module imitates the SPI to control the SLIC.

Components	Module Name	Corresponding Location in SDK
Application	Solar, solar monitor	VoIP-ATA/AP/rtk_voip/src
Application	Webs	VoIP-ATA/AP/goahead-2.1.1/LINUX
VoIP Interface	VoIP interface	VoIP-ATA/AP/rtk_voip/voip_manager
Manager Interface	VoIP manager	VoIP-ATA/linux-2.4.18/rtk_voip/voip_manager
VoIP Network Interface	VoIP_RX, VoIP_TX	VoIP-ATA/linux-2.4.18/rtk_voip/voip_rx VoIP-ATA/linux-2.4.18/rtk_voip/voip_tx
Network Interface	Network Driver	VoIP-ATA/linux-2.4.18/drivers/net
VoIP Middleware	VoIP DSP	VoIP-ATA/linux-2.4.18/rtk_voip/voip_dsp
PCM Driver	VoIP Driver	VoIP-ATA/linux-2.4.18/rtk_voip/voip_drivers
SPI Simulator	VoIP Driver	VoIP-ATA/linux-2.4.18/rtk_voip/voip_drivers

格式化：義大利文（義大利）

格式化：義大利文（義大利）

Table 3 The VoIP components

3. Flash Setting for VoIP Parameters

The VoIP parameters such as proxy server IP, phone number and login name etc are stored in flash of the system. These parameters can be set or shown in web pages. However, there is another interface for setting and getting these parameters. By inputting the command in the console, the VoIP parameters can be extracted. The Figure 2 command will show the all parameters of VoIP in the flash. The Figure 3 command will show the single parameter of the VoIP parameter. The Figure 4 command will set the definite value to the VoIP parameter.

```
#flash voip
```

Figure 2. Dump the VoIP parameters in the flash

```
#flash voip get <parameter name>
```

Figure 3. Dump the single VoIP parameter

```
#flash voip set <parameter name> <value>
```

Figure 4. Set the VoIP parameter to the definite value

The flash layout is divided into 3 sections. They are the current setting, the default setting and the hardware setting. The current setting section contains the user settings from the web pages. The default setting section contains the settings of factory reset. The hardware setting section contains the hardware information such as MAC address of Ethernet.

The file with the path “VoIP-ATA/AP/rtk_voip/includes/voip_flash.h” contains the structure of VoIP configure parameters[3]. The corresponding file “VoIP-ATA/AP/rtk_voip/flash/voip_flash.c” contains the default setting of the VoIP parameters and the flash read/write functions. The flash read sample codes are shown in Figure 5 while the flash write sample codes are shown in Figure 6.

```
voipCfgParam_t *pVoIPCfg;
if (voip_flash_get(&pVoIPCfg) != 0)
        return -1;
```

Figure 5. Flash read for VoIP parameters

[3] The location of the files will be different according to different platforms.

```
voipCfgParam_t *pVoIPCfg;
/*do setting */
…
voip_flash_set(pVoIPCfg);
```

Figure 6. Flash set for VoIP parameters

4. VoIP Middleware

The VoIP Middleware supports many features. These include setting the echo tail length, lighting the LED, setting the speaker and microphone voice gain and setting the speaker and microphone AGC.

4.1. DSP Default Setting

- Line Echo Cancellation Tail Length: 2 msec with non-linear linear process
- Jitter Buffer Control:
  - Min. delay:40ms
  - Max. delay: 130ms
  - Optimization factor: 7
- Vocie Active Detection(VAD): disable
- Packet loss concealment(PLC): disable
- Auto Gain Control:
  - Speaker AGC: disable
  - Speaker AGC Required Level: Level 0 (up to level 8)
  - Speaker AGC Gain Up: 5 dB
  - Speaker AGC Gain Down: 5 dB
  - MIC AGC: default disable
  - MIC AGC Required Level: 0 (up to level 8)
  - MIC AGC Gain Up: 5 dB
  - MIC AGC Gain Down: 5 dB
- Voice Gain:
  - Speaker (Tx): 0dB
  - MIC(Rx): 0dB

- Caller ID Generation:
  - Software Caller ID generation (Hardware mode is not supported now)
  - Caller ID before 1st Ring as default
  - No Polarity Reversal before Caller ID
  - No Short Ring before Caller ID
  - No Alert Tone before Caller ID
  - DTMF Caller ID:
    - Start digit: A
    - End digit: C
  - FSK Caller ID:
    - Bellcore as default
    - No Date, Time, and Name
- Caller ID detection: default detect DTMF caller ID only
- DTMF mode: Inband DTMF
- Tone of Country: USA
- Flash Hook Time: 100 msec ~ 300 msec

4.2. VoIP Feature

The variable g_VoIP_Feature contain the VoIP features that the user selects from menuconfigure in kernel directory. The initial value for this variable is 0. The function rtk_Get_VoIP_Feature must be called first to retrieve the VoIP feature value. The Figure 7 shows the sample codes for VoIP features extraction. The function of rtk_InitDSP must be called before any use of manipulative function of voice.

```
rtk_Get_VoIP_Feature();
if ((g_VoIP_Feature & DAA_TYPE_MASK) == NO_DAA)
    fxo_type = FXO_NO_DAA;
```

Figure 7. The extraction of the VoIP features

```
// init dsp
rtk_InitDSP(0);
```

```
rtk_Set_Voice_Gain(0, 0, 0);      // 0db
rtk_Set_echoTail(0, 4);           // 4ms
```

Figure 8. Calling rtk_InitDSP before calling voice manipulative function

4.2.1. Software API

Prototype:	Int32 rtk_Get_VoIP_Feature(void)
Description:	Get VoIP Feature
Parameters:	Void
Return Values:	0 success
Reference:	FXO

Prototype:	Int32 rtk_InitDSP(int ch)
Description:	Interface Initialization
Parameters:	ch:The channel number
Return Values:	0 success
Reference:	Call Conference, FXO, RTP

4.3. LEC

The Line Echo Cancellation (LEC) is used to remove the echo caused by the delay of the telephone network line. The tail length can be set as 2ms, 4ms, 8ms, 16ms and 32ms. The higher value of tail length requires more computation time. The Figure 9 shows the sample code for setting the channel 0 for 2ms tail length.

```
rtk_Set_echoTail(0, 2);
```

Figure 9. Set the channel 0 for 2ms tail length for LEC

4.3.1. Software API

Prototype:	Int32 rtk_Set_echoTail(uint32 chid, uint32 echo_tail)
Description:	Set echo tail for LEC
Parameters:	Chid:The channel number echo_tail:The echo tail length (ms)

Return Values:	0 success
Note:	1~16ms in G.711, 1~4ms in G.729/G.723
Reference:	FXO

5. Voice Gain

User can set up the volumes of the speaker and microphone by call the function of "rtk_Set_Voice_Gain".

5.1.1. Software API

Prototype:	int32 rtk_Set_Voice_Gain(uint32 chid, int spk_gain, int mic_gain)
Description:	Set the speaker and mic voice gain
Parameters:	chid:The channel number spk_gain:-32~31 dB (-32: mute, default is 0 dB) mic_gain:-32~31 dB (-32: mute, default is 0 dB)
Return Values:	0 success
Reference:	FXO, phone

5.2. Auto Gain Control

The function of "rtk_Set_Voice_Gain" sets the volume of the speaker and volume of microphone. The range is between –32dB and 31dB. The function of "rtk_Set_SPK_AGC" sets the speaker auto gain control. The value of "support_gain**"** is either 0 or 1(not support/support). The function of "rtk_Set_SPK_AGC_LVL" sets the speaker auto gain control level. The parameter of level is between 0 to 8 and the default value is 0. The functions of "rtk_Set_MIC_AGC_GUP" and "rtk_Set_MIC_AGC_GDOWN" set the auto gain control down and up value. The parameter of gain can be 0 to 8. The corresponding functions of microphone are the same as speaker.

5.2.1. Software API

Prototype:	int32 rtk_Set_SPK_AGC(uint32 chid, uint32 support_gain)
Description:	rtk_Set_SPK_AGC

Parameters:	chid:The channel number support_gain:support_gain 0: not support, 1:supported
Return Values:	0 success

Prototype:	int32 rtk_Set_SPK_AGC_LVL(uint32 chid, uint32 level)
Description:	rtk_Set_SPK_AGC_LVL
Parameters:	chid:The channel number level:level
Return Values:	0 success

Prototype:	int32 rtk_Set_SPK_AGC_GUP(uint32 chid, uint32 gain)
Description:	rtk_Set_SPK_AGC_GUP
Parameters:	chid:The channel number gain:gain
Return Values:	0 success

Prototype:	int32 rtk_Set_MIC_AGC(uint32 chid, uint32 support_gain)
Description:	rtk_Set_MIC_AGC
Parameters:	chid:The channel number support_gain:support_gain
Return Values:	0 success

Prototype:	int32 rtk_Set_MIC_AGC_LVL(uint32 chid, uint32 level)
Description:	rtk_Set_MIC_AGC_LVL
Parameters:	chid:The channel number level:level
Return Values:	0 success

rototype:	int32 rtk_Set_MIC_AGC_GUP(uint32 chid, uint32 gain)
Description:	rtk_Set_MIC_AGC_GUP
Parameters:	chid:The channel number

	gain:gain
Return Values:	0 success

Prototype:	int32 rtk_Set_MIC_AGC_GDOWN(uint32 chid, uint32 gain)
Description:	rtk_Set_MIC_AGC_GDOWN
Parameters:	chid:The channel number gain:gain
Return Values:	0 success

5.3. Jitter Buffer

A jitter buffer is space for storing the arriving packets temporarily. In order to make the voice to be sounded more smoothly, a jitter buffer is needed to minimize the delay variations. The minimum delay can be set at the range of 40ms to 100ms. The recommended value is 40ms. The maximum delay can be set at the range of 130ms to 300ms and the recommended value is 130ms. The optimization factor is the value for adjusting the quality of the voice. This is shown in Figure 10. The higher value of optimization means the better quality but more delay. The medium value of 7 is recommended. The value of 0 and 13 are fix delay, but first one is for low buffering application, the other is for FAX or modem only. These three values can be set to structure of rtp configuration, TstVoipMgrRtpCfg as shown in Figure 11.

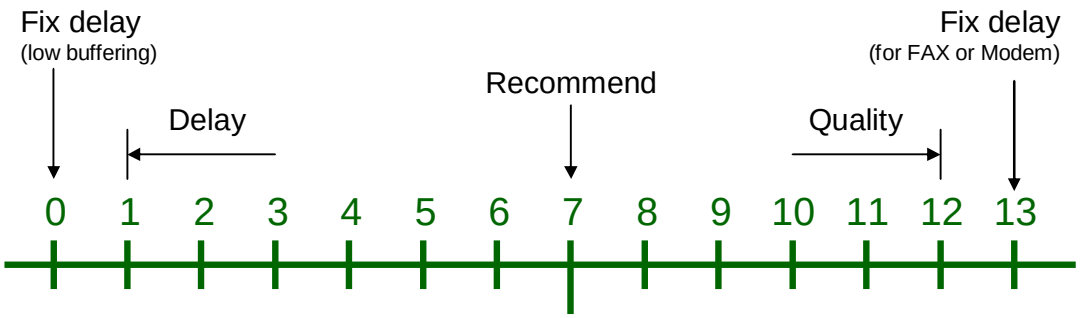

Figure 10 The optimization factor for jitter buffer

```
typedef struct
{
        …
        Tuint32 nJitterDelay;
        Tuint32 nMaxDelay;
        Tuint32 nJitterFactor;
        …
} TstVoipMgrRtpCfg;
```

Figure 11 jetter buffer parameters setting

5.4. Miscellaneous

5.4.1. Software API

Prototype:	int rtk_sip_register(unsigned int chid, unsigned int isOK)
Description:	Light LED if SIP Register OK
Parameters:	chid:The channel number isOK:1: SIP register OK, 0: SIP register failed
Return Values:	0 success

6. FXS/FXO

The analogue terminal adaptor (ATA) usually has FXS port, FXO port and Ethernet port. User can make call through internet or through PSTN. It is possible for an ATA to have two FXS ports. In this case, each telephone will have individual phone number. By using the VoIP SDK, user can accomplish the FXS and FXO functions. The following two functions will illustrate the use of these FXS APIs and FXO APIs.

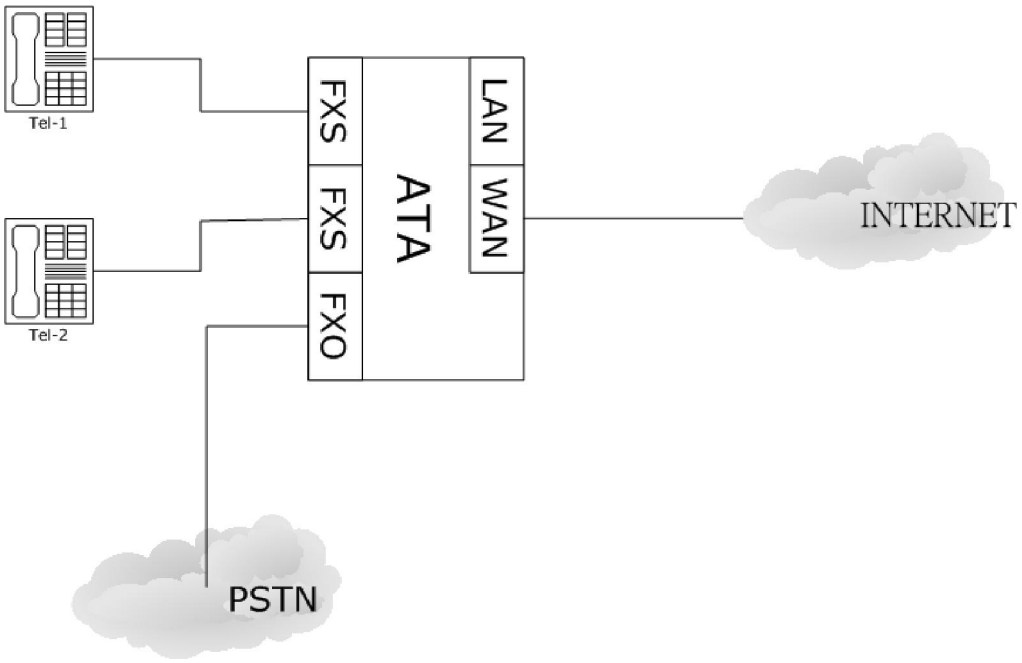

Figure 12 The FXS and FXO diagram

6.1. FXS–Line

6.1.1. Software API

Prototype:	int32 rtk_Set_Country_Impedance(_COUNTRY_ country)
Description:	Set the SLIC impedance according to the Country
Parameters:	country: the selected country COUNTRY_USA: 600 COUNTRY_UK: 270 ohm + (750 ohm||150nF) COUNTRY_AUSTRALIA: 220 ohm + (820 ohm||120nF) COUNTRY_HK: 600 COUNTRY_JAPAN: 600 ohm + 1000nF COUNTRY_SWEDEN: 200 ohm + (1000 ohm||100nF) COUNTRY_GERMANY: 220 ohm + (820 ohm||115nF) COUNTRY_FRANCE: 215 ohm + (1000 ohm||137nF)

	COUNTRY_TW: 600 COUNTRY_BELGIUM: 150 ohm + (830 ohm||72nF) COUNTRY_FINLAND: 270 ohm + (750 ohm||150nF) COUNTRY_ITALY: 400 ohm + (700 ohm||200nF) COUNTRY_CHINA: 200 ohm + (680 ohm||100nF) COUNTRY_CUSTOMER: 600
Return Values:	0 success

Prototype:	int32 rtk_Set_Impedance(uint32 chid, uint8 preset)
Description:	Set the SLIC Impedance from preset table
Parameters:	chid:The FXS channel number preset: The SLIC Impedance preset table index. 0: 600 1: 900 2: 250+(750||150nf) 3: 320+(1150||230nf) 4: 350+(1000||210nf)
Return Values:	0 success
Reference:	

Prototype:	int32 rtk_Set_SLIC_Line_Voltage(uint32 chid, uint8 flag)
Description:	Set SLIC line voltage
Parameters:	chid:The FXS channel number flag: The SLIC line voltage flag 0: zero voltage 1: normal voltage 2: reverse voltage
Return Values:	0 success
Reference:	

Prototype:	int32 rtk_Gen_SLIC_CPC(uint32 chid, uint32 cpc_ms)
Description:	Generate SLIC CPC signal

Parameters:	chid:The FXS channel number cpc_ms: SLIC CPC signal period (unit in ms)
Return Values:	0 success
Reference:	

6.2. FXS-Ring

As a caller, an FXS device initials a call by passing ring voltage over the line to the attached FXO device. As a callee, an FXS device receives a call by detecting the line that has been seized. The function of “rtk_SetRingFXS” sets the FXS to ring. The parameter of bRinging is either 0 or 1(silent or ring). The function of “rtk_Set_SLIC_Ring_Cadence” sets the period of cadence on and period of cadence off. This function should use with the function “rtk_SetRingFXS”.

```
rtk_Set_SLIC_Ring_Cadence(0, 2000, 4000);    // 2 sec ON, 4 sec OFF
rtk_SetRingFXS(0, 1);    // enable ring
```

Figure 13. Set the cadence and ring the phone

6.2.1. Software API

Prototype:	int32 rtk_SetRingFXS (uint32 chid,uint32 bRinging)
Description:	Enable/Disable FXS Ringing
Parameters:	chid:the FXS channel number bringing:1 enable, 0 disable ring
Return Values:	0 success
Reference:	Caller id generation, FXO, ring

Prototype:	int32 rtk_Set_SLIC_Ring_Cadence(uint32 chid, uint16 cad_on_msec, uint16 cad_off_msec)
Description:	Set the Ring Cadence of FXS
Parameters:	chid:The FXS channel number cad_on_msec:The time period of Cadence-On cad_off_msec:The time period of Cadence-Off

Return Values:	0 success
Reference:	FXO, ring

Prototype:	int32 rtk_Set_SLIC_Ring_Freq_Amp(uint32 chid, uint8 preset)
Description:	Set the Ring frequency and amputide of FXS
Parameters:	chid:The FXS channel number preset: The Ring frequency, amputide preset table index. 0: 20Hz, 38V 1: 20Hz, 45V 2: 20Hz, 48V (default) 3: 25Hz, 48V 4: 17Hz, 50V 5: 20Hz, 40V
Return Values:	0 success
Reference:	

6.3. FXS-Flash Hook Time

6.3.1. Software API

Prototype:	int32 rtk_Set_Flash_Hook_Time(uint32 chid, uint32 min_time, uint32 time)
Description:	Set the flash hook time for each FXS channel
Parameters:	chid:the FXS channel number min_time: the minimal acceptabl flash hook period time: the maximal acceptabl flash hook period
Return Values:	0 success
Reference:	Call Conference, phone

6.4. FXS-Caller ID

The caller-id will display in the callee's phone during the ringing signal. It can be classified into two categories. One is DTMF mode and the other is FSK mode. For DTMF mode, there are three kinds of DTMF. They are inband, SIP INFO and RFC2833. For FSK mode, there are four kinds of FSK. They are bellcore, etsi, bt and ntt.

6.4.1. DTMF

Dual-tone multi-frequency (DTMF) is a signal used in telecommunication. There are 12 DTMF signals and each of these is made up of two tones. According to the frequency, the tones forming the DTMF signal can be classified into tow groups.

	1209 Hz	1336 Hz	1477 Hz
697 Hz	1	3	3
770 Hz	4	5	6
852 Hz	7	8	9
941 Hz	*	0	#

Table 4. The DTMF signal

6.4.2. Software API

Prototype:	int32 rtk_Gen_Dtmf_CID(uint32 chid, char* str_cid)
Description:	Caller ID generation via DTMF
Parameters:	chid:The FXS channel number str_cid:The Caller ID
Return Values:	0 success
Note:	When application calls this API to send DTMF caller ID, ***DSP ring the phone automatically*** after sending caller ID. So, there is no need to ring the phone by application. If caller ID is configed to send between 1st and 2nd ring, DSP also can handle it with ring cadence automatically.
Reference:	Caller ID Generation Example 1

Prototype:	int32 rtk_Set_CID_DTMF_MODE(uint32 chid, char area, char cid_dtmf_mode)
Description:	Set the DTMF Caller ID Generation Mode

Parameters:	chid:The FXS channel number area: bit0-2: Skip (set to zero for DTMF CID) bit 3: Send cid before 1st ring (1) or between 1st ring and 2nd ring (0). bit 4: Skip (set to zero for DTMF CID) bit 5: Skip (set to zero for DTMF CID) bit 6: Reverse Polarity before Caller ID (Line Reverse) (0: Forward, 1: Reverse) bit 7: Skip (set to zero for DTMF CID) cid_dtmf_mode:set the caller id start/end digit. 0-1bits for starting digit, and 2-3 bits for ending digit. * 00: A, 01: B, 02: C, 03: D bit 4: Auto start/end digit send * 0: auto mode: DSP send start/end digit according to the bit 0-3 setting automatically * 1: non-auto mode: DSP send caller ID string only. If caller ID need start/end digits, developer should add them to caller ID strings.
Return Values:	0 success
Reference:	Caller ID Generation Example 1, FXO

6.4.3. FSK

Frequency-shift keying (FSK) is mainly used for caller ID applications. There are several types of FSK They are listed as following:

1. ETSI FSK (formed by the European Telecommunications -1 and –2)
2. Bellcore FSK (formed by the Telcordia Technologies)
3. BT FSK (formed by the British Telecom)
4. NTT FSK

6.4.4. Software API

Prototype:	int32 rtk_Set_FSK_Area(uint32 chid, char area)
Description:	Set the FSK Area for FSK Caller ID, VMWI Generation
Parameters:	chid:The FXS channel number. area:The area of FSK. (see *CIDAREA*), bit 0-2: Area -> 0: Bellcore, 1: ETSI, 2: BT, 3: NTT

	bit 3: Send Caller ID before 1st ring (1) or between 1st ring and 2nd ring (0). bit 4: Dual Tone before Caller ID (Fsk Alert Tone) (0: False, 1: True) bit 5: Short Ring before Caller ID (0: False, 1: True) bit 6: Reverse Polarity before Caller ID (Line Reverse) (0: Forward, 1: Reverse) bit 7: FSK Date & Time Sync and Display Name (0: False, 1: True)
Return Values:	0 success
Reference:	Caller ID Generation Example 1

Prototype:	int32 rtk_Gen_FSK_CID(uint32 chid, char* str_cid, char* str_date, char* str_cid_name, char mode)
Description:	FSK Caller ID generation
Parameters:	chid:The FXS channel number str_cid:The Caller ID str_date:The Date and Time str_cid_name:The Caller ID Name mode:0: Service Type I, 1: Service Type II
Return Values:	0: success, -2: busy state
Note:	When application calls this API to send FSK caller ID, ***DSP ring the phone automatically*** after sending caller ID. So, there is no need to ring the phone by application. If caller ID is configed to send between 1st and 2nd ring, DSP also can handle it with ring cadence automatically.
Reference:	Caller ID Generation Example 1, Caller ID Generation Example 2

Prototype:	int32 rtk_Gen_CID_And_FXS_Ring(uint32 chid, char cid_mode, char *str_cid, char *str_date, char *str_cid_name, char fsk_type, uint32 bRinging)
Description:	FSK Caller ID generation and Ring Control
Parameters:	chid:The FXS channel number cid_mode: 0: DTMF Caller ID, 1: FSK Caller ID str_cid:The Caller ID str_date:The Date and Time str_cid_name:The Caller ID Name fsk_type:0: Service Type I, 1: Service Type II

	bRinging: Ringing control, 0: ring off, 1: ring on
Return Values:	0 success
Note:	When application calls this API to send FSK caller ID, ***DSP ring the phone automatically*** after sending caller ID even bRinging is equal to zero. So, there is no need to ring the phone by application. If caller ID is configed to send between 1st and 2nd ring, DSP also can handle it with ring cadence automatically. If str_cid[0] is equal to zero, caller ID won't be send. In this case, this API will ring on/off the phone according to the parameter bRinging.
Reference:	Caller ID Generation Example 2

Prototype:	int32 rtk_Gen_MDMF_FSK_CID(uint32 chid, TstFskClid* pClid, uint32 num_clid_element)
Description:	MDMF FSK Caller ID generation
Parameters:	chid:The FXS channel number pClid str_date:The Caller ID Datas num_clid_element: number of Caller ID elements
Return Values:	0: success, -2: busy state
Note:	When application calls this API to send FSK caller ID, ***DSP ring the phone automatically*** after sending caller ID. So, there is no need to ring the phone by application. If caller ID is configed to send between 1st and 2nd ring, DSP also can handle it with ring cadence automatically.
Reference:	Caller ID Generation Example 2

Prototype:	int32 rtk_Set_FSK_CLID_Para(TstVoipFskPara* para)
Description:	Set the FSK Caller ID Parameters
Parameters:	para: FSK Caller ID Parameters
Return Values:	0 success
Reference:	Caller ID Generation Example 2

Prototype:	int32 rtk_Get_FSK_CLID_Para(TstVoipFskPara* para)
Description:	Get the current FSK Caller ID Parameters
Parameters:	para: FSK Caller ID Parameters

Return Values:	0 success
Reference:	Caller ID Generation Example 2

Prototype:	int32 rtk_GetFskCIDState(uint32 chid, uint32* cid_state)
Description:	get the fsk caller id send complete or not
Parameters:	chid:The FXS channel number. cid_state:0:fsk cid send complete 1:sending fsk cid
Return Values:	0 success

6.4.5. VMWI

Voice message waiting indicator is the feature of the illuminate of LED on select telephone to notify the voice mail messages.

6.4.6. Software API

Prototype:	int32 rtk_Gen_FSK_VMWI(uint32 chid, char* state, char mode)
Description:	Generate the VMWI via FSK
Parameters:	chid:The FXS channel number state:The address of value to set VMWI state. (0: off, 1: on) mode:0: type I, 1: type II
Return Values:	0: success, -2: busy state
Reference:	vmwi

6.5. FXS-Pulse Dial Detection

6.5.1. Software API

Prototype:	int rtk_Set_Pulse_Digit_Det(uint32 chid, uint32 enable, uint32 pause_time, uint32 min_break_ths, uint32 max_break_ths)
Description:	Set FXS pulse detection
Parameters:	chid:The FXS channel number. enable:0: disable, 1:enable Pulse detection for FXS

	pause_time: The threshold of the pause duration of the adjacent pulse digit (msec) min_break_ths: The threshold of the min. pulse break time (msec) max_break_ths: The threshold of the max. pulse break time (msec)
Return Values:	0 success

6.6. FXO

As a caller, an FXO device initiates a call by going off-hook to seize the telephone line or DTMF which identity the destination to be called. As a callee, an FXO device receives a call by detecting the ring voltage supplied by the FXS device or going off-hook to answer the call.
The phone connecting to FXS can make a call to PSTN. This can be achieved by connecting to the FXO. The FXO, infect, acts as a converter to transfer the data between FXS and PSTN. The Figure 14 shows the conceptional architecture for FXS making call to PSTN. As the FXS makes a call to PSTN, it sends the SIP message of INVITE to FXO. After the FXO received the INVITE message, it calls the function of “rtk_FXO_RingOn” and the kernel will response with off_hook. Calling the function of GetFXOEvent can get the status of the FXO. The FXO then responses with the 200 OK SIP message and the FXS sends its ACK to FXO. The connection of RTP is built up between FXO and FXS.

格式化: 字型: (英文) Arial

刪除: Figure 14

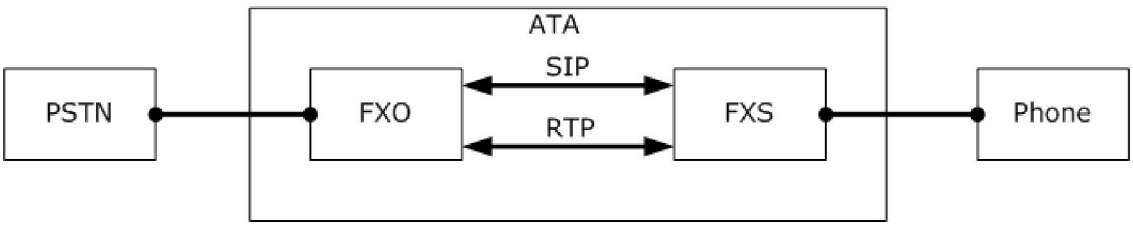

Figure 14. Internal connection for FXO to FXS

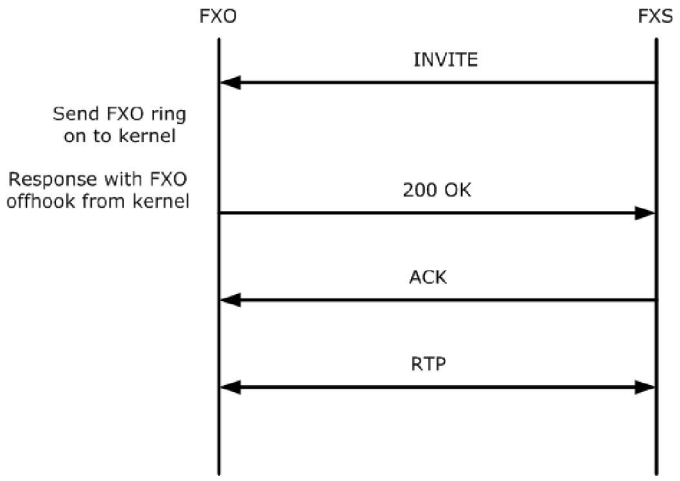

Figure 15. The SIP negoriation for FXO and FXS

As the phone of the PSTN calls the phone of FXS in the ATA, the FXO will send the signal of offhook and do the SIP negotiation with FXS. The connection of RTP between FXO and FXS is built up.

The FXO acts as a sipua that can do negotiation with the other sipua such as softphone.

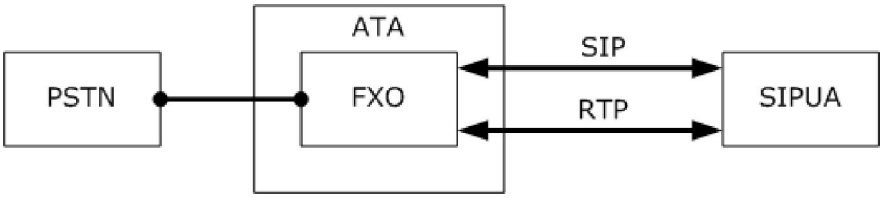

Figure 16. The FXO calls the sipua

The virtual DAA different form real DAA acts as relay between PSTN and FXS. The function of getFXSEvent can get the five statuses. They are off-hook, on hook, DTMF, RingOn and RingOff.

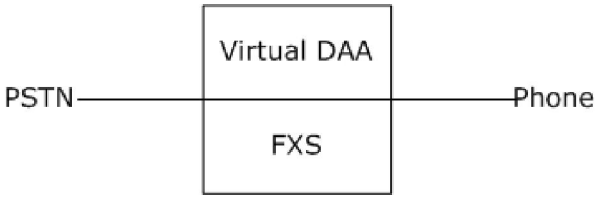

Figure 17. Virtual DAA

6.6.1. Software API

Prototype:	int rtk_DAA_off_hook(uint32 chid)
Description:	Off-Hook in PSTN line
Parameters:	chid:The FXO channel number
Return Values:	0 success
Reference:	FXO

Prototype:	int rtk_DAA_on_hook(uint32 chid)
Description:	On-Hook in PSTN line
Parameters:	chid:The FXO channel number
Return Values:	0 success
Reference:	FXO

Prototype:	int32 rtk_Dial_PSTN_Call_Forward(uint32 chid, uint32 sid, char* cf_no_str)
Description:	Dial PSTN Number for PSTN Call Forward
Parameters:	chid:The FXO channel number sid:The seesion number *cf_no_str:The Pointer of the PSTN Number String
Return Values:	0 success
Reference:	FXO

Prototype:	int rtk_FXO_offhook(uint32 chid)
Description:	FXO:off-hook
Parameters:	chid:The FXO channel number
Return Values:	0 success
Reference:	FXO

Prototype:	int rtk_FXO_onhook(uint32 chid)
Description:	FXO on-hook
Parameters:	chid:The FXO channel number
Return Values:	0 success

Note:	Virtual DAA not support
Reference:	FXO

Prototype:	int rtk_FXO_RingOn(uint32 chid)
Description:	FXO Ring
Parameters:	chid:The FXO channel number
Return Values:	0 success
Note:	Virtual DAA not support
Reference:	FXO

Prototype:	int rtk_FXO_Busy(uint32 chid)
Description:	FXO Busy
Parameters:	chid:The FXO channel number
Return Values:	0 success
Note:	Virtual DAA not support
Reference:	FXO

Prototype:	int32 rtk_GetFxoLineVoltage(uint32 chid, uint32 *pval)
Description:	Get the FXO (DAA) Line Voltage
Parameters:	chid:The FXO channel number pval: FXO line voltage
Return Values:	0 success
Reference:	

6.7. FXO Caller ID

6.7.1. Software API

Prototype:	int32 rtk_Get_DAA_CallerID(uint32 chid, char* str_cid, char* str_date)
Description:	Get the FXO Detected Caller ID
Parameters:	chid:The FXO channel number

	str_cid:The Caller ID String str_date:The Date String
Return Values:	0 success
Note:	Virtual DAA not support. ***How to get FXO Caller ID:*** 1. When application get FXO Ring event, then it's safe to get Caller ID. 2. If Caller ID is set to send between 1st ring and 2nd ring, then DSP report FXO Ring event till 2nd ring, to let application safe to get Caller ID. 3. If no Caller ID is detected, then str_cid[0] = str_date[0] = 0.
Reference:	FXO, rtk_Set_FSK_Area(), rtk_Set_CID_DTMF_MODE(),rtk_GetFxoEvent()

Prototype:	Int32 rtk_Set_CID_Det_Mode(uint32 chid, int auto_det, int cid_det_mode)
Description:	Set the FXO Caller ID Detection Mode
Parameters:	chid:The FXO channel number auto_det: 0: Disable Caller ID Auto Detection, 1: Enable Caller ID Auto Detection (NTT Support), 2: Enable Caller ID Auto Detection (NTT Not Support) cid_det_mode:The Caller ID Mode for Caller ID Detection
Return Values:	0 success
Note:	Virtual DAA not support
Reference:	FXO

6.8. FXO Pulse Dial Generation

6.8.1. Software API

Prototype:	int rtk_Set_Dail_Mode(uint32 chid, uint32 mode)
Description:	Set FXO dial mode
Parameters:	chid: The FXO channel number mode: 0:disable, 1:enable Pulse Dial for FXO
Return Values:	0 success
Note:	Virtual DAA not support

Prototype:	int rtk_Get_Dail_Mode(uint32 chid)
Description:	Get FXO dial mode
Parameters:	chid: The FXO channel number
Return Values:	0: disable 1: enable Pulse dial
Note:	Virtual DAA not support

Prototype:	int rtk_PulseDial_Gen_Cfg(char pps, short make_duration, short interdigit_duration)
Description:	Pulse dial generation config for FXO
Parameters:	pps: The pulse dial gen speed(Pulse per second) make_duration l: Make duration of the pulse digit, unit:10 msec interdigit_duration: The pause time between the pulse digit, unit:10 msec
Return Values:	0 success
Note:	Virtual DAA not support

Prototype:	int rtk_Gen_Pulse_Dial(uint32 chid, char digit)
Description:	Generate pulse dial for FXO
Parameters:	chid: The FXO channel number digit: The pulse dial digit(0~9)
Return Values:	0 success
Note:	Virtual DAA not support

6.9. FXS Event

6.9.1. Software API

Prototype:	int32 rtk_GetFxsEvent(uint32 chid, SIGNSTATE *pval)
Description:	Get the FXS Event
Parameters:	chid: The FXS channel number

	*pval: The pointer of variable to save FXS event
Return Values:	0 success, -1 unknown FXS event
Note:	
Reference:	SIGNSTATE

6.10. FXO Event

6.10.1. Software API

Prototype:	int32 rtk_GetFxoEvent(uint32 chid, SIGNSTATE *pval)
Description:	Get the FXO Event
Parameters:	chid: The FXO channel number *pval: The pointer of variable to save FXO event
Return Values:	0 success, -1 unknown FXO event
Note:	This API report the FXO event to application with *event transformation*. l Ring Start à SIGN_OFFHOOK l Ring Stop à SIGN_ONHOOK l Busy Tone à SIGN_ONHOOK l Dis-connect Tone à SIGN_ONHOOK
Reference:	SIGNSTATE

Prototype:	int32 rtk_GetRealFxoEvent(uint32 chid, FXOEVENT *pval)
Description:	Get the FXO Event
Parameters:	chid: The FXO channel number *pval: The pointer of variable to save FXO event
Return Values:	0 success, -1 unknown FXO event
Note:	This API report the real FXO event to application.
Reference:	FXOEVENT

6.11. FXS/FXO Line Status

6.11.1. Software API

Prototype:	int rtk_line_check(uint32 chid)
Description:	Check the FXS, FXO line status
Parameters:	chid: The FXS/FXO channel number
Return Values:	For FXS: 0: Phone dis-connect, 1: Phone connect, 2: Phone off-hook, 3: Check time out (may connect too many phone set => view as connect), 4: Can not check, Linefeed should be set to active state first. For FXO: 0: PSTN Line connect, 1: PSTN Line not connect, 2: PSTN Line busy
Note:	The FXS line status is not always reliable. It may depend on SLIC, PCB layout, length of line and number of Phones. So it's just for reference. But FXO line status is reliable.
Reference:	

6.12. FXS/FXO DTMF Detection

6.12.1. Software API

Prototype:	int32 rtk_set_dtmf_det_threshold(uint32 chid, int32 threshold)
Description:	Set the DTMF detection threshold
Parameters:	chid: The FXS/FXO channel number threshold: The threshold of DTMF detection. Range: 0 ~ 40, it means 0 ~ -40

	dBm.
Return Values:	0 success
Note:	If this API is not called, the default detection threshold of DSP is -32 dBm.

7. VoIP Session

The VoIP session is a part of VoIP interface. ***Each channel contains two sessions***: ***Session0 and Session1***. There are four modules in each session. The four modules are tone, RTP, DTMF and FAX. The Figure 18 shows an example of the phone A initials a call to phone B with session 0, then phone A hold phone B and makes a call with session 1.

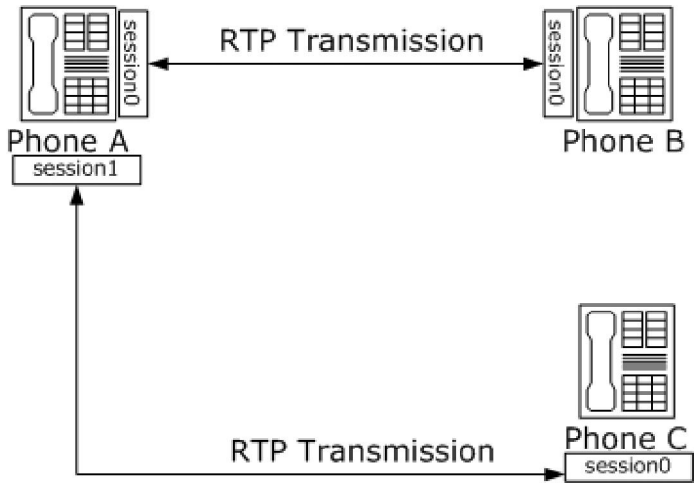

Figure 18-1 Phone A call with different sessions to phone B and Phone C

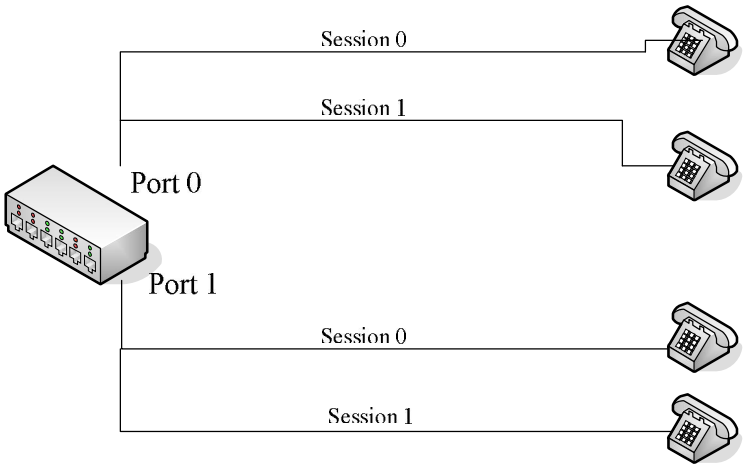

Figure 19-2 Each channel (port) contains two sessions: Session0 and Session1

7.1. Software API

Prototype:	Int32 rtk_SetTranSessionID(uint32 chid, uint32 sid)
Description:	Assign the active session
Parameters:	chid:The channel number sid:The session number

Return Values:	0 success
Reference:	Call Conference, FXO, RTP

7.2. VoIPSession-Tone

The tone table locates in "VoIP-ATA/linux-2.4.18/rtk_voip/voip_dsp/dsp_r0/dspparam.c". Developer can access the table and modify the setting of tones. The Table 5 shows the different tones in tone table.

Number	Tone Name
1	Dial tone
2	Stutter-dial tone
3	Message waiting tone
4	Confirmation tone
5	Ring-back tone
6	Busy tone
7	Congestion tone
8	Roh tone
9	Double ringback tone
10	Sit-no circuit tone
11	Site-intercept tone
12	Sit-vacant tone
13	Sit-reorder tone
14	Calling cord with event tone
15	Calling card tone
16	Call waiting #1 tone
17	Call waiting #2 tone
18	Call waiting #3 tone
19	Call waiting #4 tone
20	Ingress ringback tone

Table 5 Tone table

The tone is various in different countries. The VoIP SDK pre-defined the tones using in these countries. These countries include USA, UK, Australia, Hong Kong, Japan, Sweden, Germany,

France, Taiwan, Belgium, Finland, Italy and China. Beside this users can set their custom tone they like.

7.2.1. Software API

Prototype:	int32 rtk_SetPlayTone(uint32 chid, uint32 sid, DSPCODEC_TONE nTone, uint32 bFlag, DSPCODEC_TONEDIRECTION Path)
Description:	Play/Stop the assigned Tone
Parameters:	chid:The channel number sid:The session number nTone:The tone type bFlag:1: play, 0: stop Path:The tone direction
Return Values:	0 success
Reference:	Call Conference, FXO, Phone and RTP

Prototype:	int32 rtk_Set_Country(voipCfgParam_t* voip_ptr)
Description:	Set the country tone, busy tone detection parametes, and SLIC impedance according to the Country
Parameters:	Voip_ptr The configuration of VoIP
Return Values:	0 success

Prototype:	int32 rtk_Set_Country_Tone (_COUNTRY_ country)
Description:	Set the Tone of Country
Parameters:	country: the selected country
Return Values:	0 success

Prototype:	int32 rtk_Set_Custom_Tone(uint8 custom, st_ToneCfgParam* pstToneCfgParam)
Description:	Set the custom tone
Parameters:	custom:The custom index pstToneCfgParam:The custom tone configuration
Return Values:	0 success

Prototype:	int32 rtk_Use_Custom_Tone(uint8 dial, uint8 ringback, uint8 busy, uint8 waiting)
Description:	Use the custom tone to be dial, ringback, busy, and call waiting tone
Parameters:	dial: Customer dial tone ringback: customer ringing tone busy: customer busy tone waiting: customer call waiting tone
Return Values:	0 success

Prototype:	int32 rtk_SIP_INFO_play_tone(unsigned int chid, unsigned int sid, DSPCODEC_TONE tone, unsigned int duration)
Description:	Play tone when receive SIP INFO
Parameters:	chid:The channel number sid:The session number tone:The tone need to play duration:The tone duration (ms)
Return Values:	0 success

7.3. VoIPSession-RTP

The RTP configuration includes RTP and RTCP. For RTP, the developer can set the payload type as g.911, g.723, g.726, g.729, GSM and iLBC etc. There are 5 states for session state. They are rtp_session_undefined, rtp_session_inactive, rtp_session_sendonly, rtp_session_recvonly and rtp_session_sendrecv. Both RTP sessions are transmitting when session state is set to be rtp_session_sendrecv. In hold case, the holding peer will set rtp_session_sendonly which means to stop receiving the voice data through RTP while the held peer will set rtp_session_recvonly which means to stop sending the voice data through RTP. In this case, only the held peer can hear the voice data form holding peer. The music on hold can be achieved by this way.

The rtp_session_inactive can be used for both peers stop sending and receiving any voice data. The structure of rtp_config is shown as Table 6.

Member	**Variable type**	**Description**

Chid	uint32	Channel id
Sid	uint32	Session id
IsTcp	uint32	Tcp=1 udp=0
RemIp	uint32	Remote ip
RemPort	uint16	Remote prot
ExtIp	uint32	Local ip
ExtPort	uint16	Local port
Tos	uint8	QoS
rfc2833_payload_type_local	uint16	Local RFC2833 payload type
rfc2833_payload_type_remote	uint16	Remote RFC2833 payload type
RemoteSrtpKey	unsigned char	Remote srtp key
LocalSrtpKey	unsigned char	Local srtp key
RemoteCryptAlg	Int	Remote crypt algorithm
LocalCryptAlg	Int	Local crypt algorithm
rtp_redundant_payload_type_local	uint16	local payload type of rtp redundant
rtp_redundant_payload_type_remote	uint16	remote payload type of rtp redundant

Table 6. The variable of rtp configure

The member “rfc2833_payload_type_local” and “rfc2833_payload_type_remote” of this structure ***MUST*** be set from application to make sure that the RFC2833 behavior of DSP works fine. If local or remote side doesn’t support RFC2833, application should set it to ***ZERO***, otherwise, set to the supported payload type of local and remote side. Application should negotiate with another VoIP user agent to decide the remote RFC2833 payload type.
The Realtek VoIP SDK supports both symmetric RTP and non-symmetric RTP. The transmitting RTP port different form the receiving RTP port is said to be non-symmetric RTP.

7.3.1. RTP Payload for Redundant Audio Data

This feature is an implementation of RFC 2198, and it uses redundant data (bandwidth) to deal with packet loss. To enable this function, SIP uses a special media attribute called “red” to negotiate, and it normally looks like:

a=rtpmap: 121 red/8000/1

a=fmtp: 121 18/18

It says that we use dynamic payload type 121 as redundant RTP payload in RTP header ("rtp_redundant_payload_type_local" and " rtp_redundant_payload_type_remote" in Table 7), and primary and secondary codecs are 18 and 18. In this example, static payload types (i.e. 18) are assigned for codecs, but we can also assign dynamic payload types to them (e.g. iLBC=98). This can reference to rtk_SetRtpPayloadType() to give suitable values. However, its limitation is all redundant codecs to be the same, so SIP negotiation should guarantee this.

删除: 6

7.3.2. Software API

Prototype:	int32 rtk_SetRtpConfig(rtp_config_t* cfg)
Description:	Set RTP configuration
Parameters:	cfg The RTP configuration
Return Values:	0 success
Reference:	Call Conference, FXO and RTP

Prototype:	int32 rtk_SetRtcpConfig(rtp_config_t *cfg, unsigned short rtcp_tx_interval)
Description:	Set RTCP configuration
Parameters:	cfg:The RTCP configuration rtcp_tx_interval:The RTCP TX Interval
Return Values:	0 success

Prototype:	int32 rtk_SetRtpPayloadType(payloadtype_config_t* cfg)
Description:	Set RTP payload type
Parameters:	cfg:The RTP payload configuration
Return Values:	0 success
Reference:	Call Conference, FXO and RTP, payloadtype_config_t

Prototype:	int32 rtk_SetRtpSessionState(uint32 chid, uint32 sid, RtpSessionState state)
Description:	Set RTP Session State
Parameters:	chid:The channel number sid:The session number

	state:The RTP Session State
Return Values:	0 success
Reference:	Call Conference, FXO and RTP

Prototype:	int32 rtk_SetConference(TstVoipMgr3WayCfg* stVoipMgr3WayCfg)
Description:	Enable/Disable conference
Parameters:	stVoipMgr3WayCfg:The conference setting
Return Values:	0 success
Reference:	Call Conference

Prototype:	int32 rtk_Hold_Rtp(uint32 chid, uint32 sid, uint32 enable)
Description:	Hold/Resume RTP
Parameters:	chid:The channel number sid:The session number enable:1: Hold, 0: Resume
Return Values:	0 success
Reference:	Call Conference and RTP

Prototype:	int32 rtk_Get_Rtp_Statistics(uint32 chid, uint32 bReset, TstVoipRtpStatistics* pstVoipRtpStatistics)
Description:	Get RTP statistics by channel
Parameters:	chid:The channel number bReset:Reset statistics of the channel pstVoipRtpStatistics:RTP statistics. (If bReset is set, statistics are all zeros.)
Return Values:	0 success

Prototype:	int32 rtk_Get_Session_Statistics(uint32 chid, uint32 sid, uint32 bReset, TstVoipSessionStatistics *pstVoipSessionStatistics)
Description:	Get statistics by session
Parameters:	chid:The channel number sid: The session number

	bReset:Reset statistics of the session pstVoipSessionStatistics: session statistics. (If bReset is set, statistics are all zeros.)
Return Values:	0 success

7.4. VoIPSession-DTMF

There are three modes for DTMF. They are RFC2833, SIP INFO and inband. For RFC2833 mode, DTMF message is sent through RTP, while DTMF message is send through SIP info for SIP info mode. The Figure 20 shows the SIP info message for DTMF 5 with an indicated duration of 160 milliseconds.

```
INFO sip:7007471000@example.com SIP/2.0
Via: SIP/2.0/UDP alice.uk.example.com:5060
From: <sip:7007471234@alice.uk.example.com>;tag=d3f423d
To: <sip:7007471000@example.com>;tag=8942
Call-ID: 312352@myphone
CSeq: 5 INFO
Content-Length: 24
Content-Type: application/dtmf-relay

Signal=5
Duration=160
```

Figure 20 SIP info message for DTMF

7.4.1. Software API

Prototype:	int32 rtk_SetDTMFMODE(uint32 chid, uint32 mode)
Description:	Set the DTMF mode
Parameters:	chid:The channel number mode: The DTMF mode. (see __DTMF_TYPE__)
Return Values:	0 success
Reference:	Call Conference, FXO and RTP

Prototype:	int32 rtk_SetRFC2833SendByAP(uint32 chid, uint32 config)
Description:	Send RFC2833 by AP or DSP
Parameters:	chid:The channel number config: 0: by DSP 1: by AP
Return Values:	0 success
Reference:	RFC2833 Send

Prototype:	int32 rtk_SetRTPRFC2833(uint32 chid, uint32 sid, uint32 digit)
Description:	Send DTMF via RFC2833
Parameters:	chid:The channel number sid:The session number digit:The digit of user input. (0..11, means 0..9,*,#)
Return Values:	0 success
Reference:	RFC2833 Send, Call Conference and RTP

格式化 :字型 :12點

Prototype:	int32 rtk_LimitMaxRfc2833DtmfDuration(uint32 chid, uint32 duration_in_ms, uint8 bEnable)
Description:	Limit the Max. RFC2833 DTMF Duration
Parameters:	chid:The channel number duration: The limited Max. DTMF duration (ms) bEnable 0: disable , 1: enable Max. limitation for DTMF duration
Return Values:	0 success
Reference:	RFC2833 Send, Call Conference and RTP

Prototype:	int32 rtk_SetRFC2833TxConfig (uint32 chid, uint32 volume, uint32 tx_mode)
Description:	Set the RFC2833 Tx packet volume field
Parameters:	chid:The channel number volume: range from 0 to 31 dBm0 tx_mode: 0: DSP mode (RFC2833 packet will send automatically by DSP when DTMF key is pressed), 1: application mode(RFC2833 packet send by application)
Return Values:	0 success

Reference:	RFC2833 Send

7.5. VoIPSession-FAX

There are two kind of FAX transmission. One is T.38 and the other is T.30. For T.38, after SIP negotiation and the modem tone detection caller and callee will transmit through T.38 protocol. For T.30, after SIP negotiation the caller and callee will transmit through G.711. The Figure 21 shows the SIP negotiation for T.38 protocol.

Having detected the CED tone, the callee sends the reinvite SIP message to the caller to propose the T.38 transmission. The Figure 22 shows the SIP re-INVITE message for T.38 and the Figure 23 shows the sample code for fax tone detection.

格式化 :字型 : 英文)Arial

格式化 :字型 : 英文)Arial,使用拼字與文法檢查

刪除: Figure 22

格式化 :字型 : 英文)Arial

刪除: Figure 23

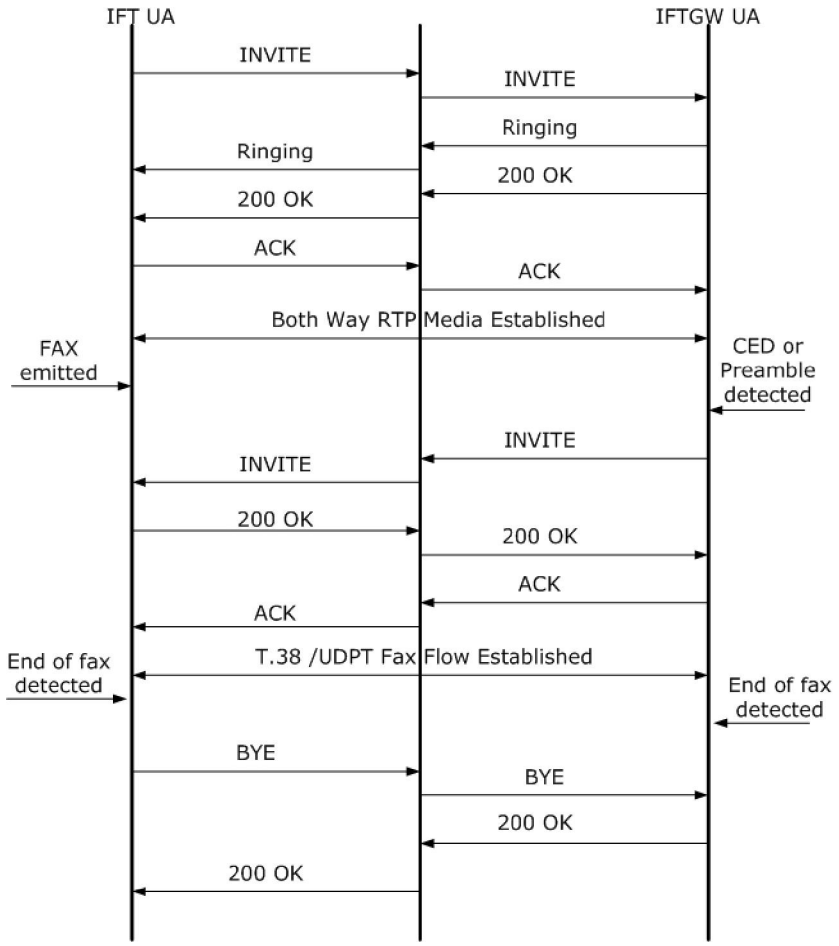

Figure 21. T.38 SIP data flow

```
INVITE sip:+1-650-555-2222@ss1.wcom.com;user=phone SIP/2.0
Via: SIP/2.0/UDP iftgw.there.com:5060
From: sip:+1-303-555-1111@ift.here.com;user=phone
To: sip:+1-650-555-2222@ss1.wcom.com;user=phone
Call-ID: 1717@ift.here.com
CSeq: 56 INVITE
Content-Type: application/sdp
Content-Length: 320
```

```
v=0
o=faxgw1 2890844527 2890844527 IN IP4 iftgw.there.com
s=Session SDP
c=IN IP4 iftmg.there.com
t=0 0
m=image 49172 udptl t38
a=T38FaxVersion:0
a=T38maxBitRate:14400
a=T38FaxFillBitRemoval:0
a=T38FaxTranscodingMMR:0
a=T38FaxTranscodingJBIG:0
a=T38FaxRateManagement:transferredTCF
a=T38FaxMaxBuffer:260
a=T38FaxUdpEC:t38UDPRedundancy
```

Figure 22. SIP re-INVITE message for T.38

```
uint32 faxFlag;
rtk_GetCEDToneDetect(0, faxFlag)
if (faxFlag == FAX_DETECT)
{
/*do re-INVITE*/
…
}
```

Figure 23. Do re-INVITE after fax detection

7.5.1. Software API

Prototype:	int32 rtk_SetFaxModemDet(uint32 chid, uint32 mode)
Description:	Fax/Modem detection config
Parameters:	chid:The channel number mode: 0:auto. 1: fax. 2:modem
Return Values:	0 success
Reference:	rtk_GetCEDToneDetect()

Prototype:	int32 rtk_GetCEDToneDetect(uint32 chid, uint32 * pval)
Description:	FAX/Modem detection
Parameters:	chid:The channel number pval:0: No Detected, 1: FAX detected, 2: Modem detected
Return Values:	0 success
Note:	When Fax/Modem detect mode is auto, detect result may be 0, 1, or 2. When Fax/Modem detect mode is fax, detect result may be 0, or 1. When Fax/Modem detect mode is modem, detect result may be 0, or 2.

Prototype:	int32 rtk_GetFaxDisTxDetect (uint32 chid, uint32 *pval)
Description:	FAX DIS(Digital Identification Signal) TX detection
Parameters:	chid:The channel number pval: 1: DIS TX detected, 0: No DIS
Return Values:	0 success
Note:	DIS TX event path: ethernet à device à FAX machine

Prototype:	int32 rtk_GetFaxDisRxDetect (uint32 chid, uint32 *pval)
Description:	FAX DIS(Digital Identification Signal) RX detection
Parameters:	chid:The channel number pval: 1: DIS RX detected, 0: No DIS
Return Values:	0 success
Note:	DIS RX event path: FAX machine à device à ethernet

Prototype:	int32 rtk_GetFaxEndDetect(uint32 chid, uint32 *pval)
Description:	FAX end detection
Parameters:	chid:The channel number pval:1: FAX end detected, 0: No FAX end.
Return Values:	0 success

8. IVR

The interactive voice response (IVR) is the pre-recorded or the dynamically generated audio for system information such as IP, default gateway and phone number. On the other hand user can configure the system by passing the special combination of keys. This function is very useful when web interface malfunctions.

8.1. Software API

Prototype:	int rtk_IvrStartPlaying(unsigned int chid, unsigned char* pszText2Speech)
Description:	Play a text speech
Parameters:	chid:The channel number pszText2Speech:The text to speech
Return Values:	Playing interval in unit of 10ms
Reference:	FXO

Prototype:	int rtk_IvrStartPlayG72363(unsigned int chid, unsigned int nFrameCount, const unsigned char* pData)
Description:	Play a G.723 6.3k voice
Parameters:	chid:The channel number nFrameCount:The number of frame to be decoded pData:Point to data
Return Values:	Copied frames, so less or equal to nFrameCount

Prototype:	int rtk_IvrPollPlaying(unsigned int chid)
Description:	Poll whether it is playing or not
Parameters:	chid:The channel number
Return Values:	1:Playing, 0:Stopped

Prototype:	int rtk_IvrStopPlaying(unsigned int chid)
Description:	Stop playing immediately
Parameters:	chid:The channel number

Return Values:	0 Success
Reference:	FXO

9. VoIP Security

There are two aspects for VoIP security. One is SIP over TLS transport and the other is security RTP (SRTP) that is defined in RFC3711. The SDP embedded in SIP INVITE message contains an attribute of crypto that indicates the algorithm of encryption and the master key for security RTP. The key exchange protocol is defined in RFC4568. The crypto-suite is a negotiated while the key parameter is a declarative parameter. The Figure 24 shows the crypto attribute of SDP. The length of master key is 32, however, it is encoded by base64.

格式化 : 字型 : (英文) Arial

刪除: Figure 24

```
v=0
o=jdoe 2890844526 2890842807 IN IP4 10.47.16.5
s=SDP Seminar
i=A Seminar on the session description protocol
u=http://www.example.com/seminars/sdp.pdf
e=j.doe@example.com (Jane Doe)
c=IN IP4 161.44.17.12/127
t=2873397496 2873404696
m=video 51372 RTP/SAVP 31
a=crypto:1 AES_CM_128_HMAC_SHA1_80
inline:d0RmdmcmVCspeEc3QGZiNWpVLFJhQX1cfHAwJSoj|2^20|1:32
m=audio 49170 RTP/SAVP 0
a=crypto:1 AES_CM_128_HMAC_SHA1_32
inline:NzB4d1BINUAvLEw6UzF3WSJ+PSdFcGdUJShpX1Zj|2^20|1:32
m=application 32416 udp wb
a=orient:portrait
```

Figure 24 SDP Secruity Descriptions

The following code will set the local master key, remote master key and negotiated algorithm.

```
TstVoipMgrSession stVoipMgrSession;
memcpy(stVoipMgrSession.remoteSrtpKey, cfg->remoteSrtpKey, SRTP_KEY_LEN);
memcpy(stVoipMgrSession.localSrtpKey, cfg->localSrtpKey, SRTP_KEY_LEN);
```

```
stVoipMgrSession.remoteCryptAlg = cfg->negotiated_algorithm;
SETSOCKOPT(VOIP_MGR_SET_SESSION, &stVoipMgrSession, TstVoipMgrSession, 1);
```

Figure 25 SIP programming for setting local master key , remote master key and negotiated algorithm

9.1. SIP over TLS

SIP message will be encrypted over SSL protocol. As shown in Figure 27, the sip protocol information is encrypted as application data and transmitted over SSL layer. According to RFC 3261, the TLS_RSA_WITH_AES_128_CBC_SHA ciphersuite must be supported at a minimum by implementers when TLS is used in a SIP application. The Figure 26 shows the SIP REGISTER message in TLS transport. The sample codes will show how to create SSL connection with SIP server supported TLS. In fact, there are several steps for doing this:

1. Create TCP connection with SIP Server.
2. The sipua acts as client and the SIP server acts as server in SSL handshake.
3. The sipua login the SIP server with its id and password.

```
REGISTER sip:192.168.1.218:5061 SIP/2.0
Via: SIP/2.0/TLS
192.168.1.215:5061;branch=z9hG4bK-d87543-f2542d2c010c3c2a-1--d87543-;rport
Max-Forwards: 70
Contact: <sip:5119@192.168.1.215:5061;rinstance=4f9d531c5a2f9d2b;transport=TLS>
To: "5119"<sip:5119@192.168.1.218:5061>
From: "5119"<sip:5119@192.168.1.218:5061>;tag=740f277d
Call-ID: MTNlNzgwYzM4ZmYzNGMxNDllYTUxMGE5NDVhY2VhOWY.
CSeq: 1 REGISTER
Expires: 3600
Allow: INVITE, ACK, CANCEL, OPTIONS, BYE, REFER, NOTIFY, MESSAGE, SUBSCRIBE,
INFO
User-Agent: sip user agent
Content-Length: 0
```

Figure 26. SIP message for TLS transport

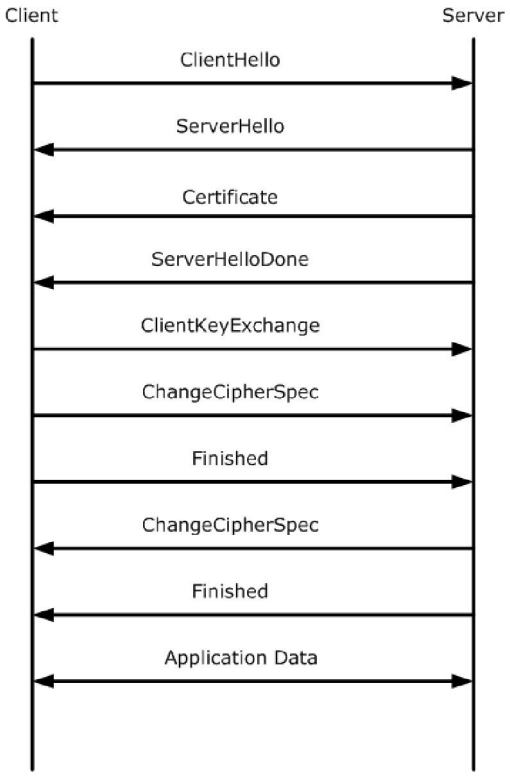

Figure 27. Data flow of SSL

9.2. Secure RTP

The SRTP is defined in RFC3711. The algorithms are AES-128 counter mode and AES-128 f8 mode. For AES counter mode, the key stream is generated by a 128-bin integer which is calculated as follows: (2^{16} x the packet index) XOR (the salting key x 2^{16}) XOR (the SSRC x 2^{64}). Figure 28 shows the key-stream generation process.

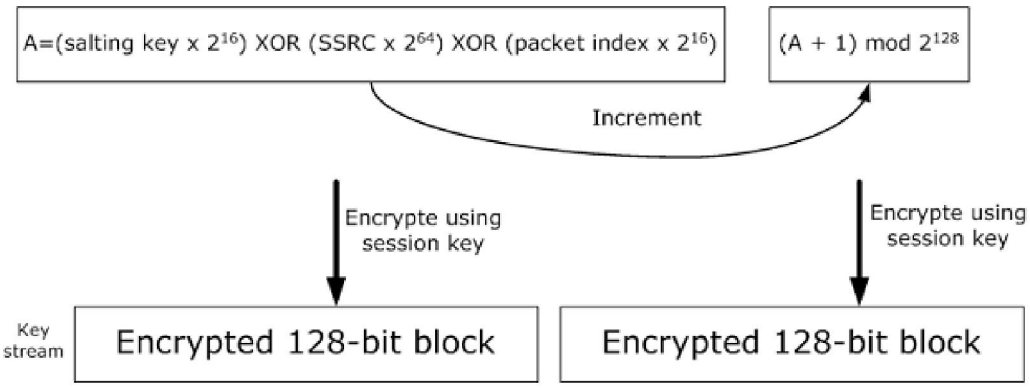

Figure 28 AES counter mode

For AES F8 mode, the XOR of the session key and a salting key is generated, and it is used to encrypt the initialization vector. If the salting key is less than 128 bits in length, it is padded with alternating zeros and one (0x555…) to 128 bits. The result is known as the internal initialization vector. The first block of the key stream is generated as the XOR of the internal initialization vector and a 128-bit variable (j = 0), and the result is encrypted with the session key. The variable j is incremented, and the second block of the key stream is generated as the XOR of the internal initialization vector, the variable j, and the previous block of the key stream. The Figure 29 shows the AES F8 mode.

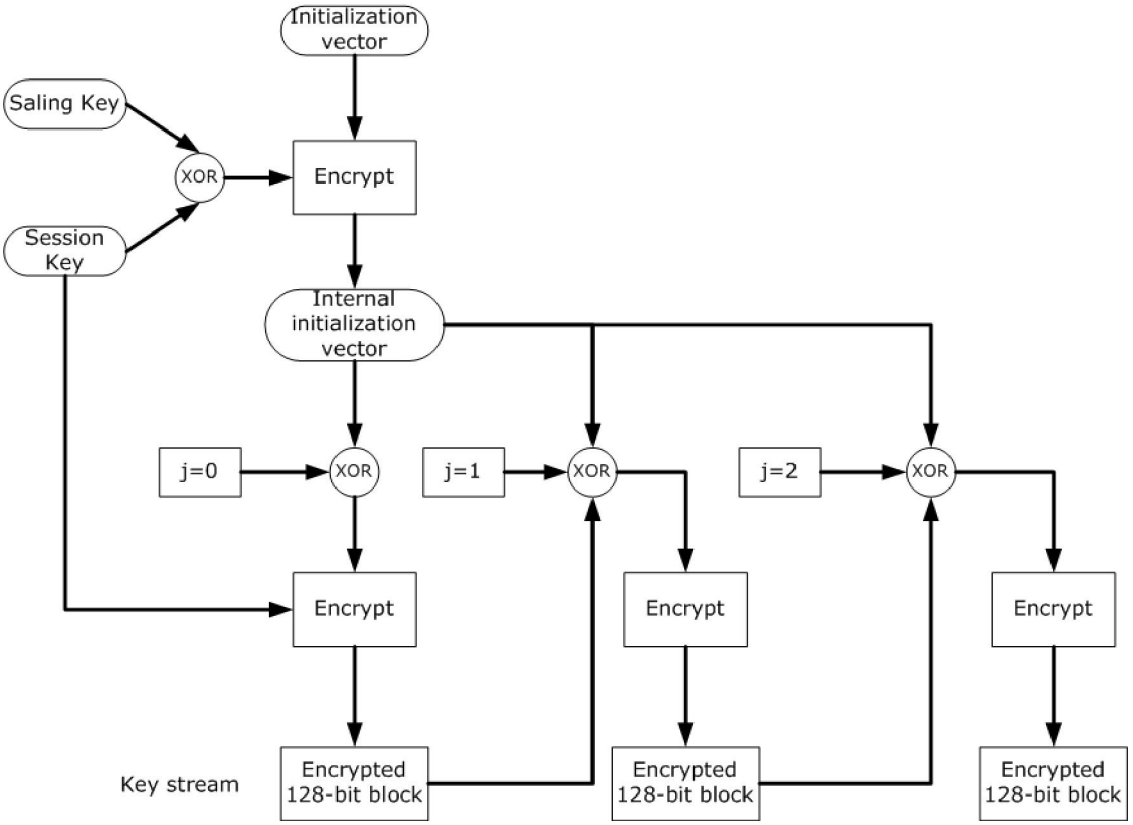

Figure 29. AES F8 Mode

10. VLAN

The RTL89xxc supports port-based, protocol-based, and tagged VLANs. VLAN processing is controlled by a VLAN table and some related register fields. Both VLAN tagged and untagged packets can be handled. A 802.1Q VLAN tag aware or unaware mode con be configured. VLAN tag auto insertion and removal is set via the VLAN table. The VLAN table support up to 4K VLAN entries. User can use following VLAN interfaces to set VLAN

10.1. Vlan Function API

Prototype:	int32 rtl865x_initVlanTable(void);

Description:	Initialize the VLAN environment. The function enables VLAN function, and it should be called before using VLAN function. After VLAN initialization, all VLAN set are empty. The function should be called once.
Parameters:	None
Return Values:	0 success

Prototype:	int32 rtl865x_addVlan(uint16 vid);
Description:	Add a VLAN :The VID should 1 ~ 4095. It only add VLAN without giving its port member. So user should call rtl865x_addVlanPortMember to set port member. User should make sure the VID has not been added, or it will return VID-exist error number.
Parameters:	Vid: vlan id
Return Values:	0 success

Prototype:	int32 rtl865x_delVlan(uint16 vid);
Description:	Delete a VLAN :Delete an existed VLAN and its port member set. If the VLAN does not exit, it will return VID-not-exist error number.
Parameters:	Vid: vlan id
Return Values:	0 success

Prototype:	int32 rtl865x_reinitVlantable(void);
Description:	Reinitialize the VLAN environment :The function will delete all VLAN.
Parameters:	None
Return Values:	0 success

Prototype:	int32 rtl865x_addVlanPortMember(uint16 vid, uint32 portMask);
Description:	Add Port to a VLAN :The VLAN must exit before add it's member port. Port should be 0 ~ 4 bitmask
Parameters:	Vid: vlan id Portmask: port member for special vlan
Return Values:	0 success

Prototype:	int32 rtl865x_delVlanPortMember(uint16 vid,uint32 portMask);
Description:	Delete Port to a VLAN: It only removes some ports from the existed VID's member set, and doesn't effect other ports of the member port. If the removed ports are not in the member port, the function has no effect and returns Success.
Parameters:	Vid: vlan id Portmask: port member for special vlan
Return Values:	0 success

Prototype:	int32 rtl865x_setVlanPortTag(uint16 vid,uint32 portMask,uint8 tag);
Description:	Set tag/untag port :It sets member port as tag or un-tag port depending on tag parameter. The tag port will add a VLAN tag on outgoing packets by it's port VLAN ID.
Parameters:	Vid: vlan id Portmask: port member for special vlan Tag: tag or untag
Return Values:	0 success

Prototype:	int32 rtl865x_setAsicPvid (uint32 portMask , uint16 vid);
Description:	It sets the vlan id for port
Parameters:	Portmask: port member for special vlan Vid: vlan id
Return Values:	0 success

Prototype:	int 32 swNic_setVlanPortTag(int portmask);
Description:	Set tag id for packet from CPU: Add vlan tag for packet without vlan tag from dev->hard_start_xmit.
Parameters:	Portmask: port member
Return Values:	0 success

10.2. Vlan Example

User can echo parameter into /proc/rtk_hw_vlan_support to set swcore to add vlan tag for packets to WAN port. Refenence Code is in “linux-2.6.30/drivers/net/rtl819x/rtl_nic.c.”

ProcName	/proc/rtk_hw_vlan_support
Description	Vlan setting.
Parameters:	1st parameter: Enable vlan tag for packet from CPU or HWNAT to WAN port. 2nd parameter: Vlan id for packets being tagged from CPU or HWNAT to WAN port. 3rd parameter: Enable lan port 1 bridge with WAN port 4th parameter: lan port 1 vlan id when brigding with WAN port 5th parameter: Enable lan port 2 bridge with WAN port 6th parameter: lan port 2 vlan id when brigding with WAN port 7th parameter: Enable lan port 3 bridge with WAN port 8th parameter: lan port 3 vlan id when brigding with WAN port
Cat information	Show vlan setting.

11. Example for Call Features Implementation

11.1. Caller ID Generation Example 1

```
#include "voip_manager.h"
void ShowUsage(char *cmd)
{
	printf("usage: %s <mode> <caller_id> [FSK area] [DTMF mode]\n" \
	"   - mode => 0 is DTMF, 1 is FSK\n" \
		"   - caller_id => caller id string\n" \
		"   - FSK area[2:0] => 0 BELLCORE, 1: ETSI, 2: BT, 3: NTT\n" \
		"   - FSK area[bit7]=> FSK date & time sync\n" \
		"   - FSK area[bit6]=> reverse polarity before caller id (For FSK)\n" \
```

```
            "   - FSK area[bit5]=> short ring before caller id (For FSK)\n" \
            "   - FSK area[bit4]=> dual alert tone before caller id (For FSK)\n" \
            "   - FSK area[bit3]=> caller id Prior Ring (FSK & DTMF)\n" \
            "   - DTMF mode[1:0]=> Start digit, 0:A, 1:B, 2:C, 3:D\n" \
            "   - DTMF mode[3:2]=> End digit, 0:A, 1:B, 2:C, 3:D\n"   \
            "   - DTMF mode[4]=> Auto start/end digit send, 0:suto mode 1:non-auto\n" \
            "         (non-auto mode: DSP send caller ID string only. If caller ID need start/end digits,
                                    developer should add them to caller ID strings.)\n", cmd);
      exit(0);
}

int main(int argc, char *argv[])
{
      unsigned int mode, fsk_area, dtmf_mode;

      if (argc < 3)
      {
            ShowUsage(argv[0]);
      }
      mode = atoi(argv[1]);
      switch (mode)
      {
            case 0:
                  if (argc == 5)
                  {
                        fsk_area = atoi(argv[3]);
                        dtmf_mode = atoi(argv[4]);
                        //for DTMF caller id prior ring or not. & set the caller id start/end digit
                        rtk_Set_CID_DTMF_MODE(0, fsk_area, dtmf_mode);
                        rtk_Gen_Dtmf_CID(0, argv[2]);
                  }
                  else if(argc == 4)
```

```
            {
                fsk_area = atoi(argv[3]);
                dtmf_mode = 0;   // start/end digit : A.
                //for DTMF caller id prior ring or not.
                rtk_Set_CID_DTMF_MODE(0, fsk_area, dtmf_mode);
                rtk_Gen_Dtmf_CID(0, argv[2]);
            }
            else if(argc == 3)
            {
                if (argv[2] == 1)
                {
                    // for DTMF caller id priori 1st ring, auto send start/end digit
                    rtk_Set_CID_DTMF_MODE(0, 0x8, 0x8);
                    rtk_Gen_Dtmf_CID(0, "654321");
                }
                else if (argv[2] == 2)
                {
                    // for DTMF caller id after 1st ring, non-auto
                    rtk_Set_CID_DTMF_MODE(0, 0, 0x10);
                    // DSP send caller ID string only. If caller ID need start/end digits,
developer should add them to caller ID strings
                    rtk_Gen_Dtmf_CID(0, "A654321C");
                }
            }

        break;
        case 1:
            if (argc == 4)
            {
                fsk_area = atoi(argv[3]);
                if ((fsk_area&7) < CID_DTMF)
                {
```

```
                        /*if ((fsk_area&7) == CID_FSK_NTT)
                        **{
                        **   printf("not support NTT area currently\n");
                        **   return 0;
                        **}
                        */
                        rtk_Set_FSK_Area(0, fsk_area);
                    }
                    else
                        ShowUsage(argv[0]);
                }
                rtk_Gen_FSK_CID(0, argv[2], (void *) 0, (void *) 0, 0); // FSK TypeI
            break;
            default:
                ShowUsage(argv[0]);
        }
        rtk_SetRingFXS(0, 1);
        sleep(1);
        rtk_SetRingFXS(0, 0);

        return 0;
}
```

11.2. Caller ID Generation Example 2

```
#include "voip_manager.h"

int fskgen_main(int argc, char *argv[])
{
    if (argc == 7)
    {
        if (atoi(argv[2]) == 1)//type I
        {
```

```
                /* area -> 0:Bellcore 1:ETSI 2:BT 3:NTT */
                rtk_Set_FSK_Area(atoi(argv[1])/*chid*/, atoi(argv[3])/*area*/);
#if 0       // This case is workable. Call rtk_Gen_FSK_CID() to send FSK Caller ID.
                // API to gen FSK caller ID and auto ring by DSP
                rtk_Gen_FSK_CID(atoi(argv[1]), argv[4]/*cid*/, argv[6]/*date_time*/,
                                argv[5]/*name*/, 0);
#elif 0     // This case is workable. Call rtk_Gen_CID_And_FXS_Ring () to send FSK Caller ID.
                // API to gen caller ID and auto ring by DSP
                rtk_Gen_CID_And_FXS_Ring(atoi(argv[1]), 1/*fsk*/, argv[4], argv[6] /*date_time*/,
                        argv[5]/*name*/, 0/*type1*/, 0);
#else       // This case is workable. Call rtk_Gen_MDMF_FSK_CID () to send FSK Caller ID.

                TstFskClid clid;
                clid.ch_id = atoi(argv[1]);
                clid.service_type = 0; //service type 1

                clid.cid_data[0].type = 0x01;
                strcpy(clid.cid_data[0].data, argv[6]);    //DATE

                if ( (argv[4][0] == 'P') || (argv[4][0] == 'O') )        //Private or Out of area
                      clid.cid_data[1].type = 0x04;
                else
                      clid.cid_data[1].type = 0x02;
                strcpy(clid.cid_data[1].data, argv[4]);    //CLI

                if ( (argv[5][0] == 'P') || (argv[5][0] == 'O') )        //Private or Out of area
                      clid.cid_data[2].type = 0x08;
                else
                      clid.cid_data[2].type = 0x07;
                strcpy(clid.cid_data[2].data, argv[5]);    //CLI_NAME

                //Only 3 elements for Caller ID data. Set other element to 0 (MUST)
```

```
                clid.cid_data[3].type = 0;
                clid.cid_data[4].type = 0;

                rtk_Gen_MDMF_FSK_CID(atoi(argv[1])/*chid*/, &clid, 3);
#endif
        }
        else if (atoi(argv[2]) == 2)//type II
        {
                //DSP default is soft gen, no need to set soft gen mode
                rtk_Set_FSK_Area(atoi(argv[1])/*chid*/, atoi(argv[3])/*area*/);      /* area -> 0:Bellcore
1:ETSI 2:BT 3:NTT */
#if 0
                rtk_Gen_FSK_CID(atoi(argv[1])/*chid*/, argv[4]/*cid*/, argv[6], argv[5]/*name*/, 1);
#elif 0
                rtk_Gen_CID_And_FXS_Ring(atoi(argv[1]), 1/*fsk*/, argv[4], argv[6],
argv[5]/*name*/, 1/*type1*/, 0);
#else
                TstFskClid clid;

                clid.ch_id = atoi(argv[1]);
                        clid.service_type = 1;    //service type 2

                        clid.cid_data[0].type = 0x01;
                        strcpy(clid.cid_data[0].data, argv[6]);    //DATE

                        if ( (argv[4][0] == 'P') || (argv[4][0] == 'O') )
                        clid.cid_data[1].type = 0x04;
                        else
                                clid.cid_data[1].type = 0x02;
                        strcpy(clid.cid_data[1].data, argv[4]);    //CLI

                                if ( (argv[5][0] == 'P') || (argv[5][0] == 'O') )
```

```
                        clid.cid_data[2].type = 0x08;
                    else
                        clid.cid_data[2].type = 0x07;
                    strcpy(clid.cid_data[2].data, argv[5]);    //CLI_NAME

                    clid.cid_data[3].type = 0;
                    clid.cid_data[4].type = 0;

                rtk_Gen_MDMF_FSK_CID(atoi(argv[1])/*chid*/, &clid, 3);
#endif
            }
            else
                printf("wrong fsk type: should be type-I(1) or type-II(2)\n");
        }
        else if (argc == 5)// no name, date, time
        {
            if (atoi(argv[2]) == 1)//type I
            {
                /* area -> 0:Bellcore 1:ETSI 2:BT 3:NTT */
                rtk_Set_FSK_Area(atoi(argv[1])/*chid*/, atoi(argv[3])/*area*/);
#if 0
                rtk_Gen_FSK_CID(atoi(argv[1]), argv[4]/*cid*/, 0/*date*/, 0/*name*/, 0);
#elif 0
                rtk_Gen_CID_And_FXS_Ring(atoi(argv[1]), 1/*fsk*/, argv[4], 0/*date*/, 0/*name*/,
                        0/*type1*/, 0);
#else
                TstFskClid clid;

                clid.ch_id = atoi(argv[1]);
                clid.service_type = 0;    //service type 1

                if ( (argv[4][0] == 'P') || (argv[4][0] == 'O') )
```

```
                clid.cid_data[0].type = 0x04;
            else
                clid.cid_data[0].type = 0x02;
            strcpy(clid.cid_data[0].data, argv[4]);    //CLI

            clid.cid_data[1].type = 0;
            clid.cid_data[2].type = 0;
            clid.cid_data[3].type = 0;
            clid.cid_data[4].type = 0;

            rtk_Gen_MDMF_FSK_CID(atoi(argv[1])/*chid*/, &clid, 1);
#endif
        }
        else if (atoi(argv[2]) == 2)//type II
        {
            /* area -> 0:Bellcore 1:ETSI 2:BT 3:NTT */
            rtk_Set_FSK_Area(atoi(argv[1])/*chid*/, atoi(argv[3])/*area*/);
#if 0
            rtk_Gen_FSK_CID(atoi(argv[1])/*chid*/, argv[4]/*cid*/, 0/*date*/, 0/*name*/, 1);
#elif 0
            rtk_Gen_CID_And_FXS_Ring(atoi(argv[1]), 1/*fsk*/, argv[4], 0/*date*/, 0/*name*/,
                    1/*type1*/, 0);
#else
            TstFskClid clid;

            clid.ch_id = atoi(argv[1]);
            clid.service_type = 1;      //service type 2

            if ( (argv[4][0] == 'P') || (argv[4][0] == 'O') )
                clid.cid_data[0].type = 0x04;
            else
                clid.cid_data[0].type = 0x02;
```

```
                strcpy(clid.cid_data[0].data, argv[4]);

                clid.cid_data[1].type = 0;
                clid.cid_data[2].type = 0;
                clid.cid_data[3].type = 0;
                clid.cid_data[4].type = 0;

                rtk_Gen_MDMF_FSK_CID(atoi(argv[1])/*chid*/, &clid, 1);
#endif
            }
            else
                printf("wrong fsk type: should be type-I(1) or type-II(2)\n");
        }
        else if (argc == 11)
        {
            TstVoipFskPara para;

            if (argv[1][0] == 's') //set para
            {
                para.ch_id = atoi(argv[2]);
                para.area = atoi(argv[3]);
                para.CS_cnt = atoi(argv[4]);
                para.mark_cnt = atoi(argv[5]);
                para.mark_gain = atoi(argv[6]);
                para.space_gain = atoi(argv[7]);
                para.type2_expected_ack_tone = argv[8][0];
                para.delay_after_1st_ring = atoi(argv[9]);
                para.delay_before_2nd_ring = atoi(argv[10]);

                rtk_Set_FSK_CLID_Para(&para);
            }
        }
```

格式化：義大利文（義大利）

```
    else if (argc == 4)
    {
        TstVoipFskPara para;

        if (argv[1][0] == 'g') //get para
        {
            para.ch_id = atoi(argv[2]);
            para.area = atoi(argv[3]);

            rtk_Get_FSK_CLID_Para(&para);

            printf("Get parameters:\n");
            printf("================================\n");
            printf(" - ch seizure cnt = %d\n", para.CS_cnt);
            printf(" - mark cnt = %d\n", para.mark_cnt);
            printf(" - mark gain = %d\n", para.mark_gain);
            printf(" - space gain = %d\n", para.space_gain);
            printf(" - type2_expected_ack_tone = %c\n", para.type2_expected_ack_tone);
            printf(" - delay_after_1st_ring = %d ms\n", para.delay_after_1st_ring);
            printf(" - delay_before_2nd_ring = %d ms\n", para.delay_before_2nd_ring);
            printf("================================\n");
        }
    }
    else
    {
        printf("Error! Usage:\n");
        printf("To send Caller ID ==> fskgen 'chid' 'fsk_type' 'fsk_area' 'caller_id' 'name'
'date_time'\n");
        printf("To set parameters ==> fskgen set 'chid' 'fsk_area' 'ch seizure cnt' 'mark cnt' 'mark gain'
'space gain' 'type-2 expexted ack tone' 'delay_after_1st_ring' 'delay_before_2nd_ring'\n");
        printf("To set parameters ==> fskgen get 'chid' 'fsk_area'\n");
```

```
        printf("Example for Type 1:\n");
        printf("'fskgen 0 1 128 035780211 tester_B 01020304' will display number 035780211 and name, date, and time.(Bellcore)\n");
        printf("'fskgen 0 1 129 035780211 tester_E 04030201' will display number 035780211 and name, date, and time.(ETSI)\n");
        printf("'fskgen 0 1 131 035780211 tester_N 06020602' will display number 035780211 and name, date, and time.(NTT)\n");
        printf("'fskgen 0 1 0 035780211' will just display number 035780211 without name, date, and time.(Bellcore)\n");
        printf("'fskgen 0 1 1 035780211' will just display number 035780211 without name, date, and time.(ETSI)\n");
        printf("'fskgen 0 1 3 035780211' will just display number 035780211 without name, date, and time.(NTT)\n");

        printf("Example for set Bellcore parameters:\n");
        printf("    fskgen set 0 0 300 180 8 8 D 150 1000\n");
        printf("Example for get ETSI parameters:\n");
        printf("    fskgen get 0 1\n\n\n");

        printf("Note:\n");
        printf(" - fsk_type: 1 -> type1, 2 -> type2\n");
        printf(" - fsk_area: 0 -> Bellcore, 1 -> ETSI, 2 -> BT, 3 -> NTT\n");
        printf(" - gain 0 ~ gain 8: 8dB ~ 0dB\n");
        printf(" - gain 9 ~ gain 24: -1dB ~ -16dB\n");
        printf(" - type-2 expexted ack tone: character A or B or C or D\n");

    }


    return 0;
}
```

11.3. RFC2833 Send

```
#include <sys/time.h>
#include "voip_manager.h"

int send_2833_main(int argc, char *argv[])
{
	if (argc == 5)
	{
		if (argv[1][0] = 'l')
			rtk_LimitMaxRfc2833DtmfDuration(atoi(argv[2]), atoi(argv[3]), atoi(argv[4]));
		else
			rtk_SetRTPRFC2833(atoi(argv[1]), atoi(argv[2]), atoi(argv[3]), atoi(argv[4]));
	}
	else if (argc == 3)
	{
		rtk_SetRFC2833SendByAP(atoi(argv[1]), atoi(argv[2]));
	}
	else if (argc == 4)
	{
		rtk_SetRFC2833TxConfig(atoi(argv[1]), atoi(argv[2]), atoi(argv[3]));
	}
	else
	{
		printf("****** Usage ******\n");
		printf("'send_2833 chid 0' to set RFC2833 sent by DSP.\n");
		printf("'send_2833 chid 1' to set RFC2833 sent by application.\n");
		printf("'send_2833 chid 10 0' set TX volume to -10 dBm and set RFC2833 sent by DSP.\n");
		printf("'send_2833 chid 5 1' set TX volume to -5 dBm and set RFC2833 sent by
application.\n");
```

```
        printf("'send_2833 chid sid dtmf_event duration' to send RFC2833 event\n");
        printf("Note: above tests are only workable when you set to RFC2833 mode, and during VoIP call\n");
        printf ("'send_2833 limit chid duration(ms) bEnable(0 or 1)' to config RFC2833 DTMF duation max. limitation\n");
    }
    return 0;
}
```

Note: Application **MUST** set the proper RTP config via API rtk_SetRtpConfig() first to let RFC2833 mode work fine. See here for detail description.

11.4. Call Conference

```
#include "voip_manager.h"

#define MAX_SESSION   (SESS_NUM / VOIP_CH_NUM)

enum {
    STATE_IDLE = 0,
    STATE_RTP
};

int main(int argc, char *argv[])
{
    int chid, ssid;
    SIGNSTATE val;
    rtp_config_t rtp_config[SLIC_CH_NUM][MAX_SESSION];
    payloadtype_config_t codec_config[SLIC_CH_NUM][MAX_SESSION];
    TstVoipMgr3WayCfg stVoipMgr3WayCfg;
    int ActiveSession[SLIC_CH_NUM];
    int session_state[SLIC_CH_NUM][MAX_SESSION];
```

```
int dtmf_val[SIGN_HASH + 1] = {0,1,2,3,4,5,6,7,8,9,0,10,11};
char dial_val[SIGN_HASH + 1] = {'\0', '1', '2', '3', '4', '5', '6', '7', '8', '9', '0', '.', '#'};
char dial_code[SLIC_CH_NUM][256];
int dial_index[SLIC_CH_NUM];
int i, j;
char *src_ip;
int   src_port;

if (argc == 2)
{
      src_ip = argv[1];
      src_port = 9000;
}
else if (argc == 3)
{
      src_ip = argv[1];
      src_port = atoi(argv[2]);
}
else
{
      printf("usage: %s src_ip [src_port]\n", argv[0]);
      return 0;
}

// init
for (i=0; i<SLIC_CH_NUM; i++)
{
      rtk_InitDSP(i);
      rtk_Set_Flash_Hook_Time(i, 0, 300);            // 0~300 ms
      rtk_Set_flush_fifo(i);                         // flush kernel fifo before app run
      rtk_SetDTMFMODE(i, DTMF_RFC2833);              // set dtmf mode
      ActiveSession[i] = 0;
```

```
            for (j=0; j<MAX_SESSION; j++)
            {
                session_state[i][j] = STATE_IDLE;
            }
        }

main_loop:

        for (chid=0; chid<SLIC_CH_NUM; chid++)
        {
            rtk_GetFxsEvent(chid, &val);
            switch (val)
            {
            case SIGN_KEY1:
            case SIGN_KEY2:
            case SIGN_KEY3:
            case SIGN_KEY4:
            case SIGN_KEY5:
            case SIGN_KEY6:
            case SIGN_KEY7:
            case SIGN_KEY8:
            case SIGN_KEY9:
            case SIGN_KEY0:
            case SIGN_STAR:
            case SIGN_HASH:
            {
                ssid = ActiveSession[chid];
                if (session_state[chid][ssid] == STATE_RTP)
                {
                    rtk_SetRTPRFC2833(chid, ssid, dtmf_val[val], 100);
                }
                else if (val == SIGN_HASH)
```

```
{
    char *p, *ip, *pPort;
    char *dest_ip;
    int dest_port;

    dial_code[chid][dial_index[chid]] = 0;

    // 192.168.1.1..9002 => ip: 192.168.1.1, rtp port:9002
    pPort = strstr(dial_code[chid], "..");
    if (pPort)
    {
        *pPort = 0;
        pPort += 2;
        dest_ip = dial_code[chid];
        dest_port = atoi(pPort);
    }
    else
    {
        dest_ip = dial_code[chid];
        dest_port = 9000;
    }

    // set rtp session
    rtp_config[chid][ssid].chid = chid;          // use channel 0
    rtp_config[chid][ssid].sid = ssid;
    rtp_config[chid][ssid].isTcp = 0;       // use udp
    rtp_config[chid][ssid].extIp = inet_addr(src_ip);
    rtp_config[chid][ssid].remIp = inet_addr(dest_ip);
    rtp_config[chid][ssid].extPort = htons(src_port + chid * 2);
    rtp_config[chid][ssid].remPort = htons(dest_port);
    rtp_config[chid][ssid].tos = 0;
    rtp_config[chid][ssid].rfc2833_payload_type_local = 96;
```

```
                rtp_config[chid][ssid].rfc2833_payload_type_remote = 96;
                rtk_SetRtpConfig(&rtp_config[chid][ssid]);

                // set rtp payload, and other session parameters.
                codec_config[chid][ssid].chid = chid;       // use channel 0
                codec_config[chid][ssid].sid = ssid;
                codec_config[chid][ssid].local_pt = 0;              // PCMU use pt 0
                codec_config[chid][ssid].remote_pt = 0;         // PCMU use pt 0
                codec_config[chid][ssid].uPktFormat = rtpPayloadPCMU;
                codec_config[chid][ssid].nG723Type = 0;       // 6.3K
                codec_config[chid][ssid].nFramePerPacket = 1;
                codec_config[chid][ssid].bVAD = 0;
                codec_config[chid][ssid].nJitterDelay = 4;
                codec_config[chid][ssid].nMaxDelay = 13;
                codec_config[chid][ssid].nJitterFactor = 7;
                codec_config[chid][ssid].nG726Packing = 2;    //pack right
                rtk_SetRtpPayloadType(&codec_config[chid][ssid]);

                // start rtp (channel number = 0, session number = 0)
                rtk_SetRtpSessionState(chid, ssid, rtp_session_sendrecv);

                session_state[chid][ssid] = STATE_RTP;

                printf("%s:%d -> %s:%d\n",
                    src_ip, src_port + chid * 2, dest_ip, dest_port);
            }
            else
            {
                // stop dial tone in current session
                rtk_SetPlayTone(chid, ssid, DSPCODEC_TONE_HOLD, 0,
DSPCODEC_TONEDIRECTION_LOCAL);
                dial_code[chid][dial_index[chid]++] = dial_val[val];
```

```
                }
                break;
            }

            case SIGN_OFFHOOK:
            {
                // do rtk_SLIC_Offhook_Action at first
                rtk_Offhook_Action(chid);
                rtk_SetTranSessionID(chid, 0);
                ActiveSession[chid] = 0;
                dial_index[chid] = 0;
                for (i=0; i<MAX_SESSION; i++)
                {
                    session_state[chid][i] = STATE_IDLE;
                }
                rtk_SetPlayTone(chid, 0, DSPCODEC_TONE_DIAL, 1,
DSPCODEC_TONEDIRECTION_LOCAL);    // play dial tone
                break;
            }

            case SIGN_ONHOOK:
                for (i=0; i<MAX_SESSION; i++)
                {
                    rtk_SetRtpSessionState(chid, i, rtp_session_inactive);
                    rtk_SetPlayTone(chid, i, DSPCODEC_TONE_HOLD, 0,
DSPCODEC_TONEDIRECTION_LOCAL); // stop tone
                }
                // clear resource count
                rtk_SetTranSessionID(chid, 255);
                // close conference
                memset(&stVoipMgr3WayCfg, 0, sizeof(stVoipMgr3WayCfg));
                stVoipMgr3WayCfg.ch_id = chid;
```

```
            stVoipMgr3WayCfg.enable = 0;
            rtk_SetConference(&stVoipMgr3WayCfg);
            // do rtk_SLIC_Onhook_Action at last
            rtk_Onhook_Action(chid);
            break;

        case SIGN_FLASHHOOK:
        {
            dial_index[chid] = 0;
            ssid = ActiveSession[chid];
            if (session_state[chid][ssid] == STATE_RTP)
            {
                // check another session state
                if (session_state[chid][!ssid] == STATE_IDLE)
                {
                    // hold current session
                    rtk_Hold_Rtp(chid, ssid, 1);
                    // change to new session
                    ssid = !ssid;
                    ActiveSession[chid] = ssid;
                    rtk_SetTranSessionID(chid, ssid);
                    // play dial tone in new session
                    rtk_SetPlayTone(chid, ssid, DSPCODEC_TONE_DIAL, 1,
DSPCODEC_TONEDIRECTION_LOCAL);
                }
                else
                {
                    // resume another session
                    rtk_Hold_Rtp(chid, !ssid, 0);
                    // do conference
                    memset(&stVoipMgr3WayCfg, 0, sizeof(stVoipMgr3WayCfg));
                    stVoipMgr3WayCfg.ch_id = chid;
```

```
                    stVoipMgr3WayCfg.enable = 1;
                    for (i=0; i<MAX_SESSION; i++)
                    {
                        stVoipMgr3WayCfg.rtp_cfg[i].ch_id = chid;
                        stVoipMgr3WayCfg.rtp_cfg[i].m_id = rtp_config[chid][i].sid;
                        stVoipMgr3WayCfg.rtp_cfg[i].ip_src_addr = rtp_config[chid][i].remIp;
                        stVoipMgr3WayCfg.rtp_cfg[i].ip_dst_addr = rtp_config[chid][i].extIp;
                        stVoipMgr3WayCfg.rtp_cfg[i].udp_src_port =
rtp_config[chid][i].remPort;
                        stVoipMgr3WayCfg.rtp_cfg[i].udp_dst_port =
rtp_config[chid][i].extPort;
                        stVoipMgr3WayCfg.rtp_cfg[i].tos = rtp_config[chid][i].tos;
                        stVoipMgr3WayCfg.rtp_cfg[i].rfc2833_payload_type_local =
                            rtp_config[chid][i].rfc2833_payload_type_local;
                        stVoipMgr3WayCfg.rtp_cfg[i].rfc2833_payload_type_remote =
                            rtp_config[chid][i].rfc2833_payload_type_remote;
                        stVoipMgr3WayCfg.rtp_cfg[i].local_pt =
codec_config[chid][i].local_pt;
                        stVoipMgr3WayCfg.rtp_cfg[i].remote_pt =
codec_config[chid][i].remote_pt;
                        stVoipMgr3WayCfg.rtp_cfg[i].uPktFormat =
codec_config[chid][i].uPktFormat;
                        stVoipMgr3WayCfg.rtp_cfg[i].nG723Type =
codec_config[chid][i].nG723Type;
                        stVoipMgr3WayCfg.rtp_cfg[i].nFramePerPacket =
codec_config[chid][i].nFramePerPacket;
                        stVoipMgr3WayCfg.rtp_cfg[i].bVAD = codec_config[chid][i].bVAD;
                        stVoipMgr3WayCfg.rtp_cfg[i].nJitterDelay =
codec_config[chid][i].nJitterDelay;
                        stVoipMgr3WayCfg.rtp_cfg[i].nMaxDelay =
codec_config[chid][i].nMaxDelay;
                        stVoipMgr3WayCfg.rtp_cfg[i].nJitterFactor =
```

```
codec_config[chid][i].nJitterFactor;
                        }
                        rtk_SetConference(&stVoipMgr3WayCfg);
                    }
                }
                else
                {
                    if (session_state[chid][!ssid] == STATE_RTP)
                    {
                        // stop dial tone in current session
                        rtk_SetPlayTone(chid, ssid, DSPCODEC_TONE_HOLD, 0,
DSPCODEC_TONEDIRECTION_LOCAL);
                        // change to old session
                        ssid = !ssid;
                        ActiveSession[chid] = ssid;
                        rtk_SetTranSessionID(chid, ssid);
                        // resume old session
                        rtk_Hold_Rtp(chid, ssid, 0);
                    }
                    else
                    {
                        // do nothing if no rtp session exist
                    }
                }
                break;
            }

            default:
                break;
            }

            usleep(100000 / VOIP_CH_NUM); // 100ms
```

```
    }

    goto main_loop;

    return 0;
}
```

11.5. FXO

```
#include "voip_manager.h"

// test option:
//    - FXO_DIAL_FXS0 has defined
//        * fxo ring will dial fxs0 automatically
//    - FXO_DIAL_FXS0 has not defined
//        * fxo ring will prompt user dial voip number via IVR
#define FXO_DIAL_FXS0_AUTO

#define FUNCKEY_FXO_0     ".0"
#define FUNCKEY_FXS_0     ".1"
#define FUNCKEY_FXS_1     ".2"

#define CHANNEL_IS_FXS(chid)    ((chid) < SLIC_CH_NUM)
#define CHANNEL_IS_FXO(chid)    ((chid) >= SLIC_CH_NUM)

#define NULL      ((void *) 0)

enum {
    FXO_NO_DAA = 0,
    FXO_VIRTUAL_DAA,
    FXO_DAA
};
```

格式化 :義大利文 (義大利)

```
enum {
	STATE_IDLE = 0,
	STATE_RING,
	STATE_CONNECT,
	STATE_DAA_RING,		// virtual daa state
	STATE_DAA_CONNECT	// virtual daa state
};

enum {
	DIAL_NONE = 0,
	DIAL_FXO_0,
	DIAL_FXS_0,
	DIAL_FXS_1,
	DIAL_OTHER,
};

typedef struct channel_s channel_t;

struct channel_s {
	// phone info
	int chid;
	int state;
	char dial_code[101];
	int dial_index;
	// rtp info
	unsigned long ip;
	unsigned short port;
	// remote channel (just peer simulation)
	channel_t *remote;
};

void channel_init(channel_t *channel, int chid);
```

```
void channel_reset(channel_t *channel, int chid);
int channel_input(channel_t *channel, SIGNSTATE val);
void channel_dial_local(channel_t *channel, channel_t *remote_channel);
void channel_dial_out(channel_t *channel);
void channel_switch_to_fxo(channel_t *channel); // virtual daa
void channel_rtp_start(channel_t *channel);
void channel_rtp_stop(channel_t *channel);

// global variable
int fxo_type;
channel_t channels[VOIP_CH_NUM];

void channel_init(channel_t *channel, int chid)
{
	memset(channel, 0, sizeof(*channel));
	// phone
	channel->chid = chid;
	channel->state = STATE_IDLE;
	channel->dial_code[0] = 0;
	channel->dial_index = 0;
	// rtp
	channel->ip = ntohl(inet_addr("127.0.0.1"));
	channel->port = 9000 + chid;
	// reomte
	channel->remote = NULL;
}

int channel_input(channel_t *channel, SIGNSTATE val)
{
	if (channel->dial_index >= sizeof(channel->dial_code) - 1)
	{
		printf("%s: out of buffer\n", __FUNCTION__);
```

```
            return DIAL_NONE;
        }

        if (val >= SIGN_KEY1 && val <= SIGN_KEY9)
            channel->dial_code[channel->dial_index++] = (char) '0' + val;
        else if (val == SIGN_KEY0)
            channel->dial_code[channel->dial_index++] = '0';
        else if (val == SIGN_STAR)
            channel->dial_code[channel->dial_index++] = '.';
        else if (val == SIGN_HASH)
            return DIAL_OTHER;

        channel->dial_code[channel->dial_index] = 0;

        // check local dial
        if (strcmp(channel->dial_code, FUNCKEY_FXO_0) == 0)
            return DIAL_FXO_0;
        else if (strcmp(channel->dial_code, FUNCKEY_FXS_0) == 0)
            return DIAL_FXS_0;
        else if (strcmp(channel->dial_code, FUNCKEY_FXS_1) == 0)
            return DIAL_FXS_1;

        return DIAL_NONE;
}

void channel_dial_local(channel_t *channel, channel_t *remote_channel)
{
        if (remote_channel->chid == channel->chid)
        {
            printf("%s: couldn't dial itself\n", __FUNCTION__);
            rtk_SetPlayTone(channel->chid, 0, DSPCODEC_TONE_BUSY, 1,
DSPCODEC_TONEDIRECTION_LOCAL);
```

格式化：義大利文（義大利）

格式化：義大利文（義大利）

格式化：義大利文（義大利）

```
        return;
    }

    // 1. create local rtp
    channel->remote = remote_channel;
    channel_rtp_start(channel);
    channel->state = STATE_CONNECT;

    // 2. signal remote channel
    remote_channel->remote = channel;

    if (CHANNEL_IS_FXS(remote_channel->chid))
    {
        rtk_SetPlayTone(channel->chid, 0, DSPCODEC_TONE_RINGING, 1,
DSPCODEC_TONEDIRECTION_LOCAL);
        // signal fxs
        rtk_SetRingFXS(remote_channel->chid, 1);
    }
    else
    {
        // signal fxo
        rtk_FXO_RingOn(remote_channel->chid);
    }

    remote_channel->state = STATE_RING;
}

void channel_dial_out(channel_t *channel)
{
    printf("%s: dial %s done.\n", __FUNCTION__, channel->dial_code);
    rtk_SetPlayTone(channel->chid, 0, DSPCODEC_TONE_BUSY, 1,
DSPCODEC_TONEDIRECTION_LOCAL);
```

格式化 :英文 (美國)

```
}

void channel_switch_to_fxo(channel_t *channel)
{
	if (rtk_DAA_off_hook(channel->chid) == 0)
	{
		printf("%s: ok.\n", __FUNCTION__);
	}
	else // pstn out failed
	{
		printf("%s: failed.\n", __FUNCTION__);
		rtk_SetPlayTone(channel->chid, 0, DSPCODEC_TONE_BUSY, 1,
DSPCODEC_TONEDIRECTION_LOCAL);
	}
}

void channel_rtp_start(channel_t *channel)
{
	rtp_config_t rtp_config;
	payloadtype_config_t codec_config;

	// set rtp session
	rtp_config.chid = channel->chid;
	rtp_config.sid = 0;
	rtp_config.isTcp = 0;		// use udp
	rtp_config.extIp = htonl(channel->ip);
	rtp_config.remIp = htonl(channel->remote->ip);
	rtp_config.extPort = htons(channel->port);
	rtp_config.remPort = htons(channel->remote->port);
	rtp_config.tos = 0;
	rtp_config.rfc2833_payload_type_local = 0;		// use in-band
	rtp_config.rfc2833_payload_type_remote = 0;		// use in-band
```

```
	rtk_SetRtpConfig(&rtp_config);

	// set rtp payload, and other session parameters.
	codec_config.chid = channel->chid;
	codec_config.sid = 0;
	codec_config.local_pt = 0;
	codec_config.remote_pt = 0;
	codec_config.uPktFormat = rtpPayloadPCMU;
	codec_config.nG723Type = 0;
	codec_config.nFramePerPacket = 1;
	codec_config.bVAD = 0;
	codec_config.nJitterDelay = 4;
	codec_config.nMaxDelay = 13;
	codec_config.nJitterFactor = 7;
	codec_config.nG726Packing = 2;//pack right
	rtk_SetRtpPayloadType(&codec_config);

	// start rtp
	rtk_SetRtpSessionState(channel->chid, 0, rtp_session_sendrecv);
}

void channel_rtp_stop(channel_t *channel)
{
	rtk_SetRtpSessionState(channel->chid, 0, rtp_session_inactive);
}

void channel_handle_input(channel_t *channel, SIGNSTATE val)
{
	int ret;

	rtk_SetPlayTone(channel->chid, 0, DSPCODEC_TONE_HOLD, 0,
DSPCODEC_TONEDIRECTION_LOCAL);
```

```
    ret = channel_input(channel, val);
    switch (ret)
    {
    case DIAL_FXO_0:
        if (fxo_type == FXO_NO_DAA)
        {
            printf("%s: no fxo\n", __FUNCTION__);
            rtk_SetPlayTone(channel->chid, 0, DSPCODEC_TONE_BUSY, 1,
DSPCODEC_TONEDIRECTION_LOCAL);
            return;
        }
        if (fxo_type == FXO_VIRTUAL_DAA)
        {
            channel_switch_to_fxo(channel);                    // switch to fxo
        }
        else
        {
            channel_dial_local(channel, &channels[SLIC_CH_NUM]);  // dial fxo 0
        }
        break;
    case DIAL_FXS_0:
        channel_dial_local(channel, &channels[0]);    // dial fxs 0
        break;
    case DIAL_FXS_1:
        if (SLIC_CH_NUM < 2)
        {
            printf("%s: no fxs 1\n", __FUNCTION__);
            rtk_SetPlayTone(channel->chid, 0, DSPCODEC_TONE_BUSY, 1,
DSPCODEC_TONEDIRECTION_LOCAL);
            return;
        }
        channel_dial_local(channel, &channels[1]);    // dial fxs 1
```

```
            break;
        case DIAL_OTHER:
            channel_dial_out(channel);
            break;
        default:
            // nothing
            break;
        }
}

void channel_handle_onhook(channel_t *channel)
{
        switch (channel->state)
        {
        case STATE_CONNECT:
            channel_rtp_stop(channel);
            if (CHANNEL_IS_FXS(channel->chid))
            {
                // fxs
            }
            else
            {
                // daa
                rtk_FXO_onhook(channel->chid);
            }

            // check remote channel is terminated
            if (channel->remote->remote == NULL)
                break;

            // signal remote channel
            if (CHANNEL_IS_FXS(channel->remote->chid))
```

```
            {
                rtk_SetPlayTone(channel->remote->chid, 0, DSPCODEC_TONE_BUSY, 1,
DSPCODEC_TONEDIRECTION_LOCAL);
            }
            else
            {
                rtk_FXO_Busy(channel->remote->chid);
            }
            break;
        case STATE_DAA_CONNECT:
            // virtual daa
            rtk_DAA_on_hook(channel->chid);
            break;
        default:
            break;
        }

#ifdef CONFIG_RTK_VOIP_IVR
        rtk_IvrStopPlaying(channel->chid); // clear ivr
#endif
        rtk_SetTranSessionID(channel->chid, 255);   // clear resource count
        rtk_SetPlayTone(channel->chid, 0, DSPCODEC_TONE_HOLD, 0,
DSPCODEC_TONEDIRECTION_LOCAL);    // clear tone
        channel_init(channel, channel->chid);
}

void channel_handle_offhook(channel_t *channel)
{
        // active session 0, and record resource + 1
        rtk_SetTranSessionID(channel->chid, 0);

        switch (channel->state)
```

```
    {
    case STATE_IDLE:
        if (CHANNEL_IS_FXS(channel->chid))
        {
            // fxs
            rtk_SetPlayTone(channel->chid, 0, DSPCODEC_TONE_DIAL, 1,
DSPCODEC_TONEDIRECTION_LOCAL);
        }
        else
        {
            char caller_id[21];
            char caller_date[9];
            char caller_name[51];

        #ifdef FXO_DIAL_FXS0_AUTO
            rtk_Get_DAA_CallerID(channel->chid, caller_id, caller_date, caller_name);
            if (caller_id[0])
                printf("caller id is %s\n", caller_id);
            else
                printf("no caller id detect\n");

            if (caller_name[0])
                printf("caller name is %s\n", caller_name);

            channel_dial_local(channel, &channels[0]);    // dial fxs 0
        #else
            char text[] = {IVR_TEXT_ID_PLZ_ENTER_NUMBER, '\0'};

            usleep(500000); // after phone off-hook, wait for a second, and then play IVR
            rtk_FXO_offhook(channel->chid);
        #ifdef CONFIG_RTK_VOIP_IVR
            rtk_IvrStartPlaying(channel->chid, text);
```

```
        #else
            // use dial tone to replace if no IVR
            rtk_SetPlayTone(channel->chid, 0, DSPCODEC_TONE_DIAL, 1,
DSPCODEC_TONEDIRECTION_LOCAL);
        #endif
        #endif
        }
        break;
    case STATE_RING:
        if (CHANNEL_IS_FXS(channel->chid))
        {
            // fxs
            rtk_SetRingFXS(channel->chid, 0);
            rtk_SetPlayTone(channel->remote->chid, 0, DSPCODEC_TONE_RINGING, 0,
DSPCODEC_TONEDIRECTION_LOCAL);
            channel_rtp_start(channel);
            channel->state = STATE_CONNECT;
        }
        else
        {
            // daa
            rtk_FXO_offhook(channel->chid);
            channel_rtp_start(channel);
            channel->state = STATE_CONNECT;
        }

        // ack remote channel
        if (CHANNEL_IS_FXS(channel->remote->chid))
        {
            // fxs
        }
        else
```

```
            {
                // daa
                rtk_FXO_offhook(channel->remote->chid);
            }
            break;
        case STATE_DAA_RING:
            // virtual daa
            rtk_DAA_off_hook(channel->chid);
            channel->state = STATE_DAA_CONNECT;
            break;
        default:
            break;
        }
}

int main()
{
        int i;
        SIGNSTATE val;
        channel_t *channel;

        rtk_Get_VoIP_Feature();

        if ((g_VoIP_Feature & DAA_TYPE_MASK) == NO_DAA)
            fxo_type = FXO_NO_DAA;
        else if ((g_VoIP_Feature & DAA_TYPE_MASK) == VIRTUAL_DAA)
            fxo_type = FXO_VIRTUAL_DAA;
        else
            fxo_type = FXO_DAA;

        // init
        for (i=0; i<VOIP_CH_NUM; i++)
```

```
{
    // init dsp
    rtk_InitDSP(i);

    // init phone
    rtk_Set_Voice_Gain(i, 0, 0);      // 0db
    rtk_Set_echoTail(i, 4);           // 4ms

    // init fxs & fxo
    if (CHANNEL_IS_FXS(i))
    {
        // fxs
        rtk_Set_Flash_Hook_Time(i, 0, 300);              // 0~300 ms
        rtk_Set_SLIC_Ring_Cadence(i, 1500, 1500); // 1500 ms
        rtk_Set_CID_DTMF_MODE(i,
            CID_DTMF | 0x08,                             // DTMF mode, Prior Ring
            CID_DTMF_A | CID_DTMF_C << 2                 // start digit = A, end digit = C
        );
    }
    else
    {
        // fxo
        rtk_Set_CID_Det_Mode(i,
            2,                                           // auto detect mode (NOT NTT)
            CID_DTMF
        );
    }

    // set dtmf mode
    rtk_SetDTMFMODE(i, DTMF_INBAND);

    // flush kernel fifo before app run
```

```
        rtk_Set_flush_fifo(i);

        // init local data
        channel_init(&channels[i], i);
    }

    // main loop
main_loop:

    for (i=0; i<VOIP_CH_NUM; i++)
    {
        channel = &channels[i];

        if (CHANNEL_IS_FXS(i))
            rtk_GetFxsEvent(i, &val);
        else
            rtk_GetFxoEvent(i, &val);

        switch (val)
        {
        case SIGN_KEY1:
        case SIGN_KEY2:
        case SIGN_KEY3:
        case SIGN_KEY4:
        case SIGN_KEY5:
        case SIGN_KEY6:
        case SIGN_KEY7:
        case SIGN_KEY8:
        case SIGN_KEY9:
        case SIGN_KEY0:
        case SIGN_STAR:
        case SIGN_HASH:
```

```
                if (channel->state == STATE_IDLE)
                {
                    channel_handle_input(channel, val);
                }
                break;
            case SIGN_ONHOOK:
                channel_handle_onhook(channel);
                rtk_Onhook_Action(i); // rtk_Onhook_Action have to be the last step in
SIGN_ONHOOK
                break;
            case SIGN_OFFHOOK:
                rtk_Offhook_Action(i); // rtk_Offhook_Action have to be the first step in
SIGN_OFFHOOK
                channel_handle_offhook(channel);
                break;
        case SIGN_RING_ON:          // virtual daa ring on
                if (channel->state == STATE_IDLE)
                {
                    channel->state = STATE_DAA_RING;
                    rtk_SetRingFXS(channel->chid, 1);
                }
                break;
            case SIGN_RING_OFF:    // virtual daa ring off
                if (channel->state == STATE_DAA_RING)
                {
                    rtk_SetRingFXS(channel->chid, 0);
                    channel->state = STATE_IDLE;
                }
                break;
            case SIGN_FLASHHOOK:
                break;
            case SIGN_OFFHOOK_2: // off-hook but no event
```

```
                break;
            case SIGN_NONE:
                break;
            default:
                printf("unknown(%d)\n", val);
                break;
            }

            usleep(100000 / VOIP_CH_NUM); // 100ms
        }

        goto main_loop;

        return 0;
}
```

11.6. phone

```
#include "voip_manager.h"

int main()
{
        int i;
        SIGNSTATE val;

        for (i=0; i<SLIC_CH_NUM; i++)
        {
                rtk_Set_Voice_Gain(i, 0, 0);
                rtk_Set_Flash_Hook_Time(i, 0, 300);
                rtk_Set_flush_fifo(i);                  // flush kernel fifo before app run
        }

main_loop:
```

```
    for (i=0; i<SLIC_CH_NUM; i++)
    {
        rtk_GetFxsEvent(i, &val);
        switch (val)
        {
        case SIGN_KEY1:
        case SIGN_KEY2:
        case SIGN_KEY3:
        case SIGN_KEY4:
        case SIGN_KEY5:
        case SIGN_KEY6:
        case SIGN_KEY7:
        case SIGN_KEY8:
        case SIGN_KEY9:
        case SIGN_KEY0:
        case SIGN_STAR:
        case SIGN_HASH:
            rtk_SetPlayTone(i, 0, DSPCODEC_TONE_HOLD, 0,
DSPCODEC_TONEDIRECTION_LOCAL);
            break;
        case SIGN_ONHOOK:
            rtk_SetPlayTone(i, 0, DSPCODEC_TONE_HOLD, 0,
DSPCODEC_TONEDIRECTION_LOCAL);
            rtk_Onhook_Action(i);
            break;
        case SIGN_OFFHOOK:
            rtk_Offhook_Action(i);
            rtk_SetPlayTone(i, 0, DSPCODEC_TONE_DIAL, 1,
DSPCODEC_TONEDIRECTION_LOCAL);
            break;
        case SIGN_FLASHHOOK:
```

```
                rtk_SetPlayTone(i, 0, DSPCODEC_TONE_HOLD, 1,
DSPCODEC_TONEDIRECTION_LOCAL);
                break;
            case SIGN_NONE:
                break;
            case SIGN_OFFHOOK_2:
                break;
            default:
                printf("unknown(%d)\n", val);
                break;
            }

            usleep(100000 / SLIC_CH_NUM); // 100 ms
        }

        goto main_loop;

        return 0;
}
```

11.7. ring

```
#include "voip_manager.h"

int main()
{
        int i;

        for (i=0; i<SLIC_CH_NUM; i++)
        {
            if (i == 0)
                rtk_Set_SLIC_Ring_Cadence(i, 600, 600); // test value
            else
```

```
                rtk_Set_SLIC_Ring_Cadence(i, 1500, 1000); // default value

            rtk_SetRingFXS(i, 1);
            sleep(3);
            rtk_SetRingFXS(i, 0);
        }

        return 0;
}
```

11.8. RTP

```
#include "voip_manager.h"

int main(int argc, char *argv[])
{
        SIGNSTATE val;
        rtp_config_t rtp_config;
        payloadtype_config_t codec_config;
        int ActiveSession;
        int bStartRTP;
        int dtmf_val[SIGN_HASH + 1] = {0,1,2,3,4,5,6,7,8,9,0,10,11};
        int chid = 0;

        if (argc != 4)
        {
                printf("usage: %s chid src_ip dest_ip\n", argv[0]);
                return 0;
        }

        // init channel
        chid = atoi(argv[1]);
        // init dsp
```

```
rtk_InitDSP(chid);
rtk_SetDTMFMODE(chid, DTMF_RFC2833);// set dtmf mode
// init flag
bStartRTP = 0;
ActiveSession = 0;
while (1)
{
    rtk_GetFxsEvent(chid, &val);
    switch (val)
    {
    case SIGN_KEY1:
    case SIGN_KEY2:
    case SIGN_KEY3:
    case SIGN_KEY4:
    case SIGN_KEY5:
    case SIGN_KEY6:
    case SIGN_KEY7:
    case SIGN_KEY8:
    case SIGN_KEY9:
    case SIGN_KEY0:
    case SIGN_STAR:
    case SIGN_HASH:
        if (bStartRTP && ActiveSession == 0)
            rtk_SetRTPRFC2833(chid, 0, dtmf_val[val], 100);
        break;
    case SIGN_OFFHOOK:
        // call rtk_Offhook_Action at first
        rtk_Offhook_Action(chid);
        ActiveSession = 0;
        rtk_SetTranSessionID(chid, ActiveSession);

        // set rtp session
```

```
memset(&rtp_config, 0, sizeof(rtp_config));
rtp_config.chid = chid;        // use channel
rtp_config.sid = ActiveSession;
rtp_config.isTcp = 0;          // use udp
rtp_config.extIp = inet_addr(argv[2]);
rtp_config.remIp = inet_addr(argv[3]);
rtp_config.extPort = htons(9000 + chid);
rtp_config.remPort = htons(9000 + (chid ? 0 : 1));
rtp_config.tos = 0;
rtp_config.rfc2833_payload_type_local = 96;
rtp_config.rfc2833_payload_type_remote = 96;
rtk_SetRtpConfig(&rtp_config);

// set rtp payload, and other session parameters.
codec_config.chid = chid;
codec_config.sid = ActiveSession;
codec_config.local_pt = 18;             // G729 use static pt 18
codec_config.remote_pt = 18;   // G729 use static pt 18
codec_config.uPktFormat = rtpPayloadG729;
codec_config.nG723Type = 0;
codec_config.nFramePerPacket = 1;
codec_config.bVAD = 0;
codec_config.nJitterDelay = 4;
codec_config.nMaxDelay = 13;
codec_config.nJitterFactor = 7;
codec_config.nG726Packing = 2;//pack right
rtk_SetRtpPayloadType(&codec_config);

// start rtp (channel number = 0, session number = 0)
rtk_SetRtpSessionState(chid, ActiveSession, rtp_session_sendrecv);
bStartRTP = 1;
printf("%s:%d -> %s:%d\n",
```

```
                argv[2], 9000 + chid, argv[3], 9000 + (chid ? 0 : 1));
            break;

        case SIGN_ONHOOK:
            // close rtp
            rtk_SetRtpSessionState(chid, 0, rtp_session_inactive);
            rtk_SetRtpSessionState(chid, 1, rtp_session_inactive);
            bStartRTP = 0;
            // call rtk_Offhook_Action at last
            rtk_Onhook_Action(chid);
            break;

        case SIGN_FLASHHOOK:
            ActiveSession = !ActiveSession;
            rtk_SetTranSessionID(chid, ActiveSession);
            if (ActiveSession == 1)
            {
                // hold session
                rtk_Hold_Rtp(chid, 0, 1);
                // play dial tone
                rtk_SetPlayTone(chid, 1, DSPCODEC_TONE_DIAL, 1,
DSPCODEC_TONEDIRECTION_LOCAL);
            }
            else
            {
                // resume session
                rtk_Hold_Rtp(chid, 0, 0);
                // stop dial tone
                rtk_SetPlayTone(chid, 1, DSPCODEC_TONE_HOLD, 0,
DSPCODEC_TONEDIRECTION_LOCAL);
            }
            break;
```

```
            default:
                  break;
            }

            usleep(100000); // 100ms
      }

      return 0;
}
```

11.9. vmwi

```
#include "voip_manager.h"

int main()
{
      char vmwi_on = 1, vmwi_off = 0;

      // VMWI test via FSK Type I
      rtk_Gen_FSK_VMWI(0, &vmwi_on, 0);
      sleep(2);
      rtk_Gen_FSK_VMWI(0, &vmwi_off, 0);

      return 0;
}
```

11.10. TLS REGISTER

```
#include <stdio.h>
#include <stdlib.h>
#include <sys/types.h>
#include <sys/socket.h>
```

```
#include <arpa/inet.h>
#include <netdb.h>
#include <stdio.h>
#include <string.h>
#include <signal.h>
#include <unistd.h>

#include <openssl/ssl.h>
#include <openssl/err.h>

#define CA_LIST "cacert.pem"

BIO *bio_err=0;

static void sigpipe_handle(int x){
      printf("+++++%s+++++\n",__FUNCTION__);
}

/* Print SSL errors and exit*/
int berr_exit(char *string)
{
     BIO_printf(bio_err,"%s\n",string);
     ERR_print_errors(bio_err);
     exit(0);
}

SSL_CTX *initialize_ctx(){
      SSL_METHOD *meth;
      SSL_CTX *ctx;

      if(!bio_err){
      /* Global system initialization*/
```

格式化：義大利文（義大利）

```
        SSL_library_init();
        SSL_load_error_strings();

    /* An error write context */
        bio_err=BIO_new_fp(stderr,BIO_NOCLOSE);
    }

    /* Set up a SIGPIPE handler */
    signal(SIGPIPE,sigpipe_handle);

    /* Create our context*/
    meth=TLSv1_method();
    ctx=SSL_CTX_new(meth);

    /* Load the CAs we trust*/
    if(!(SSL_CTX_load_verify_locations(ctx,CA_LIST,0)))
        berr_exit("Can't read CA list");
#if (OPENSSL_VERSION_NUMBER < 0x00905100L)
        SSL_CTX_set_verify_depth(ctx,1);
#endif

    return ctx;
}
char* genRegMsg(){
    char *sipMsg;
    char *sipurl="REGISTER sip:192.168.1.218:5061 SIP/2.0\r\n\0";
    char *via="Via: SIP/2.0/TLS
192.168.1.200:5061;rport;branch=z9hG4bK-d87543-d641201b415f0331-1--d87543\r\n\0";
    char *forward="Max-Forwards: 70\r\n\0";
    char *contact="Contact:
<sip:104@192.168.1.200:5060;rinstance=ba81852eac1b47c3;transport=TLS>\r\n\0";
    char *to="To: \"4487\"<sip:104@192.168.1.218:5061>\r\n\0";
```

```
	char *from="From: \"4487\"<sip:104@192.168.1.218:5061>;tag=7a05a24e\r\n\0";
	char *callid="Call-ID: 633697324@192.168.1.200\r\n\0";
	char *cseq="CSeq: 13 REGISTER\r\n\0";
	char *expire="Expires: 3600\r\n\0";
	char *allow="Allow: INVITE, ACK, CANCEL, OPTIONS, BYE, REFER, NOTIFY, MESSAGE,
SUBSCRIBE, INFO\r\n\0";
	char *userAgent="User-Agent: eyeBeam\r\n\0";
	char *content="Content-Length: 0\r\n\0";

	sipMsg = (char*)malloc(sizeof(char)*4000);
	sprintf(sipMsg,"%s%s%s%s%s%s%s%s%s%s%s\r\n\0",
		sipurl,via,forward,contact,to,from,callid,cseq,expire,allow,userAgent,content);
	return sipMsg;
}

int tlsSipSendMsg(char *sipMsg, SSL *ssl, int len){
	int ret;

	ret=SSL_write(ssl, sipMsg, len);
	if(ret < 0)
		printf("+++++SSL_write error!+++++\n");

	return ret;
}
int main(int argc, char *argv[])
{
	int tcpSocket;
	char *sipRegMsgStr;
	SSL_CTX *ctx;
	SSL *ssl;
	BIO *sbio;
	struct sockaddr_in addr;
```

```
    if(argc != 2){
        printf("usage: reg <IP>\n");
        exit(1);
    }

    /* Build our SSL context*/
    ctx=initialize_ctx();

  /*build socket*/
    tcpSocket = socket(PF_INET, SOCK_STREAM, IPPROTO_TCP);
    if(tcpSocket == -1)
        perror("open socket error");
    addr.sin_family = PF_INET;
    addr.sin_addr.s_addr=inet_addr(argv[1]);
    addr.sin_port = htons(5061);
    connect(tcpSocket, (struct sockaddr*)&addr, sizeof(addr));

    /* Connect the SSL socket */
    ssl=SSL_new(ctx);
    sbio=BIO_new_socket(tcpSocket,BIO_NOCLOSE);
    SSL_set_bio(ssl,sbio,sbio);

    if(SSL_connect(ssl)<=0)
      berr_exit("SSL connect error");

    sipRegMsgStr = genRegMsg();
#if 1
    printf("---send sip message---\n%s",sipRegMsgStr);
    tlsSipSendMsg(sipRegMsgStr, ssl, strlen(sipRegMsgStr));
    while(1){
        int i;
```

```
            char *buf=NULL;
            int bufSize=0;

            buf = (char*)malloc(4000*sizeof(char));
            if((bufSize=SSL_read(ssl, buf, 4000)) == 0){
                  free(buf);
                  continue;
            }else{
                  printf("---recv sip message---\n%s",buf);
                  free(buf);
            }
            break;
      }
#endif
      free(sipRegMsgStr);
      SSL_CTX_free(ctx);
      close(tcpSocket);
      return 0;
}
```

11.11. Pulse Dial Generation/Detection

```
#include "voip_manager.h"

int pulse_dial_main(int argc, char *argv[])
{
          int chid, i, enable;
          unsigned int pause_time, max_break_ths, min_break_ths;

          if (argc < 2)
                    goto PULSE_DIAL_USAGE;
```

```
        if (argv[1][0] == 'g') // Pulse Dial Generation
        {
                if (argc == 4)
                {
                        if (argv[1][1] == 'm')
                        {
                                rtk_Set_Dail_Mode(atoi(argv[2]), atoi(argv[3]));

                                if (atoi(argv[2]) == 1)
                                        printf("Enable pulse dial gen for ch=%d\n", chid);
                                else if (atoi(argv[2]) == 0)
                                        printf("Disable pulse dial gen for ch=%d\n", chid);
                        }
                        else if (argv[1][1] == 'g')
                        {
                                int sum = 0;
                                int len = 0;

                                rtk_debug(0); // disable kernel debug message to avoid effect
pulse dial

                                chid = atoi(argv[2]);
                                len = strlen(argv[3]);

                                rtk_FXO_offhook(chid);
                                sleep(2);
                                for (i=0; i < len; i++)
                                {
                                        //printf("gen %d\n", argv[2][i] - '0');
                                        rtk_Gen_Pulse_Dial(chid, argv[3][i] - '0');
                                        sum = sum + (argv[3][i] - '0');
                                }
```

```
                                usleep(100000*sum + 1000*800*(len-1));   // sleep enough time
to make sure pulse dial gen finish, assume interdigit_duration = 800 ms
                                sleep(2);
                                rtk_FXO_onhook(chid);

                                rtk_debug(3); //enable kernel debug message
                        }
                }
                else if (argc == 5)
                {

                        rtk_PulseDial_Gen_Cfg(atoi(argv[2]), atoi(argv[3]), atoi(argv[4]));
                        printf("Config pulse dial gen to %d pps, make: %d ms, interdigit
duration:%d ms\n", atoi(argv[2]), atoi(argv[3]), atoi(argv[4]));
                }
                else
                {
                        printf("usage error!\n\n");
                        goto PULSE_DIAL_USAGE;
                }
        }
        else if (argv[1][0] == 'd') // Pulse Dial Detection
        {
                if (argc == 7)
                {
                        chid = atoi(argv[2]);
                        enable = atoi(argv[3]);
                        pause_time = atoi(argv[4]);
                        min_break_ths = atoi(argv[5]);
                        max_break_ths = atoi(argv[6]);

                        rtk_Set_Pulse_Digit_Det(chid, enable, pause_time, min_break_ths,
```

```
max_break_ths);
                        if (enable == 1)
                        {
                                printf("Enable pulse dial det for ch= %d, ", chid);
                                printf("pause_time= %d ms, min_break_ths = %d ms,
max_break_ths = %d ms\n", pause_time, min_break_ths, max_break_ths);
                        }
                        else if (enable == 0)
                                printf("Disable pulse dial det for ch= %d\n", chid);
                }
                else
                {
                        printf("usage error!\n\n");
                        goto PULSE_DIAL_USAGE;
                }
        }
        else
        {
PULSE_DIAL_USAGE:
                printf("pulse_dial usage:\n");
                printf(" - Enable/Disable pulse dial generation: pulse_dial gm chid flag\n");
                printf("     + Example to enable pulse dial gen: pulse_dial g 2 1\n");
                printf(" - Enable/Disable pulse dial detection: pulse_dial d chid flag pause_time
min_break_ths max_break_ths\n");
                printf("     + Example to enable pulse dial det: pulse_dial d 0 1 450 20 70\n");
                printf(" - Config pulse dial generation:    pulse_dial g pps make_time
interdigit_duration\n");
                printf("     + Example1: pulse_dial g 10 40 700\n");
                printf("     + Example2: pulse_dial g 20 20 750\n");
                printf(" - Generate pulse dial :    pulse_dial gg chid digit(0~9)\n");
                printf("     + Example1 to gen digit 3939889: pulse_dial gg 2 3939889\n");
```

```
                return -1;
        }

        return 0;
}
```

格式化：字型：(中文) 新細明體

12. Appendix A – Enumeration

12.1. Enumeration for phone state

```
typedef enum
{
    SIGN_NONE = 0 ,
    SIGN_KEY1 = 1 ,
    SIGN_KEY2 = 2 ,
    SIGN_KEY3 = 3 ,
    SIGN_KEY4 = 4,
    SIGN_KEY5 = 5 ,
    SIGN_KEY6 = 6 ,
    SIGN_KEY7 = 7 ,
    SIGN_KEY8 = 8 ,
    SIGN_KEY9 = 9 ,
    SIGN_KEY0 = 10 ,
    SIGN_STAR = 11,
    SIGN_HASH = 12 ,
    SIGN_ONHOOK = 13 ,
    SIGN_OFFHOOK = 18,
    SIGN_FLASHHOOK = 19,
    SIGN_AUTODIAL = 20,
    SIGN_OFFHOOK_2 = 21,
```

```
    SIGN_RING_ON = 22,
    SIGN_RING_OFF = 23
} SIGNSTATE ;
```

12.2. Enumeration for FXO Event

```
typedef enum
{
    FXOEVENT_NONE = 0 ,
    FXOEVENT_KEY1 = 1 ,
    FXOEVENT_KEY2 = 2 ,
    FXOEVENT_KEY3 = 3 ,
    FXOEVENT_KEY4 = 4,
    FXOEVENT_KEY5 = 5 ,
    FXOEVENT_KEY6 = 6 ,
    FXOEVENT_KEY7 = 7 ,
    FXOEVENT_KEY8 = 8 ,
    FXOEVENT_KEY9 = 9 ,
    FXOEVENT_KEY0 = 10 ,
    FXOEVENT_STAR = 11,
    FXOEVENT_HASH = 12 ,
    FXOEVENT_RING_START = 13 ,
    FXOEVENT_RING_STOP = 14,
    //FXOEVENT_PSTN_OFFHOOK = 15,    // not support yet
    //FXOEVENT_PSTN_ONHOOK = 16,    // not support yet
    FXOEVENT_BUSY_TONE = 17,
    FXOEVENT_DIST_TONE = 18,
    FXOEVENT_POLARITY_REVERSAL = 19,
    FXOEVENT_BAT_DROP_OUT = 20,
    FXOEVENT_MAX = 21
} FXOEVENT;
```

12.3. Enumeration for supported RTP payload

```
typedef enum
{
    rtpPayloadUndefined = -1,
    rtpPayloadPCMU = 0,
    rtpPayload1016 = 1,
    rtpPayloadG726_32 = 2,
    rtpPayloadGSM = 3,
    rtpPayloadG723 = 4,
    rtpPayloadDVI4_8KHz = 5,
    rtpPayloadDVI4_16KHz = 6,
    rtpPayloadLPC = 7,
    rtpPayloadPCMA = 8,
    rtpPayloadG722 = 9,
    rtpPayloadL16_stereo = 10,
    rtpPayloadL16_mono = 11,
    rtpPayloadQCELP = 12,
    rtpPayloadMPA = 14,
    rtpPayloadG728 = 15,
    rtpPayloadDVI4_11KHz = 16,
    rtpPayloadDVI4_22KHz = 17,
    rtpPayloadG729 = 18,
    rtpPayloadCN = 19,
    rtpPayloadG726_40    = 21,
    rtpPayloadG726_24    = 22,
    rtpPayloadG726_16    = 23,
    rtpPayloadCelB = 25,
    rtpPayloadJPEG = 26,
    rtpPayloadNV = 28,
    rtpPayloadH261 = 31,
    rtpPayloadMPV = 32,
```

```
    rtpPayloadMP2T = 33,
    rtpPayloadH263 = 34,
    // fake codec
    rtpPayloadSilence = 35,
    // dynamic payload type in rtk_voip
    rtpPayload_iLBC = 97,
    rtpPayload_iLBC_20ms = 98,
    rtpPayloadDTMF_RFC2833 = 101,
    rtpPayloadT38_Virtual = 110,
    rtpPayloadCiscoRtp = 121,
    rtpPayloadL16_8k_mono = 122,
    rtpPayload_AMR_NB = 123,
    rtpPayload_SPEEX_NB_RATE8 = 111,
    rtpPayload_SPEEX_NB_RATE2P15 = 112,
    rtpPayload_SPEEX_NB_RATE5P95 = 113,
    rtpPayload_SPEEX_NB_RATE11 = 114,
    rtpPayload_SPEEX_NB_RATE15 = 115,
    rtpPayload_SPEEX_NB_RATE18P2 = 116,
    rtpPayload_SPEEX_NB_RATE24P6 = 117,
    rtpPayload_SPEEX_NB_RATE3P95 = 118,
} RtpPayloadType;
```

12.4. Enumeration for RTP session state

```
typedef enum
{
    rtp_session_undefined = -1,
    rtp_session_inactive = 0,
    rtp_session_sendonly = 1,
    rtp_session_recvonly = 2,
    rtp_session_sendrecv = 3
```

```
} RtpSessionState;
```

12.5. Enumeration for country

```
typedef enum
{
    DSPCODEC_COUNTRY_USA,
    DSPCODEC_COUNTRY_UK,
    DSPCODEC_COUNTRY_AUSTRALIA,
    DSPCODEC_COUNTRY_HK,
    DSPCODEC_COUNTRY_JP,
    DSPCODEC_COUNTRY_SE,
    DSPCODEC_COUNTRY_GR,
    DSPCODEC_COUNTRY_FR,
    DSPCODEC_COUNTRY_TR,
    DSPCODEC_COUNTRY_BE,
    DSPCODEC_COUNTRY_FL,
    DSPCODEC_COUNTRY_IT,
    DSPCODEC_COUNTRY_CN,
    DSPCODEC_COUNTRY_CUSTOME
} DSPCODEC_COUNTRY;
```

12.6. Enumeration for DSP tone

```
typedef enum
{
    DSPCODEC_TONE_NONE = -1,
    DSPCODEC_TONE_0 = 0,
    DSPCODEC_TONE_1,
    DSPCODEC_TONE_2,
    DSPCODEC_TONE_3,
    DSPCODEC_TONE_4,
    DSPCODEC_TONE_5,
```

```
    DSPCODEC_TONE_6,
    DSPCODEC_TONE_7,
    DSPCODEC_TONE_8,
    DSPCODEC_TONE_9,
    DSPCODEC_TONE_STARSIGN,                 // *
    DSPCODEC_TONE_HASHSIGN,                 // #
    DSPCODEC_TONE_DIAL,                 // code = 66 in RFC2833
    DSPCODEC_TONE_STUTTERDIAL,              //
    DSPCODEC_TONE_MESSAGE_WAITING,              //
    DSPCODEC_TONE_CONFIRMATION,             //
    DSPCODEC_TONE_RINGING, /* ring back tone */ // code = 70 in RFC2833
    DSPCODEC_TONE_BUSY,                 // code = 72 in RFC2833
    DSPCODEC_TONE_CONGESTION,               //
    DSPCODEC_TONE_ROH,                  //
    DSPCODEC_TONE_DOUBLE_RING,              //
    DSPCODEC_TONE_SIT_NOCIRCUIT,                //
    DSPCODEC_TONE_SIT_INTERCEPT,                //
    DSPCODEC_TONE_SIT_VACANT,               //
    DSPCODEC_TONE_SIT_REORDER,              //
    DSPCODEC_TONE_CALLING_CARD_WITHEVENT,           //
    DSPCODEC_TONE_CALLING_CARD,             //
    DSPCODEC_TONE_CALL_WAITING,             // code = 79 in RFC2833
    DSPCODEC_TONE_CALL_WAITING_2,               //
    DSPCODEC_TONE_CALL_WAITING_3,               //
    DSPCODEC_TONE_CALL_WAITING_4,               //
    DSPCODEC_TONE_INGRESS_RINGBACK,             //

    DSPCODEC_TONE_HOLD,                 // code = 76 in RFC2833
    DSPCODEC_TONE_OFFHOOKWARNING,               // code = 88 in RFC2833
    DSPCODEC_TONE_RING,                 // code = 89 in RFC2833
#ifdef SW_DTMF_CID
    DSPCODEC_TONE_A,                    // DTMF digit A (19)
```

格式化 :英文 (美國)

```
    DSPCODEC_TONE_B,                    // DTMF digit B (20)
    DSPCODEC_TONE_C,                    // DTMF digit C (21)
    DSPCODEC_TONE_D,                    // DTMF digit D (22)
#endif
    DSPCODEC_TONE_FSK_SAS,                  // alert signal (23)
    DSPCODEC_TONE_FSK_ALERT,                 // alert signal (23)
    DSPCODEC_TONE_NTT_IIT_TONE,              // for ntt type 2 caller id
    // continous DTMF tone play for RFC2833
    DSPCODEC_TONE_0_CONT,
    DSPCODEC_TONE_1_CONT,
    DSPCODEC_TONE_2_CONT,
    DSPCODEC_TONE_3_CONT,
    DSPCODEC_TONE_4_CONT,
    DSPCODEC_TONE_5_CONT,
    DSPCODEC_TONE_6_CONT,
    DSPCODEC_TONE_7_CONT,
    DSPCODEC_TONE_8_CONT,
    DSPCODEC_TONE_9_CONT,
    DSPCODEC_TONE_STARSIGN_CONT,
    DSPCODEC_TONE_HASHSIGN_CONT,
    DSPCODEC_TONE_A_CONT,
    DSPCODEC_TONE_B_CONT,
    DSPCODEC_TONE_C_CONT,
    DSPCODEC_TONE_D_CONT,

    DSPCODEC_TONE_KEY                   // the others key tone

} DSPCODEC_TONE;
```

12.7. Enumeration for DSP play tone direction

```
typedef enum
{
```

```
    DSPCODEC_TONEDIRECTION_LOCAL,          // local
    DSPCODEC_TONEDIRECTION_REMOTE,         // remote
    DSPCODEC_TONEDIRECTION_BOTH            // local and remote
} DSPCODEC_TONEDIRECTION;
```

12.8. Enumeration for IVR play

```
typedef enum {
    IVR_PLAY_TYPE_TEXT,        /* play a string, such as '192.168.0.0' */
    IVR_PLAY_TYPE_G723_63,     /* play a G723 6.3k data (24 bytes per 30 ms) */
    IVR_PLAY_TYPE_G729,        /* play a G729 data (10 bytes per 10ms) */
    IVR_PLAY_TYPE_G711A,       /* play a G711 a-law (80 bytes per 10ms) */
} IvrPlayType_t;
```

12.9. Enumeration for IVR play direction (play TEXT only)

```
typedef enum {
    IVR_DIR_LOCAL,
    IVR_DIR_REMOTE,
} IvrPlayDir_t;
```

12.10. Enumeration for FAX State

```
typedef enum
{
    FAX_IDLE = 0,
    FAX_DETECT ,
    MODEM_DETECT,
    FAX_RUN,
    MODEM_RUN
} FAXSTATE;
```

12.11. Enumeration for DTMF type

```
enum __DTMF_TYPE__
{
    DTMF_RFC2833 = 0,
    DTMF_SIPINFO,
    DTMF_INBAND,
    DTMF_MAX
};
```

12.12. Enumeration for the Country

```
typedef enum
{
    COUNTRY_USA = 0,
    COUNTRY_UK,
    COUNTRY_AUSTRALIA,
    COUNTRY_HK,
    COUNTRY_JAPAN,
    COUNTRY_SWEDEN,
    COUNTRY_GERMANY,
    COUNTRY_FRANCE,
#if 0
    COUNTRY_TR57,
#else
    COUNTRY_TW,
#endif
    COUNTRY_BELGIUM,
    COUNTRY_FINLAND,
    COUNTRY_ITALY,
    COUNTRY_CHINA,
    COUNTRY_CUSTOMER,
    COUNTRY_MAX
```

```
}_COUNTRY_;
```

12.13. Enumeration for FSK Caller ID Area

```
typedef enum
{
    FSK_Bellcore = 0,
    FSK_ETSI,
    FSK_BT,
    FSK_NTT
}TfskArea;
```

12.14. Enumeration for FSK Caller ID Parameter Type

```
typedef enum
{
    FSK_PARAM_NULL = 0,                 // No Data
    FSK_PARAM_DATEnTIME = 0x01,         // Date and Time
    FSK_PARAM_CLI = 0x02,               // Calling Line Identify (CLI)
    FSK_PARAM_CLI_ABS = 0x04,           // Reason for absence of CLI
    FSK_PARAM_CLI_NAME = 0x07,          // Calling Line Identify (CLI) Name
    FSK_PARAM_CLI_NAME_ABS = 0x08,      // Reason for absence of (CLI) Name
    FSK_PARAM_MW = 0x0b,                // Message Waiting

}TfskParaType;
```

12.15. Enumeration for FSK Caller ID Gain

```
typedef enum
{
    FSK_CLID_GAIN_8DB_UP = 0,
    FSK_CLID_GAIN_7DB_UP,
```

```
    FSK_CLID_GAIN_6DB_UP,
    FSK_CLID_GAIN_5DB_UP,
    FSK_CLID_GAIN_4DB_UP,
    FSK_CLID_GAIN_3DB_UP,
    FSK_CLID_GAIN_2DB_UP,
    FSK_CLID_GAIN_1DB_UP,
    FSK_CLID_GAIN_0DB,
    FSK_CLID_GAIN_1DB_DOWN,
    FSK_CLID_GAIN_2DB_DOWN,
    FSK_CLID_GAIN_3DB_DOWN,
    FSK_CLID_GAIN_4DB_DOWN,
    FSK_CLID_GAIN_5DB_DOWN,
    FSK_CLID_GAIN_6DB_DOWN,
    FSK_CLID_GAIN_7DB_DOWN,
    FSK_CLID_GAIN_8DB_DOWN,
    FSK_CLID_GAIN_9DB_DOWN,
    FSK_CLID_GAIN_10DB_DOWN,
    FSK_CLID_GAIN_11DB_DOWN,
    FSK_CLID_GAIN_12DB_DOWN,
    FSK_CLID_GAIN_13DB_DOWN,
    FSK_CLID_GAIN_14DB_DOWN,
    FSK_CLID_GAIN_15DB_DOWN,
    FSK_CLID_GAIN_16DB_DOWN,
    FSK_CLID_GAIN_MAX

}TfskClidGain;
```

13. Appendix B – Data Structure

13.1. Data Structure for RTP payload type

- Our jitter buffer is an adaptive one which adjusts buffering delay automatically. One can use API to set minimum and maximum delay to make delay within specified range. Though buffering delay always falls into this range, it makes delay as smaller as possible. Because it is a trade off problem between quality and delay, optimization factor is introduced. This factor is designed for preference to quality or delay. If someone prefers better quality, delay will be larger. In contrast, smaller delay causes worse quality. We recommend default values of minimum and maximum delay are 40 and 130 ms respectively, and set optimization factor to be 7.
- nFramePerPacket means the voiced frame number per RTP packet. For G.711u/A, G.729a/b, G.726, the unit is 10ms. If nFramePerPacket is set to 3, it means RTP packet includes 30ms voice. For G.723, iLBC-30ms, the unit is 30ms. If nFramePerPacket is set to 2, it means RTP packet includes 60ms voice.
- nG726Packing means the G726 RTP packet packing order. 0- non-pack, 1- pack left, 2- pack right. See RFC3551 for detail.
- uPktFormat_vbd, local_pt_vbd and remote_pt_vbd with suffix '_vbd' are used to V.152 application. If SIP negotiation doesn't support V.152, they have to be filled with rtpPayloadUndefined. On the other hand, if it supports V.152, uPktFormat_vbd can specify rtpPayloadPCMU or rtpPayloadPCMA. local_pt_vbd and remote_pt_vbd specify payload type between 96 to 127. One should ignore rtk_GetCEDToneDetect() event, because CED event is handled by DSP to start V.152 mechanism.

```
typedef struct
{
    uint32 chid;                    ///< channel id
    uint32 sid;                     ///< session id
    RtpPayloadType uPktFormat;      ///< payload type in rtk_voip
    RtpPayloadType local_pt;        ///< payload type in local
    RtpPayloadType remote_pt;       ///< payload type in remote answer
```

```
    RtpPayloadType uPktFormat_vbd; ///< payload type in rtk_voip
    RtpPayloadType local_pt_vbd;      ///< payload type in local
    RtpPayloadType remote_pt_vbd;    ///< payload type in remote answer
    int32 nG723Type;                  ///< G.723 bit rate, 0: 6.3K, 1: 5.3K
    int32 nFramePerPacket;            ///< voiced frame number per RTP packet
    int32 bVAD;                       ///< VAD on: 1, off: 0
    uint32 nJitterDelay;              ///< jitter buffer min. delay
    uint32 nMaxDelay;                 ///< jitter buffer max. delay
    uint32 nJitterFactor;             ///< optimization factor of jitter buffer
    uint32 nG726Packing;              ///< G.726 packing order
} payloadtype_config_t;
```

13.2. Data Structure for RTP Configuration

```
typedef struct
{
    uint32 chid;                          ///< channel id
    uint32 sid;                           ///< session id
    uint32 isTcp;                         ///< tcp = 1 or udp = 0
    uint32 remIp;                         ///< remote ip
    uint16 remPort;                       ///< remote port
    uint32 extIp;                         ///< local ip
    uint16 extPort;                       ///< local port
    uint8 tos;                            ///< QoS
    uint16 rfc2833_payload_type_local;    ///< rfc2833 payload type
    uint16 rfc2833_payload_type_remote;   ///< remote rfc2833 payload type
#ifdef CONFIG_RTK_VOIP_SRTP
    unsigned char remoteSrtpKey[SRTP_KEY_LEN];
    unsigned char localSrtpKey[SRTP_KEY_LEN];
    int remoteCryptAlg;
    int localCryptAlg;
#endif /*CONFIG_RTK_VOIP_SRTP*/
```

```
#ifdef SUPPORT_RTP_REDUNDANT
    uint16 rtp_redundant_payload_type_local;    ///< local payload type of rtp redundant
    uint16 rtp_redundant_payload_type_remote; ///< remote payload type of rtp redundant
#endif
} rtp_config_t;
```

13.3. Data Structure for RTP Session

```
typedef struct
{
    uint32 ip_src_addr;
    uint32 ip_dst_addr;
    uint16 udp_src_port;
    uint16 udp_dst_port;
    uint8 protocol;
    uint8 ch_id;
    int32 m_id;
    int32 result;
    uint8 tos;
    uint16 rfc2833_payload_type_local;
    uint16 rfc2833_payload_type_remote;
#ifdef CONFIG_RTK_VOIP_SRTP
    unsigned char remoteSrtpKey[SRTP_KEY_LEN];
    unsigned char localSrtpKey[SRTP_KEY_LEN];
    int remoteCryptAlg;
    int localCryptAlg;
#endif /*CONFIG_RTK_VOIP_SRTP*/
#ifdef SUPPORT_RTP_REDUNDANT
    uint16 rtp_redundant_payload_type_local;
    uint16 rtp_redundant_payload_type_remote;
#endif
    int32       ret_val;
```

```
}
TstVoipMgrSession;
```

13.4. Data Structure for Tone Configuration

```
typedef struct ToneCfgParam st_ToneCfgParam;

struct ToneCfgParam
{
    unsigned long toneType;
    unsigned short cycle;

    unsigned short cadNUM;

    unsigned long CadOn0;
    unsigned long CadOn1;
    unsigned long CadOn2;
    unsigned long CadOn3;
    unsigned long CadOff0;
    unsigned long CadOff1;
    unsigned long CadOff2;
    unsigned long CadOff3;

    unsigned long PatternOff;
    unsigned long ToneNUM;

    unsigned long Freq1;
    unsigned long Freq2;
    unsigned long Freq3;
    unsigned long Freq4;

    long Gain1;
```

```
    long Gain2;
    long Gain3;
    long Gain4;

};
```

13.5. Data Structure for 3-way Conference Configuration

```
typedef struct
{
    uint8 ch_id;
    uint8 enable;
    TstVoipMgrRtpCfg rtp_cfg[2];
}
TstVoipMgr3WayCfg;

typedef struct
{
    // RTP session info
    uint8 ch_id;
    int32 m_id;
    uint32 ip_src_addr;
    uint32 ip_dst_addr;
    uint16 udp_src_port;
    uint16 udp_dst_port;
    uint16 rtcp_src_port;
    uint16 rtcp_dst_port;
    uint8 protocol;
    uint8 tos;
    uint16 rfc2833_payload_type_local;
    uint16 rfc2833_payload_type_remote;
    //Payload info
```

```
    RtpPayloadType local_pt; // payload type in local
    RtpPayloadType remote_pt; // payload type in remote answer
    RtpPayloadType uPktFormat; // payload type in rtk_voip
    int32 nG723Type;
    int32 nFramePerPacket;
    int32 bVAD;
    uint32 nJitterDelay;
    uint32 nMaxDelay;
    uint32 nJitterFactor;
    // RTP session state
    uint8       state;
    int32 result;
}
TstVoipMgrRtpCfg;
```

13.6. Data Structure for RTP Statistics

```
typedef struct {
    unsigned char chid;                 /* input */
    unsigned char bResetStatistics;     /* input */
    /* follows are output */
    unsigned long nRxRtpStatsCountByte;
    unsigned long nRxRtpStatsCountPacket;
    unsigned long nRxRtpStatsLostPacket;
    unsigned long nTxRtpStatsCountByte;
    unsigned long nTxRtpStatsCountPacket;
} TstVoipRtpStatistics;
```

13.7. Data Structure for VoIP Configuration

```
typedef struct voipCfgParam_s voipCfgParam_t;
```

```
struct voipCfgParam_s {
    // voip flash check
    unsigned long signature;                // voip flash check
    unsigned short version;                 // update if big change (size/offset not match..)
    unsigned long feature;                  // 32 bit feature
    unsigned long extend_feature;           // 32 extend feature

    unsigned short mib_version;          // update if some mib name change    (ex:
VOIP.PORT[0].NUMBER => VOIP.PORT[0].USER.NUMBER)
                                         // used in import/export flash setting
                                         // if mib version not match, then import will fail

    // RFC flags
    unsigned int    rfc_flags;              ///< RFC flags

    //tone              ///< unit: 10ms
    unsigned char       tone_of_country;
    unsigned char       tone_of_custdial;
    unsigned char       tone_of_custring;
    unsigned char       tone_of_custbusy;
    unsigned char       tone_of_custwaiting;
    unsigned char       tone_of_customize;
    st_ToneCfgParam   cust_tone_para[TONE_CUSTOMER_MAX];     // for customized tone (up to
8 tones)

    //disconnect tone det
    unsigned char  distone_num;
    unsigned char  d1freqnum;
    unsigned short d1Freq1;
    unsigned short d1Freq2;
    unsigned char  d1Accur;
    unsigned short d1Level;
```

```
unsigned short d1ONup;
unsigned short d1ONlow;
unsigned short d1OFFup;
unsigned short d1OFFlow;
unsigned char  d2freqnum;
unsigned short d2Freq1;
unsigned short d2Freq2;
unsigned char  d2Accur;
unsigned short d2Level;
unsigned short d2ONup;
unsigned short d2ONlow;
unsigned short d2OFFup;
unsigned short d2OFFlow;

//ring
unsigned char  ring_cad;
unsigned char  ring_group;
unsigned int    ring_phone_num[RING_GROUP_MAX];
unsigned char  ring_cadence_use[RING_GROUP_MAX];
unsigned char  ring_cadence_sel;
unsigned short ring_cadon[RING_CADENCE_MAX];
unsigned short ring_cadoff[RING_CADENCE_MAX];

// function key
char            funckey_pstn[FUNC_KEY_LENGTH];          // default is *0
char            funckey_transfer[FUNC_KEY_LENGTH];      // default is *1

// other
unsigned short auto_dial;                   // 0, 3~9 sec, default is 5 sec, 0 is disable
unsigned short off_hook_alarm;              // 0, 10~60 sec, default is 30 sec, 0 is disable
unsigned short    cid_auto_det_select;      // caller id auto detection selection
unsigned short caller_id_det_mode;          // caller id detection mode
```

```
unsigned char  one_stage_dial;              // 0: disable
                                            // 1: enable one stage dial
                                            //     - VoIP -> FXO -> PSTN
                                            //     - PSTN number is assigned by "To" filed
unsigned char  two_stage_dial;              // 0: disable
                                            // 1: enable two stage dial
                                            //     - VoIP -> FXO -> IVR prompt -> PSTN
unsigned char  auto_bypass_relay;           // bit0: enable/disable bypass relay
                                            //     - SIP Register failed
                                            //     - TODO: Network failed
                                            // bit1: enable/disable warning tone

// pulse dial
unsigned char  pulse_dial_gen;          // 0: disable, 1: enable pulse dial generation for FXO port
unsigned char  pulse_gen_pps;
unsigned char  pulse_gen_make_time;
unsigned short pulse_gen_interdigit_pause;
unsigned char  pulse_dial_det;          // 0: disable, 1: enable pulse dial detection for FXS port
unsigned short pulse_det_pause;

// auto config
unsigned short auto_cfg_ver;                // load config if version is newer
unsigned char  auto_cfg_mode;               // 0: disable, 1: http
char           auto_cfg_http_addr[DNS_LEN];
unsigned short auto_cfg_http_port;
/***   For auto provision for tftp and ftp ***/
char           auto_cfg_tftp_addr[DNS_LEN];
char               auto_cfg_ftp_addr[DNS_LEN];
char           auto_cfg_ftp_user[20];
char               auto_cfg_ftp_passwd[20];
/*** end ***/
char           auto_cfg_file_path[MAX_AUTO_CONFIG_PATH];
```

```
unsigned short auto_cfg_expire;          // 0..365 days

// fw update setting                     // load config if version is newer
unsigned char  fw_update_mode;           // 0: disable, 1: TFTP 2: FTP 3: HTTP
char        fw_update_tftp_addr[DNS_LEN];
char        fw_update_http_addr[DNS_LEN];
unsigned short fw_update_http_port;

char        fw_update_ftp_addr[DNS_LEN];
char        fw_update_ftp_user[MAX_FW_UPDATE_FTP];
char        fw_update_ftp_passwd[MAX_FW_UPDATE_FTP];
char        fw_update_file_path[MAX_FW_UPDATE_PATH];

unsigned char  fw_update_power_on;
unsigned short fw_update_scheduling_day;
unsigned char  fw_update_scheduling_time;
unsigned char  fw_update_auto;

char        fw_update_file_prefix[MAX_FW_UPDATE_FILE_PREFIX];

unsigned long  fw_update_next_time;
char        fw_update_fw_version[MAX_FW_VERSION];

// VLAN setting of WAN Port
unsigned char  wanVlanEnable;
// VLAN for Voice and protocl packets (SIP, STUN, DHCP, ARP, etc.)
unsigned short wanVlanIdVoice;
unsigned char  wanVlanPriorityVoice;
unsigned char  wanVlanCfiVoice;
// VLAN for Data
unsigned short wanVlanIdData;
unsigned char  wanVlanPriorityData;
```

```
unsigned char  wanVlanCfiData;
// VLAN for Video
unsigned short wanVlanIdVideo;
unsigned char  wanVlanPriorityVideo;
unsigned char  wanVlanCfiVideo;

// enable HW-NAT
unsigned char      hwnat_enable;

// bandwidth Mgr
unsigned short bandwidth_LANPort0_Egress;
unsigned short bandwidth_LANPort1_Egress;
unsigned short bandwidth_LANPort2_Egress;
unsigned short bandwidth_LANPort3_Egress;
unsigned short bandwidth_WANPort_Egress;

unsigned short bandwidth_LANPort0_Ingress;
unsigned short bandwidth_LANPort1_Ingress;
unsigned short bandwidth_LANPort2_Ingress;
unsigned short bandwidth_LANPort3_Ingress;
unsigned short bandwidth_WANPort_Ingress;

// FXO valume
unsigned char  daa_txVolumne;                  ///< 1..10
unsigned char  daa_rxVolumne;                  ///< 1..10

// DSCP
unsigned char  rtpDscp;
unsigned char  sipDscp;

voipCfgPortParam_t ports[VOIP_PORTS];
```

```
    // UI
#ifdef CONFIG_RTK_VOIP_IP_PHONE
    ui_falsh_layout_t ui;
#endif
};
```

13.8. Data Structure for FSK Caller ID

```
#define MAX_CLID_DATA_SIZE      80
#define FSK_MDMF_SIZE           5

typedef struct
{
        TfskParaType type;
        char data[MAX_CLID_DATA_SIZE];
}
TstFskClid_Data;

typedef struct
{
        unsigned char ch_id;
        unsigned char service_type;
        TstFskClid_Data cid_data[FSK_MDMF_SIZE];
        int32       ret_val;
}
TstFskClid;

typedef struct
{
    unsigned char ch_id;
    TfskArea area;                          /* area -> 0:Bellcore 1:ETSI 2:BT 3:NTT */
```

```
    unsigned int CS_cnt;                    /* channel seizure signal bit count */
    unsigned int mark_cnt;                  /* mark signal bit count */
    TfskClidGain mark_gain;                 /* unit in dB */
    TfskClidGain space_gain;                /* unit in dB */
    unsigned int type2_expected_ack_tone;   /* DTMF A, or B, or C, or D*/
    unsigned int delay_after_1st_ring;      /* unit in ms */
    unsigned int delay_before_2nd_ring;     /* unit in ms */
    int32       ret_val;
}
TstVoipFskPara;
```