Number :

Secret :

PSTN API

For Specs 1.7

Edit	**Zhijie, Li**	**Date**	**April,30th,2006**
Authen		**Date**	
Confirm		**Date**	

Research and Development Center

Chengdu, Assist Corp.

Modification log

Version	Date	Content	Author	Branch

Content

1. FXS

1.1. FXS Specs

1.1.1. Telephone connection

Telephone connect with a general, fixed telephone is enabled

1.1.2. Check on-hook/off-hook

*as_state_t as_device_check_state_on_startup(int fd, const char *name)*

hook status returned will be *AS_DEVICE_STATE_ONHOOK* or *AS_DEVICE_STATE_OFFHOOK*

When SLIC is broken and phone connect to PSTN line, the hook-status will be checked by call

phone_hook_status as_hook_check(void)

return as

```
typedef enum
{
    G_PHONE_OFFHOOK = 0,
    G_PHONE_ONHOOK
}phone_hook_status;
```

refer to *test_hook_status.c*

Because only this case is only for the first FXS device, so no parameter is needed.

1.1.3. Polarity Control

Implemented by SLIC chip

1.1.4. Check Loop

OnHook

int as_lib_event_get(int fd)

return as *AS_EVENT_ONHOOK*

refer to *void as_lib_wait_onhook(int fd)* in *assist_pstn_ring.c*

OffHook

int as_lib_event_get(int fd)

return as *AS_EVENT_RINGOFFHOOK*

refer to *void as_lib_wait_offhook(int fd)* in *assist_pstn_ring.c*

1.1.5. 700ms ON-HOOK

int as_lib_event_get(int fd)

return as *AS_EVENT_WINKFLASH*

refer to the test program *test_flash.c*

1.1.6. Phone Support

Receive PB DTMF

*int as_dtmf_is_ready(int fd , const char *devicename)*

call this function, the program is blocked. When it is returned, a DTMF signal is detected in firmware

*unsigned char as_dtmf_get_digit(as_device_t *dev)*

call this function to get DTMF char from firmware

Refer to *test_rx_dtmf_4_fxs.c*

Send PB DTMF

*int as_dsp_play_dtmf(as_dsp_t *dsp, int fd, char *callerId)*

use this function before, initialize DSP by *as_dsp_t *as_dsp_init(law_t law, int ms)*

Refer to *test_dsp_dtmf_gen.c*

Receive DP signal

First, use function of *as_get_dp_signal()* to determine is Pulse Dial happened,

Then, use *unsigned char as_get_dtmf_4_fxs(int fd)* to get the last digit called.

Please refer to *test_rx_dp.c*

Send DP signal

Use the function of *int assist_dsp_fxo_send_pps(int fd,char *digitstring,pps_type pps)* to dial the phone number in DP format.

Parameter *pps_type pps* can be used to defined DP10 or DP20.

Refer to *test_fxo_dp.c*

1.1.7. Audioable Tone

DT(Dial Tone)

int as_tone_play_dial(int fd)

After call this function, a dial tone is play to phone. It should be stop by call

int as_tone_play_stop(int fd)

Refer to *test_tones.c*

SDT(Second Dial Tone)

int as_tone_play_dialrecall(int fd)

After call this function, a dial tone is play to phone. It should be stop by call

int as_tone_play_stop(int fd)

Refer to *test_tones.c*

RBT(Ring Back Tone)

int as_tone_play_ringback(int fd)

After call this function, a dial tone is play to phone. It should be stop by call

int as_tone_play_stop(int fd)

Refer to *test_tones.c*

BT(Busy Tone)

int as_tone_play_busy(int fd)

After call this function, a dial tone is play to phone. It should be stop by call

int as_tone_play_stop(int fd)

Refer to *test_tones.c*

IIT(Incoming Identification Tone)

int as_tone_play_income_id_tone(int fd)

After call this function, a dial tone is play to phone. It should be stop by call

int as_tone_play_stop(int fd)

Refer to *test_tones.c*

CPT(Acceptance Tone)

int as_tone_play_accept_tone(int fd)

After call this function, a dial tone is play to phone. It should be stop by call

int as_tone_play_stop(int fd)

Refer to *test_tones.c*

NFT(Notification Tone)

int as_tone_play_notify_tone(int fd)

After call this function, a dial tone is play to phone. It should be stop by call

int as_tone_play_stop(int fd)

Refer to *test_tones.c*

Note 1: After call these tone play functions, firmware will play it forever. So you must stop it by call *as_tone_play_stop()*

Note 2: Maybe frequency coefficients of some tones are not correct. We can tune it as client requirement.

1.1.8. Create IR(Ring Signal)

fd=open("/dev/astel/x",…)

int as_ring_on_hook(int fd)

ringing is stop by call *int as_ring_stop(int fd, const char *devicename)*

where devicename parameter can be ignored

Refer to *test_fsk_ntt_callerid.c*

1.1.9. Create CAR(Data Receiving terminal active signal)

int as_car_on_fxs(int fd)

After call this function, program will be blocked. Return from this function when a call is come from network.

Refer to *test_car_fxs.c*

1.1.10. MODEM Called ID

*int as_dsp_fsk_ring_ntt(as_dsp_t *dsp, int fd , char *number)*

Before call this function, CAR and other call signal must be process as the requirement of carrier. For NTT, please refer to *test_fsk_ntt_callerid.c*

Now, it only send Caller ID. Other data such as DID, etc. can be added as requirement.

1.1.11. Echo Cancel

Echo is canceled by our firmware. No application control is needed.

1.1.12. T38

FAX is called as a normal telephone, so special process is executed in our firmware.

T.38 is not provided as requirement.

1.2. FXS mechanism

Call Control is not provided as requirement.

2. FXO

2.1. FXO Specs

2.1.1. PSTN Line connect

Refer to DAA hardware

2.1.2. FXO line detect

*int as_device_check_fxo_online(int fd, const char *name)*

this function is used to check whether a line connect to FXO port, so it is only used in FXO device

Refer to *test_fxo_line.c*

2.1.3. Line Type detect

PB/DP10/DP20 can not be detected, PB/DP10/DP20 can be send out on the FXO device controlled by the user program.

2.1.4. Line polarity detect

Is not provided as user space API.

2.1.5. Loop generation

int as_lib_offhook(int fd)

FXO off-hook, then loop is generated.

2.1.6. PB(DTMF transmit)

First off hook the FXO device with *int as_lib_offhook(int fd)*

Then, waiting some time or detect DT(Dial Tone)

Last, send out DTMF data with *int as_dsp_play_dtmf(as_dsp_t *dsp, int fd, char *callerId)*

Refer to *test_fxo_calling.c*

2.1.7. DP20/DP transmit

*int assist_dsp_fxo_send_pps(int fd,char *digitstring,pps_type pps)*

send DP signal by FXO

Refer to *test_fxo_dp.c*

2.1.8. IR detect

int as_lib_event_get(int fd)

return an *AS_EVENT_RINGOFFHOOK* , eg. an IR event had detected

consult by *test_busy_fxo.c*

2.1.9. CAR detect

int as_lib_event_get(int fd)

return as *AS_EVENT_CAR* , a CAR event had detected

Refer to *test_car_fxo.c*

2.1.10. Audioable tone detect

DT(Dial Tone):

*int as_dsp_progress_detect(as_dsp_t *dsp, unsigned char *buff,int length)*

return as *AS_PROGRESS_DIALTONE*

Refer to *test_progress_fxo.c*

SDT(Second Dial Tone)

*int as_dsp_progress_detect(as_dsp_t *dsp, unsigned char *buff,int length)*

return as *AS_PROGRESS_2DIALTONE*

Refer to *test_progress_fxo.c*

BT(Busy Tone)

*int as_dsp_progress_detect(as_dsp_t *dsp, unsigned char *buff,int length)*

return as *AS_PROGRESS_BUSYTONE*

Refer to *test_progress_fxo.c*

RBT(Ring Back Tone)

*int as_dsp_progress_detect(as_dsp_t *dsp, unsigned char *buff,int length)*

return as *AS_PROGRESS_RINGBACK*

Refer to *test_progress_fxo.c*

2.1.11. Modem Caller ID detect

*int as_dsp_fsk_ntt_decode(as_dsp_t *dsp, int fd, char *number)*

run this function before , initializtion DSP by *as_dsp_t *as_dsp_init(law_t law, int ms)*

Refer to *test_ntt_detect.c*

2.1.12. JATE support

Provided by DAA chip

2.2. FXO mechanism

2.2.1. Special phone number

```
int as_delay_chan_ctl(delay_switch_chan_tpye status)
typedef enum
{
    DELAY_SWITCH_2_FXS = 0,
    DELAY_SWITCH_2_LINE
}delay_switch_chan_tpye;
```

Refer to *test_delay_chan_ctl.c*

switch the telephone back to FXS port Or switch the telephone to line.

2.2.2. Caller ID detect

```
int as_fsk_decode_ntt_clid_from_fd( fsk_t *dsp, int fd, char *number )
```

FSK modem Caller ID is detected from a device file.

This function can be modified and enhanced to parse Caller ID from buffer.

2.2.3 Connect by straight

```
int as_delay_chan_ctl(delay_switch_chan_tpye status)
```

3. LED

- power LED
- works status LED
- line LED

- voip LED
- pppoe1 LED
- pppoe2 LED
- slot 1 LED
- slot 2 LED

power LED ,slot 1 LED, slot 2 LED control by hardware . other LED control by software :

```
int as_led_line(led_LINE_type status)

int as_led_voip(led_LINE_type status)

int as_led_pppoe1(led_LINE_type status)

int as_led_pppoe2(led_LINE_type status)

typedef enum

{

    LED_LINE_ON = 0,

    LED_LINE_OFF

}led_LINE_type;

int as_led_status(led_status_type type)

typedef enum

{

    LED_STATUS_ON = 0,

    LED_STATUS_OFF

}led_status_type;
```