**Audio\_hw.c (vendor\modules\audio\normal)**

**struct audio\_module HAL\_MODULE\_INFO\_SYM = {**

**.common = {**

**.tag = HARDWARE\_MODULE\_TAG,**

**.module\_api\_version = AUDIO\_MODULE\_API\_VERSION\_0\_1,**

**.hal\_api\_version = HARDWARE\_HAL\_API\_VERSION,**

**.id = AUDIO\_HARDWARE\_MODULE\_ID,**

**.name = "Spreadtrum Audio HW HAL",**

**.author = "The Android Open Source Project",**

**.methods = &hal\_module\_methods,**

**},**

**};**

**--->hal\_module\_methods**

**--->.open = adev\_open**

**--->//实现hal层的接口方法**

**adev->hw\_device.common.close = adev\_close;**

**adev->hw\_device.get\_supported\_devices = adev\_get\_supported\_devices;**

**adev->hw\_device.init\_check = adev\_init\_check;**

**adev->hw\_device.set\_voice\_volume = adev\_set\_voice\_volume;**

**adev->hw\_device.set\_master\_volume = adev\_set\_master\_volume;**

**adev->hw\_device.set\_mode = adev\_set\_mode;**

**adev->hw\_device.set\_master\_mute = adev\_set\_master\_mute;**

**adev->hw\_device.get\_master\_mute = adev\_get\_master\_mute;**

**adev->hw\_device.set\_mic\_mute = adev\_set\_mic\_mute;**

**adev->hw\_device.get\_mic\_mute = adev\_get\_mic\_mute;**

**adev->hw\_device.set\_parameters = adev\_set\_parameters;**

**adev->hw\_device.get\_parameters = adev\_get\_parameters;**

**adev->hw\_device.get\_input\_buffer\_size = adev\_get\_input\_buffer\_size;**

**adev->hw\_device.open\_output\_stream = adev\_open\_output\_stream;**

**adev->hw\_device.close\_output\_stream = adev\_close\_output\_stream;**

**adev->hw\_device.open\_input\_stream = adev\_open\_input\_stream;**

**adev->hw\_device.close\_input\_stream = adev\_close\_input\_stream;**

**adev->hw\_device.dump = adev\_dump;**

**adev->hw\_device.get\_microphones = adev\_get\_microphones;**

**adev->hw\_device.set\_audio\_port\_config = adev\_set\_audio\_port\_config;**

**adev->hw\_device.create\_audio\_patch = adev\_create\_audio\_patch;**

**adev->hw\_device.release\_audio\_patch = adev\_release\_audio\_patch;**

**ret = adev\_modem\_parse(adev);**

**//获取音频参数**

**/\* get audio para from audio\_para.txt\*/**

**vb\_effect\_getpara(adev);**

**vb\_effect\_setpara(adev->audio\_para);**

**//查询声卡**

**/\* query sound cards\*/**

**s\_tinycard =** **get\_snd\_card\_number(CARD\_SPRDPHONE);**

**s\_vaudio = get\_snd\_card\_number(CARD\_VAUDIO);**

**s\_voip = get\_snd\_card\_number(CARD\_SCO);**

**s\_bt\_sco = get\_snd\_card\_number(CARD\_BT\_SCO);//外置codec的使用**

**s\_vaudio\_w = get\_snd\_card\_number(CARD\_VAUDIO\_W);**

**s\_vaudio\_lte = get\_snd\_card\_number(CARD\_VAUDIO\_LTE);**

**1:打开ap的声卡和mixer**

**--->1:** **adev->mixer = mixer\_open(s\_tinycard);**

**2:打开vbc\_access\_en**

**2:--->** **adev->vbc\_access = mixer\_get\_ctl\_by\_name(adev->mixer,"vbc\_access\_en");**

**3:设置vbc mixer 为0**

**3:--->** **mixer\_ctl\_set\_value(adev->vbc\_access, 0, 0);**

**4:读取音频xml文件**

**4:--->** **ret = adev\_config\_parse(adev);**

**4.1:--->** **set\_route\_by\_array(s->adev->mixer, s->path, s->path\_len);**

**5:--->** **init\_vbc\_eq(adev);**

**6:---> adev->pga = audio\_pga\_init(adev->mixer);**

**7:** **//switch to arm**

**7:--->** **mixer\_ctl\_set\_value(adev->private\_ctl.vbc\_switch, 0, VBC\_ARM\_CHANNELID);**

**8:--->** **adev->vbc\_2arm = mixer\_ctl\_get\_value(adev->private\_ctl.vbc\_switch,0);**

**9: /\* Create a task to get vbpipe message from cp when voice-call \*/**

**9:--->** **vbc\_ctrl\_open(adev)**

**10:--->** **ret = audio\_bt\_sco\_thread\_create(adev);**

**11:--->** **voice\_command\_manager\_create(adev)**

**12:--->** **NXP\_SMART\_PA**

**12:--->** **adev->i2s\_mixer = mixer\_open(s\_bt\_sco);**

音乐播放流程

1:AudioTrack: set(): streamType 3(AUDIO\_STREAM\_MUSIC = 3), sampleRate 44100, format 0x1000000(AUDIO\_FORMAT\_MP3= 0x01000000u), channelMask 0x3, frameCount 0, flags #10(AUDIO\_OUTPUT\_FLAG\_COMPRESS\_OFFLOAD = 0x10), notificationFrames 0, sessionId 25, transferType 1, uid 10025, pid 30468

2：AudioFlinger: openOutput() this 0xedcbe000, module 10 Device 0x2, SamplingRate 44100, Format 0x1000000, Channels 0x3, flags 0x31

3：audio\_hw\_primary: adev\_open\_output\_stream, devices = 0x2 flags:0x31 rate:44100

4：audio\_hw\_primary: adev\_open\_output\_stream out:0xf5118c40

5：audio\_hw\_primary: compress\_config fragment\_size:0x5000 fragments:0x40

compress\_offload：硬解输出流设备，用于需要硬件解码的数据输出，对应着标识为 AUDIO\_OUTPUT\_FLAG\_COMPRESS\_OFFLOAD 的音频流和一个 OffloadThread 回放线程实例 ,硬解码有: AUDIO\_FORMAT\_MP3和AUDIO\_FORMAT\_AAC

6：audio\_hw\_offload: audio\_offload\_create\_thread, successful, id:4095334768//创建硬解码线程。

7: audio\_hw\_primary: adev\_open\_output\_stream pcm config:48000 960 960

8: audio\_hw\_primary: audio\_add\_output type:1(三种: 原始PRIMARY:0 需要硬件编解码OFFLOAD:1 VOICE\_TX:2 语音) out:0xf5118c40

9: audio\_hw\_offload: audio\_offload\_thread\_loop in

10: audio\_hw\_primary: adev\_open\_output\_stream Successful audio\_app\_type:1 out:0xf5118c40

11: audio\_hw\_offload: audio\_offload\_thread\_loop audio\_offload\_cmd\_list is empty,wait

12: AudioFlinger: HAL output buffer size 20480 frames, normal sink buffer size 20480 frames

13: audio\_hw\_primary: adev\_set\_voice\_volume volume:0.800000 level:4

14: AudioFlinger: AudioFlinger's thread 0xe9b8b000 tid=30590 ready to run

15: APM\_AudioPolicyManager: getOutputForAttr() returns output 29 selectedDeviceId 3(AUDIO\_STREAM\_MUSIC = 3)

16: AudioTrack: start mClientUid:10025 mClientPid:30468 mSessionId:25 app name:/system/bin/mediaserver

17: APM\_AudioPolicyManagerSPRD: startOutput() output 29, stream 3 (MUSIC:3), session 25

18: AudioFlinger: moveEffectChain\_l() effect chain for session 0 not on source thread 0xeab83400

19: audio\_hw\_primary: adev\_create\_audio\_patch: source[0] type=2 (AUDIO\_PORT\_TYPE\_DEVICE:1 AUDIO\_PORT\_TYPE\_MIX:2)address=

20: audio\_hw\_primary: AUDIO\_PORT\_TYPE\_MIX source handle:29 hw\_module:10 stream:-1 source:-1

21: audio\_hw\_primary: adev\_create\_audio\_patch: sink[0] type=1(AUDIO\_PORT\_TYPE\_DEVICE:1 AUDIO\_PORT\_TYPE\_MIX:2) address=

22: audio\_hw\_primary: AUDIO\_PORT\_TYPE\_DEVICE sink device type:0x2 hw\_module:10

23: audio\_hw\_primary: sources\_device\_type:0 sinks\_device\_type:2

24: audio\_hw\_primary: adev\_out\_apm\_devices\_check update apm\_devices:0x2

25: audio\_hw\_primary: adev\_out\_apm\_devices\_check update apm\_devices:0x2

26: audio\_hw\_control: select\_devices\_new devices 0x2, is in 0 app type:-1 sync is 0,force:0

27: APM\_AudioPolicyManagerSPRD: startOutput() isVoipSet 0,stream 3,

28: audio\_hw\_primary: out\_set\_parameters type:1:delay\_samples=576;music\_offload\_avg\_bit\_rate=192000;music\_offload\_sample\_rate=44100;padding\_samples=2208

29：audio\_hw\_offload: audio\_get\_compress\_metadata successfully, new encoder\_delay: 576, encoder\_padding: 2208

30：audio\_hw\_offload: audio\_get\_compress\_metadata successfully, new samplerate:44100, out->mdata\_channel: 0, bitrate: 192000

31：audio\_hw\_control: do\_select\_device in device:2,is\_in:0,force\_set:0,actl->usecase:0,in:0,out:2

32：audio\_hw\_control: do\_select\_device usecase is 0

33：audio\_hw\_primary: out\_write start up bytes:20480

34：audio\_hw\_control: set\_usecase cur :0x0 usecase=0x400 on

35：audio\_hw\_dsp: dsp\_sleep\_ctrl\_l:1

36：audio\_hw\_dsp: dsp\_sleep\_ctrl: count:0,on\_off:1,dsp\_ctl->agdsp\_sleep\_status:1

37：audio\_hw\_SmartAmp: FbSmartamp is not Calibrated

38：audio\_hw\_dsp: agdsp\_send\_msg cmd:0x29 param:0x1 0x0 0x0 0xf5eac440

39：audio\_hw\_control: switch\_vbc\_route iis in,device:2 usecase:400

40：audio\_hw\_control: VBC OUT DEVICES only\_codec\_p Route ON

41：audio\_hw\_dsp: agdsp\_msg\_process cmd:0x29 parameter:0x1 0x0 0x0

42：audio\_hw\_dsp: begin to receive agdsp pipe message is\_exit:0

43：audio\_hw\_control: Set 'VBC\_IIS\_TX0\_WD\_SEL' to 'WD\_24BIT'

44：audio\_hw\_control: Set 'VBC\_MUX\_DAC0\_IIS\_PORT\_SEL' to 'VBC\_IIS\_PORT\_IIS0'

45：audio\_hw\_control: Set 'VBC\_MUX\_DAC1\_IIS\_PORT\_SEL' to 'VBC\_IIS\_PORT\_IIS0'

46：audio\_hw\_control: Set 'VBC\_MUX\_IIS0\_PORT\_DO\_SEL' to 'IIS\_DO\_VAL\_DAC0'

47：audio\_hw\_control: Set 'VBC\_MUX\_IIS2\_PORT\_DO\_SEL' to 'IIS\_DO\_VAL\_DAC2'

audio\_hw\_control: Set 'VBC\_MUX\_IIS3\_PORT\_DO\_SEL' to 'IIS\_DO\_VAL\_DAC2'

audio\_hw\_control: switch\_vbc\_route iis out be in,device:2

audio\_hw\_control: VBC OUT DEVICES codec\_p Route ON

audio\_hw\_control: Set 'ag\_iis0\_ext\_sel' to 'enable'

audio\_hw\_control: Set 'S\_NORMAL\_AP01\_P\_CODEC SWITCH' to 1

audio\_hw\_control: Set 'S\_NORMAL\_AP23\_P\_CODEC SWITCH' to 1

audio\_hw\_control: Set 'S\_FAST\_P\_CODEC SWITCH' to 1

audio\_hw\_control: Set 'S\_OFFLOAD\_CODEC SWITCH' to 1

audio\_hw\_control: Set 'S\_VOICE\_P\_CODEC SWITCH' to 1

audio\_hw\_control: Set 'S\_VOIP\_P\_CODEC SWITCH' to 1

audio\_hw\_control: Set 'S\_FM\_CODEC SWITCH' to 1

audio\_hw\_control: Set 'S\_LOOP\_P\_CODEC SWITCH' to 1

audio\_hw\_control: Set 'S\_FM\_DSP\_CODEC SWITCH' to 1

audio\_hw\_control: switch\_vbc\_route be out,device:2

audio\_hw\_offload: offload\_samplerate:44100

48：audio\_hw\_offload: audio\_start\_compress\_output: compress\_open out compress:0xf3d830a0 app\_type:1

49：

49：

audio\_hw\_control: \_set\_mdg\_mute:0xf5ebe884 0

audio\_hw\_control: select\_devices\_new devices 0x2, is in 0 app type:1 sync is 1,force:1

audio\_hw\_control: do\_select\_device in device:2,is\_in:0,force\_set:1,actl->usecase:400,in:0,out:2

audio\_hw\_control: device\_access\_enable

audio\_hw\_dsp: dsp\_sleep\_ctrl\_pair: count:1,on\_off:1,dsp\_ctl->agdsp\_sleep\_status:1

audio\_hw\_control: switch\_vbc\_route\_unlock:2 usecase:400

audio\_hw\_control: OUT DEVICES speaker Route ON

audio\_hw\_control: Set 'SPKL Mixer DACLSPKL Switch' to 1

ServiceManagement: Waited one second for vendor.sprd.hardware.radio@1.0::IExtRadio/slot1. Waiting another...

audio\_hw\_control: Set 'Speaker Function' to 1

audio\_hw\_SmartAmp: FbSmartamp is not Calibrated

audio\_hw\_control: device\_access\_restore

audio\_hw\_dsp: dsp\_sleep\_ctrl\_pair: count:0,on\_off:0,dsp\_ctl->agdsp\_sleep\_status:1

audio\_hw\_control: do\_select\_device out device:2,is\_in:0

audio\_hw\_control: set\_offload\_volume left=0.023646,right=0.023646, max=4096

CPLOG\_CONNMGR: dump\_notifier\_process: It has no notifier client [0(Success)]

audio\_hw\_control: clear\_all\_vbc\_dg\_param\_state

audio\_hw\_control: UPDATE\_PARAM\_VDG:Music\Handsfree\Playback volme:0

sensor : PlsSensor: mPendingEvents[Light].light = 55.000000

enhanceHAL: slpSetBrightness: ambient = 55

SLOGCP : try\_open: open /dev/spipe\_nr0 error [19(No such device)]

SLOGCP : try\_open: open /dev/spipe\_nr0 error [19(No such device)]

MediaPlaybackActivity: main handle msg:4

MediaPlaybackActivity: REFESH\_POSITION\_MSG: next= 172

audio\_hw\_control: Apply Gain Control [VBC DAC0 DG Set. 33]

audio\_hw\_param: get\_audio\_param\_mode type:0 param\_id:61 param\_mode:39 default\_mode:3d

audio\_hw\_control: set\_vbc\_param:61 case:0 255 255

50：audio\_hw\_control: UPDATE\_PARAM\_VBC\_PLAY:Music\Handsfree\Playback play:255 dsp\_case:0

51：audio\_hw\_param: get\_audio\_param\_mode type:4 param\_id:61 param\_mode:27 default\_mode:3d

audio\_hw\_control: UPDATE\_PARAM\_CODEC\_PLAY:Music\Handsfree\Playback cur\_codec\_p\_volume:255 volume:0 codec\_param:0xf3285970

audio\_hw\_control: set\_sprd\_output\_devices\_param vol\_index:0 out\_devices:2

audio\_hw\_control: set\_sprd\_output\_devices\_param set dacs\_playback\_volume :0x3

audio\_hw\_control: set\_sprd\_output\_devices\_param set spkl\_playback\_volume :0x0

audio\_hw\_control: set\_sprd\_output\_devices\_param set inner\_pa :0x12f

52：audio\_hw\_offload: out\_write\_compress start bytes:20480 state:0

53：audio\_hw\_offload: out\_write\_compress: need to send new metadata to driver

54：audio\_hw\_offload: out\_write\_compress end ret:20480

55：audio\_hw\_offload: out\_write\_compress start bytes:20480 state:1

56：audio\_hw\_offload: out\_write\_compress end ret:20480

57：audio\_hw\_offload: out\_write\_compress start bytes:20480 state:1

58：audio\_hw\_offload: out\_write\_compress end ret:20480

59：audio\_hw\_offload: out\_write\_compress start bytes:4096 state:1

60：

vim av/media/libaudioclient/IAudioFlinger.cpp +1145

设备打开流程:

设备：audio\_devices\_t

static struct platform\_driver sprd\_pcm\_driver = {

.driver = {

.name = "sprd-pcm-audio",

.owner = THIS\_MODULE,

.of\_match\_table = sprd\_pcm\_of\_match,

},

.probe = sprd\_soc\_platform\_probe,

.remove = sprd\_soc\_platform\_remove,

};

sprd\_soc\_platform\_probe

--->snd\_soc\_register\_platform(&pdev->dev, &sprd\_soc\_platform)

static struct snd\_soc\_platform\_driver sprd\_soc\_platform = {

.ops = &sprd\_pcm\_ops,

.probe = sprd\_snd\_platform\_probe,

.remove = sprd\_snd\_platform\_remove,

.pcm\_new = sprd\_pcm\_new,

.pcm\_free = sprd\_pcm\_free\_dma\_buffers,

};

音频设备

enum {

AUDIO\_DEVICE\_NONE = 0x0u,

AUDIO\_DEVICE\_BIT\_IN = 0x80000000u,

AUDIO\_DEVICE\_BIT\_DEFAULT = 0x40000000u,

AUDIO\_DEVICE\_OUT\_EARPIECE = 0x1u,

AUDIO\_DEVICE\_OUT\_SPEAKER = 0x2u,

AUDIO\_DEVICE\_OUT\_WIRED\_HEADSET = 0x4u,

AUDIO\_DEVICE\_OUT\_WIRED\_HEADPHONE = 0x8u,

AUDIO\_DEVICE\_OUT\_BLUETOOTH\_SCO = 0x10u,

AUDIO\_DEVICE\_OUT\_BLUETOOTH\_SCO\_HEADSET = 0x20u,

AUDIO\_DEVICE\_OUT\_BLUETOOTH\_SCO\_CARKIT = 0x40u,

AUDIO\_DEVICE\_OUT\_BLUETOOTH\_A2DP = 0x80u,

AUDIO\_DEVICE\_OUT\_BLUETOOTH\_A2DP\_HEADPHONES = 0x100u,

AUDIO\_DEVICE\_OUT\_BLUETOOTH\_A2DP\_SPEAKER = 0x200u,

AUDIO\_DEVICE\_OUT\_AUX\_DIGITAL = 0x400u,

AUDIO\_DEVICE\_OUT\_HDMI = 0x400u, // OUT\_AUX\_DIGITAL

AUDIO\_DEVICE\_OUT\_ANLG\_DOCK\_HEADSET = 0x800u,

AUDIO\_DEVICE\_OUT\_DGTL\_DOCK\_HEADSET = 0x1000u,

AUDIO\_DEVICE\_OUT\_USB\_ACCESSORY = 0x2000u,

AUDIO\_DEVICE\_OUT\_USB\_DEVICE = 0x4000u,

AUDIO\_DEVICE\_OUT\_REMOTE\_SUBMIX = 0x8000u,

AUDIO\_DEVICE\_OUT\_TELEPHONY\_TX = 0x10000u,

AUDIO\_DEVICE\_OUT\_LINE = 0x20000u,

AUDIO\_DEVICE\_OUT\_HDMI\_ARC = 0x40000u,

AUDIO\_DEVICE\_OUT\_SPDIF = 0x80000u,

AUDIO\_DEVICE\_OUT\_FM = 0x100000u,

AUDIO\_DEVICE\_OUT\_AUX\_LINE = 0x200000u,

AUDIO\_DEVICE\_OUT\_SPEAKER\_SAFE = 0x400000u,

AUDIO\_DEVICE\_OUT\_IP = 0x800000u,

AUDIO\_DEVICE\_OUT\_BUS = 0x1000000u,

AUDIO\_DEVICE\_OUT\_PROXY = 0x2000000u,

AUDIO\_DEVICE\_OUT\_USB\_HEADSET = 0x4000000u,

AUDIO\_DEVICE\_OUT\_HEARING\_AID = 0x8000000u,

AUDIO\_DEVICE\_OUT\_ECHO\_CANCELLER = 0x10000000u,

AUDIO\_DEVICE\_OUT\_DEFAULT = 0x40000000u, // BIT\_DEFAULT

AUDIO\_DEVICE\_IN\_COMMUNICATION = 0x80000001u, // BIT\_IN | 0x1

AUDIO\_DEVICE\_IN\_AMBIENT = 0x80000002u, // BIT\_IN | 0x2

AUDIO\_DEVICE\_IN\_BUILTIN\_MIC = 0x80000004u, // BIT\_IN | 0x4

AUDIO\_DEVICE\_IN\_BLUETOOTH\_SCO\_HEADSET = 0x80000008u, // BIT\_IN | 0x8

AUDIO\_DEVICE\_IN\_WIRED\_HEADSET = 0x80000010u, // BIT\_IN | 0x10

AUDIO\_DEVICE\_IN\_AUX\_DIGITAL = 0x80000020u, // BIT\_IN | 0x20

AUDIO\_DEVICE\_IN\_HDMI = 0x80000020u, // IN\_AUX\_DIGITAL

AUDIO\_DEVICE\_IN\_VOICE\_CALL = 0x80000040u, // BIT\_IN | 0x40

AUDIO\_DEVICE\_IN\_TELEPHONY\_RX = 0x80000040u, // IN\_VOICE\_CALL

AUDIO\_DEVICE\_IN\_BACK\_MIC = 0x80000080u, // BIT\_IN | 0x80

AUDIO\_DEVICE\_IN\_REMOTE\_SUBMIX = 0x80000100u, // BIT\_IN | 0x100

AUDIO\_DEVICE\_IN\_ANLG\_DOCK\_HEADSET = 0x80000200u, // BIT\_IN | 0x200

AUDIO\_DEVICE\_IN\_DGTL\_DOCK\_HEADSET = 0x80000400u, // BIT\_IN | 0x400

AUDIO\_DEVICE\_IN\_USB\_ACCESSORY = 0x80000800u, // BIT\_IN | 0x800

AUDIO\_DEVICE\_IN\_USB\_DEVICE = 0x80001000u, // BIT\_IN | 0x1000

AUDIO\_DEVICE\_IN\_FM\_TUNER = 0x80002000u, // BIT\_IN | 0x2000

AUDIO\_DEVICE\_IN\_TV\_TUNER = 0x80004000u, // BIT\_IN | 0x4000

AUDIO\_DEVICE\_IN\_LINE = 0x80008000u, // BIT\_IN | 0x8000

AUDIO\_DEVICE\_IN\_SPDIF = 0x80010000u, // BIT\_IN | 0x10000

AUDIO\_DEVICE\_IN\_BLUETOOTH\_A2DP = 0x80020000u, // BIT\_IN | 0x20000

AUDIO\_DEVICE\_IN\_LOOPBACK = 0x80040000u, // BIT\_IN | 0x40000

AUDIO\_DEVICE\_IN\_IP = 0x80080000u, // BIT\_IN | 0x80000

AUDIO\_DEVICE\_IN\_BUS = 0x80100000u, // BIT\_IN | 0x100000

AUDIO\_DEVICE\_IN\_PROXY = 0x81000000u, // BIT\_IN | 0x1000000

AUDIO\_DEVICE\_IN\_USB\_HEADSET = 0x82000000u, // BIT\_IN | 0x2000000

AUDIO\_DEVICE\_IN\_BLUETOOTH\_BLE = 0x84000000u, // BIT\_IN | 0x4000000

AUDIO\_DEVICE\_IN\_DEFAULT = 0xC0000000u, // BIT\_IN | BIT\_DEFAULT

};

这个音频设备的值 最终对应为audio\_route.xml来具体化:

如spk :AUDIO\_DEVICE\_OUT\_SPEAKER = 0x2u,

<speaker device="0x2">

路由选择:

通话时:用设备的听筒和喇叭

<ctl name="VBC\_MUX\_ADC3\_IIS\_PORT\_SEL" val="VBC\_IIS\_PORT\_IIS3"/>

Usb 耳机通话:

<ctl name="VBC\_MUX\_ADC3\_IIS\_PORT\_SEL" val="VBC\_IIS\_PORT\_IIS2" />

蓝牙通话：

<ctl name="VBC\_MUX\_ADC3\_IIS\_PORT\_SEL" val="VBC\_IIS\_PORT\_IIS0"/>

// input devices

public static final int DEVICE\_IN\_COMMUNICATION = DEVICE\_BIT\_IN | 0x1;

public static final int DEVICE\_IN\_AMBIENT = DEVICE\_BIT\_IN | 0x2;

public static final int DEVICE\_IN\_BUILTIN\_MIC = DEVICE\_BIT\_IN | 0x4;

public static final int DEVICE\_IN\_BLUETOOTH\_SCO\_HEADSET = DEVICE\_BIT\_IN | 0x8;

public static final int DEVICE\_IN\_WIRED\_HEADSET = DEVICE\_BIT\_IN | 0x10;

public static final int DEVICE\_IN\_AUX\_DIGITAL = DEVICE\_BIT\_IN | 0x20;

public static final int DEVICE\_IN\_HDMI = DEVICE\_IN\_AUX\_DIGITAL;

public static final int DEVICE\_IN\_VOICE\_CALL = DEVICE\_BIT\_IN | 0x40;

public static final int DEVICE\_IN\_TELEPHONY\_RX = DEVICE\_IN\_VOICE\_CALL;

public static final int DEVICE\_IN\_BACK\_MIC = DEVICE\_BIT\_IN | 0x80;

public static final int DEVICE\_IN\_REMOTE\_SUBMIX = DEVICE\_BIT\_IN | 0x100;

public static final int DEVICE\_IN\_ANLG\_DOCK\_HEADSET = DEVICE\_BIT\_IN | 0x200;

public static final int DEVICE\_IN\_DGTL\_DOCK\_HEADSET = DEVICE\_BIT\_IN | 0x400;

public static final int DEVICE\_IN\_USB\_ACCESSORY = DEVICE\_BIT\_IN | 0x800;

public static final int DEVICE\_IN\_USB\_DEVICE = DEVICE\_BIT\_IN | 0x1000;

public static final int DEVICE\_IN\_FM\_TUNER = DEVICE\_BIT\_IN | 0x2000;

public static final int DEVICE\_IN\_TV\_TUNER = DEVICE\_BIT\_IN | 0x4000;

public static final int DEVICE\_IN\_LINE = DEVICE\_BIT\_IN | 0x8000;

public static final int DEVICE\_IN\_SPDIF = DEVICE\_BIT\_IN | 0x10000;

public static final int DEVICE\_IN\_BLUETOOTH\_A2DP = DEVICE\_BIT\_IN | 0x20000;

public static final int DEVICE\_IN\_LOOPBACK = DEVICE\_BIT\_IN | 0x40000;

public static final int DEVICE\_IN\_IP = DEVICE\_BIT\_IN | 0x80000;

public static final int DEVICE\_IN\_BUS = DEVICE\_BIT\_IN | 0x100000;

public static final int DEVICE\_IN\_PROXY = DEVICE\_BIT\_IN | 0x1000000;

public static final int DEVICE\_IN\_USB\_HEADSET = DEVICE\_BIT\_IN | 0x2000000;

public static final int DEVICE\_IN\_DEFAULT = DEVICE\_BIT\_IN | DEVICE\_BIT\_DEFAULT;

public static final int DEVICE\_IN\_ALL = (DEVICE\_IN\_COMMUNICATION |

DEVICE\_IN\_AMBIENT |

DEVICE\_IN\_BUILTIN\_MIC |

DEVICE\_IN\_BLUETOOTH\_SCO\_HEADSET |

DEVICE\_IN\_WIRED\_HEADSET |

DEVICE\_IN\_HDMI |

DEVICE\_IN\_TELEPHONY\_RX |

DEVICE\_IN\_BACK\_MIC |

DEVICE\_IN\_REMOTE\_SUBMIX |

DEVICE\_IN\_ANLG\_DOCK\_HEADSET |

DEVICE\_IN\_DGTL\_DOCK\_HEADSET |

DEVICE\_IN\_USB\_ACCESSORY |

DEVICE\_IN\_USB\_DEVICE |

DEVICE\_IN\_FM\_TUNER |

DEVICE\_IN\_TV\_TUNER |

DEVICE\_IN\_LINE |

DEVICE\_IN\_SPDIF |

DEVICE\_IN\_BLUETOOTH\_A2DP |

DEVICE\_IN\_LOOPBACK |

DEVICE\_IN\_IP |

DEVICE\_IN\_BUS |

DEVICE\_IN\_PROXY |

DEVICE\_IN\_USB\_HEADSET |

DEVICE\_IN\_DEFAULT);

public static final int DEVICE\_IN\_ALL\_SCO = DEVICE\_IN\_BLUETOOTH\_SCO\_HEADSET;

public static final int DEVICE\_IN\_ALL\_USB = (DEVICE\_IN\_USB\_ACCESSORY |

DEVICE\_IN\_USB\_DEVICE |

DEVICE\_IN\_USB\_HEADSET);

AudioSystem.java (frameworks\base\media\java\android\media) 44580 2022/2/28

public static final int DEVICE\_NONE = 0x0;

// reserved bits

public static final int DEVICE\_BIT\_IN = 0x80000000;

public static final int DEVICE\_BIT\_DEFAULT = 0x40000000;

// output devices, be sure to update AudioManager.java also

public static final int DEVICE\_OUT\_EARPIECE = 0x1;

public static final int DEVICE\_OUT\_SPEAKER = 0x2;

public static final int DEVICE\_OUT\_WIRED\_HEADSET = 0x4;

public static final int DEVICE\_OUT\_WIRED\_HEADPHONE = 0x8;

public static final int DEVICE\_OUT\_BLUETOOTH\_SCO = 0x10;

public static final int DEVICE\_OUT\_BLUETOOTH\_SCO\_HEADSET = 0x20;

public static final int DEVICE\_OUT\_BLUETOOTH\_SCO\_CARKIT = 0x40;

public static final int DEVICE\_OUT\_BLUETOOTH\_A2DP = 0x80;

public static final int DEVICE\_OUT\_BLUETOOTH\_A2DP\_HEADPHONES = 0x100;

public static final int DEVICE\_OUT\_BLUETOOTH\_A2DP\_SPEAKER = 0x200;

public static final int DEVICE\_OUT\_AUX\_DIGITAL = 0x400;

public static final int DEVICE\_OUT\_HDMI = DEVICE\_OUT\_AUX\_DIGITAL;

public static final int DEVICE\_OUT\_ANLG\_DOCK\_HEADSET = 0x800;

public static final int DEVICE\_OUT\_DGTL\_DOCK\_HEADSET = 0x1000;

public static final int DEVICE\_OUT\_USB\_ACCESSORY = 0x2000;

public static final int DEVICE\_OUT\_USB\_DEVICE = 0x4000;

public static final int DEVICE\_OUT\_REMOTE\_SUBMIX = 0x8000;

public static final int DEVICE\_OUT\_TELEPHONY\_TX = 0x10000;

public static final int DEVICE\_OUT\_LINE = 0x20000;

public static final int DEVICE\_OUT\_HDMI\_ARC = 0x40000;

public static final int DEVICE\_OUT\_SPDIF = 0x80000;

public static final int DEVICE\_OUT\_FM = 0x100000;

/\*\* SPRD: bug687987, add FM devices: headset and speaker. @{ \*/

public static final int DEVICE\_OUT\_FM\_HEADSET = 0x10000000;

public static final int DEVICE\_OUT\_FM\_SPEAKER = 0x20000000;

/\*\* @} \*/

public static final int DEVICE\_OUT\_AUX\_LINE = 0x200000;

public static final int DEVICE\_OUT\_SPEAKER\_SAFE = 0x400000;

public static final int DEVICE\_OUT\_IP = 0x800000;

public static final int DEVICE\_OUT\_BUS = 0x1000000;

public static final int DEVICE\_OUT\_PROXY = 0x2000000;

public static final int DEVICE\_OUT\_USB\_HEADSET = 0x4000000;

public static final int DEVICE\_OUT\_HEARING\_AID = 0x8000000;

public static final int DEVICE\_OUT\_DEFAULT = DEVICE\_BIT\_DEFAULT;

public static final int DEVICE\_OUT\_ALL = (DEVICE\_OUT\_EARPIECE |

DEVICE\_OUT\_SPEAKER |

DEVICE\_OUT\_WIRED\_HEADSET |

DEVICE\_OUT\_WIRED\_HEADPHONE |

DEVICE\_OUT\_BLUETOOTH\_SCO |

DEVICE\_OUT\_BLUETOOTH\_SCO\_HEADSET |

DEVICE\_OUT\_BLUETOOTH\_SCO\_CARKIT |

DEVICE\_OUT\_BLUETOOTH\_A2DP |

DEVICE\_OUT\_BLUETOOTH\_A2DP\_HEADPHONES |

DEVICE\_OUT\_BLUETOOTH\_A2DP\_SPEAKER |

DEVICE\_OUT\_HDMI |

DEVICE\_OUT\_ANLG\_DOCK\_HEADSET |

DEVICE\_OUT\_DGTL\_DOCK\_HEADSET |

DEVICE\_OUT\_USB\_ACCESSORY |

DEVICE\_OUT\_USB\_DEVICE |

DEVICE\_OUT\_REMOTE\_SUBMIX |

DEVICE\_OUT\_TELEPHONY\_TX |

DEVICE\_OUT\_LINE |

DEVICE\_OUT\_HDMI\_ARC |

DEVICE\_OUT\_SPDIF |

DEVICE\_OUT\_FM |

DEVICE\_OUT\_AUX\_LINE |

DEVICE\_OUT\_SPEAKER\_SAFE |

DEVICE\_OUT\_IP |

DEVICE\_OUT\_BUS |

DEVICE\_OUT\_PROXY |

DEVICE\_OUT\_USB\_HEADSET |

DEVICE\_OUT\_HEARING\_AID |

/\*\* SPRD: bug687987, add FM devices: headset and speaker. @{ \*/

DEVICE\_OUT\_FM\_HEADSET |

DEVICE\_OUT\_FM\_SPEAKER |

/\*\* @} \*/

DEVICE\_OUT\_DEFAULT);

public static final int DEVICE\_OUT\_ALL\_A2DP = (DEVICE\_OUT\_BLUETOOTH\_A2DP |

DEVICE\_OUT\_BLUETOOTH\_A2DP\_HEADPHONES |

DEVICE\_OUT\_BLUETOOTH\_A2DP\_SPEAKER);

public static final int DEVICE\_OUT\_ALL\_SCO = (DEVICE\_OUT\_BLUETOOTH\_SCO |

DEVICE\_OUT\_BLUETOOTH\_SCO\_HEADSET |

DEVICE\_OUT\_BLUETOOTH\_SCO\_CARKIT);

public static final int DEVICE\_OUT\_ALL\_USB = (DEVICE\_OUT\_USB\_ACCESSORY |

DEVICE\_OUT\_USB\_DEVICE |

DEVICE\_OUT\_USB\_HEADSET);

public static final int DEVICE\_OUT\_ALL\_HDMI\_SYSTEM\_AUDIO = (DEVICE\_OUT\_AUX\_LINE |

DEVICE\_OUT\_HDMI\_ARC |

DEVICE\_OUT\_SPDIF);

public static final int DEVICE\_ALL\_HDMI\_SYSTEM\_AUDIO\_AND\_SPEAKER =

(DEVICE\_OUT\_ALL\_HDMI\_SYSTEM\_AUDIO |

DEVICE\_OUT\_SPEAKER);

audio.h中音频设备定义

typedef enum {

UC\_CALL = 0x1,

UC\_VOIP = 0x2,

UC\_FM = 0x4,

UC\_NORMAL\_PLAYBACK = 0x8,

UC\_LOOP = 0x10,

UC\_MM\_RECORD = 0x20,

UC\_AGDSP\_ASSERT = 0x40,

UC\_FM\_RECORD = 0x80,

UC\_FAST\_PLAYBACK = 0x100,

UC\_DEEP\_BUFFER\_PLAYBACK = 0x200,

UC\_OFFLOAD\_PLAYBACK =0x400,

UC\_VOICE\_RECORD = 0x800,

UC\_VOIP\_RECORD = 0x1000,

UC\_AUDIO\_TEST = 0x2000,

UC\_BT\_RECORD = 0x4000,

UC\_VBC\_PCM\_DUMP = 0x8000,

UC\_BT\_OFFLOAD\_MIXER = 0x10000,

UC\_BT\_OFFLOAD = 0x20000,

UC\_MMAP\_PLAYBACK = UC\_BT\_OFFLOAD<<1,

UC\_MMAP\_RECORD = UC\_MMAP\_PLAYBACK<<1,

UC\_RECOGNITION = UC\_MMAP\_RECORD<<1,

UC\_SYSTEM\_UNSLEEP = UC\_RECOGNITION<<1,

UC\_UNKNOWN = 0x00000000,

} USECASE;

enum {

AUDIO\_DEVICE\_NONE = 0x0,

/\* reserved bits \*/

AUDIO\_DEVICE\_BIT\_IN = 0x80000000,

AUDIO\_DEVICE\_BIT\_DEFAULT = 0x40000000,

/\* output devices \*/

AUDIO\_DEVICE\_OUT\_EARPIECE = 0x1, // 听筒

AUDIO\_DEVICE\_OUT\_SPEAKER = 0x2, // 扬声器

AUDIO\_DEVICE\_OUT\_WIRED\_HEADSET = 0x4, // 线控耳机，可以通过耳机控制远端播放、暂停、音量调节等功能的耳机

AUDIO\_DEVICE\_OUT\_WIRED\_HEADPHONE = 0x8, // 普通耳机，只能听，不能操控播放

AUDIO\_DEVICE\_OUT\_BLUETOOTH\_SCO = 0x10, // 单声道蓝牙耳机，十进制32

AUDIO\_DEVICE\_OUT\_BLUETOOTH\_SCO\_HEADSET = 0x20, // 车载免提蓝牙设备，十进制64

AUDIO\_DEVICE\_OUT\_BLUETOOTH\_SCO\_CARKIT = 0x40, // 立体声蓝牙耳机

AUDIO\_DEVICE\_OUT\_BLUETOOTH\_A2DP = 0x80, // 十进制128

AUDIO\_DEVICE\_OUT\_BLUETOOTH\_A2DP\_HEADPHONES = 0x100, // 十进制256

AUDIO\_DEVICE\_OUT\_BLUETOOTH\_A2DP\_SPEAKER = 0x200, // 十进制512

AUDIO\_DEVICE\_OUT\_AUX\_DIGITAL = 0x400, // 十进制1024

AUDIO\_DEVICE\_OUT\_ANLG\_DOCK\_HEADSET = 0x800, // 十进制2048

AUDIO\_DEVICE\_OUT\_DGTL\_DOCK\_HEADSET = 0x1000, // 十进制4096

AUDIO\_DEVICE\_OUT\_USB\_ACCESSORY = 0x2000,

AUDIO\_DEVICE\_OUT\_USB\_DEVICE = 0x4000,

AUDIO\_DEVICE\_OUT\_REMOTE\_SUBMIX = 0x8000,

AUDIO\_DEVICE\_OUT\_DEFAULT = AUDIO\_DEVICE\_BIT\_DEFAULT,

AUDIO\_DEVICE\_OUT\_ALL = (AUDIO\_DEVICE\_OUT\_EARPIECE |

AUDIO\_DEVICE\_OUT\_SPEAKER |

AUDIO\_DEVICE\_OUT\_WIRED\_HEADSET |

AUDIO\_DEVICE\_OUT\_WIRED\_HEADPHONE |

AUDIO\_DEVICE\_OUT\_BLUETOOTH\_SCO |

AUDIO\_DEVICE\_OUT\_BLUETOOTH\_SCO\_HEADSET |

AUDIO\_DEVICE\_OUT\_BLUETOOTH\_SCO\_CARKIT |

AUDIO\_DEVICE\_OUT\_BLUETOOTH\_A2DP |

AUDIO\_DEVICE\_OUT\_BLUETOOTH\_A2DP\_HEADPHONES |

AUDIO\_DEVICE\_OUT\_BLUETOOTH\_A2DP\_SPEAKER |

AUDIO\_DEVICE\_OUT\_AUX\_DIGITAL |

AUDIO\_DEVICE\_OUT\_ANLG\_DOCK\_HEADSET |

AUDIO\_DEVICE\_OUT\_DGTL\_DOCK\_HEADSET |

AUDIO\_DEVICE\_OUT\_USB\_ACCESSORY |

AUDIO\_DEVICE\_OUT\_USB\_DEVICE |

AUDIO\_DEVICE\_OUT\_REMOTE\_SUBMIX |

AUDIO\_DEVICE\_OUT\_DEFAULT),

AUDIO\_DEVICE\_OUT\_ALL\_A2DP = (AUDIO\_DEVICE\_OUT\_BLUETOOTH\_A2DP |

AUDIO\_DEVICE\_OUT\_BLUETOOTH\_A2DP\_HEADPHONES |

AUDIO\_DEVICE\_OUT\_BLUETOOTH\_A2DP\_SPEAKER),

AUDIO\_DEVICE\_OUT\_ALL\_SCO = (AUDIO\_DEVICE\_OUT\_BLUETOOTH\_SCO |

AUDIO\_DEVICE\_OUT\_BLUETOOTH\_SCO\_HEADSET |

AUDIO\_DEVICE\_OUT\_BLUETOOTH\_SCO\_CARKIT),

AUDIO\_DEVICE\_OUT\_ALL\_USB = (AUDIO\_DEVICE\_OUT\_USB\_ACCESSORY |

AUDIO\_DEVICE\_OUT\_USB\_DEVICE),

/\* input devices \*/

AUDIO\_DEVICE\_IN\_COMMUNICATION = AUDIO\_DEVICE\_BIT\_IN | 0x1,

AUDIO\_DEVICE\_IN\_AMBIENT = AUDIO\_DEVICE\_BIT\_IN | 0x2,

AUDIO\_DEVICE\_IN\_BUILTIN\_MIC = AUDIO\_DEVICE\_BIT\_IN | 0x4,　　//手机自带MIC

AUDIO\_DEVICE\_IN\_BLUETOOTH\_SCO\_HEADSET = AUDIO\_DEVICE\_BIT\_IN | 0x8,

AUDIO\_DEVICE\_IN\_WIRED\_HEADSET = AUDIO\_DEVICE\_BIT\_IN | 0x10,　　//耳机

AUDIO\_DEVICE\_IN\_AUX\_DIGITAL = AUDIO\_DEVICE\_BIT\_IN | 0x20,

AUDIO\_DEVICE\_IN\_VOICE\_CALL = AUDIO\_DEVICE\_BIT\_IN | 0x40,

AUDIO\_DEVICE\_IN\_BACK\_MIC = AUDIO\_DEVICE\_BIT\_IN | 0x80,

AUDIO\_DEVICE\_IN\_REMOTE\_SUBMIX = AUDIO\_DEVICE\_BIT\_IN | 0x100,

AUDIO\_DEVICE\_IN\_ANLG\_DOCK\_HEADSET = AUDIO\_DEVICE\_BIT\_IN | 0x200,

AUDIO\_DEVICE\_IN\_DGTL\_DOCK\_HEADSET = AUDIO\_DEVICE\_BIT\_IN | 0x400,

AUDIO\_DEVICE\_IN\_USB\_ACCESSORY = AUDIO\_DEVICE\_BIT\_IN | 0x800,

AUDIO\_DEVICE\_IN\_USB\_DEVICE = AUDIO\_DEVICE\_BIT\_IN | 0x1000,

AUDIO\_DEVICE\_IN\_DEFAULT = AUDIO\_DEVICE\_BIT\_IN | AUDIO\_DEVICE\_BIT\_DEFAULT,

AUDIO\_DEVICE\_IN\_ALL = (AUDIO\_DEVICE\_IN\_COMMUNICATION |

AUDIO\_DEVICE\_IN\_AMBIENT |

AUDIO\_DEVICE\_IN\_BUILTIN\_MIC |

AUDIO\_DEVICE\_IN\_BLUETOOTH\_SCO\_HEADSET |

AUDIO\_DEVICE\_IN\_WIRED\_HEADSET |

AUDIO\_DEVICE\_IN\_AUX\_DIGITAL |

AUDIO\_DEVICE\_IN\_VOICE\_CALL |

AUDIO\_DEVICE\_IN\_BACK\_MIC |

AUDIO\_DEVICE\_IN\_REMOTE\_SUBMIX |

AUDIO\_DEVICE\_IN\_ANLG\_DOCK\_HEADSET |

AUDIO\_DEVICE\_IN\_DGTL\_DOCK\_HEADSET |

AUDIO\_DEVICE\_IN\_USB\_ACCESSORY |

AUDIO\_DEVICE\_IN\_USB\_DEVICE |

AUDIO\_DEVICE\_IN\_DEFAULT),

AUDIO\_DEVICE\_IN\_ALL\_SCO = AUDIO\_DEVICE\_IN\_BLUETOOTH\_SCO\_HEADSET,

};

AUDIO\_DEVICE\_OUT\_EARPIECE = 0x1,// 听筒

AUDIO\_DEVICE\_OUT\_SPEAKER = 0x2,// 扬声器

AUDIO\_DEVICE\_OUT\_WIRED\_HEADSET = 0x4,//线控耳机

AUDIO\_DEVICE\_OUT\_WIRED\_HEADPHONE = 0x8,//普通耳机

AUDIO\_DEVICE\_OUT\_BLUETOOTH\_SCO = 0x10,//单声道蓝牙耳机

AUDIO\_DEVICE\_OUT\_BLUETOOTH\_SCO\_HEADSET = 0x20,//蓝牙电话

AUDIO\_DEVICE\_OUT\_BLUETOOTH\_SCO\_CARKIT = 0x40, //车载免提蓝牙设备

AUDIO\_DEVICE\_OUT\_BLUETOOTH\_A2DP = 0x80, //立体声蓝牙耳机

定义每种模式的DMA 是与ap 上层交互的audio 数据

static struct sprd\_pcm\_dma\_params vbc\_pcm\_normal\_ap01\_p;

static struct sprd\_pcm\_dma\_params vbc\_pcm\_normal\_ap01\_c;

static struct sprd\_pcm\_dma\_params vbc\_pcm\_normal\_ap23\_p;

static struct sprd\_pcm\_dma\_params vbc\_pcm\_normal\_ap23\_c;

static struct sprd\_pcm\_dma\_params pcm\_dsp\_cap\_mcdt;

static struct sprd\_pcm\_dma\_params pcm\_fast\_play\_mcdt;

static struct sprd\_pcm\_dma\_params pcm\_hifi\_fast\_play\_mcdt;

static struct sprd\_pcm\_dma\_params vbc\_pcm\_voice\_capture\_mcdt;

static struct sprd\_pcm\_dma\_params pcm\_loop\_record\_mcdt;

static struct sprd\_pcm\_dma\_params pcm\_loop\_play\_mcdt;

static struct sprd\_pcm\_dma\_params pcm\_voip\_record\_mcdt;

static struct sprd\_pcm\_dma\_params pcm\_voip\_play\_mcdt;

static struct sprd\_pcm\_dma\_params vbc\_pcm\_fm\_caputre;

static struct sprd\_pcm\_dma\_params vbc\_pcm\_a2dp\_p;

static struct sprd\_pcm\_dma\_params pcm\_dsp\_fm\_cap\_mcdt;

static struct sprd\_pcm\_dma\_params pcm\_dsp\_btsco\_cap\_mcdt;

static struct sprd\_pcm\_dma\_params vbc\_pcm\_dump;

static struct sprd\_pcm\_dma\_params vbc\_btsco\_cap\_ap;

static struct sprd\_pcm\_dma\_params vbc\_pcm\_recognise\_capture\_mcdt;

static struct sprd\_pcm\_dma\_params pcm\_voice\_play\_mcdt;

static struct sprd\_pcm\_dma\_params pcm\_hifi\_play\_mcdt;

static char \*fe\_dai\_id\_str[FE\_DAI\_ID\_MAX] = {

[FE\_DAI\_ID\_NORMAL\_AP01] = TO\_STRING(FE\_DAI\_ID\_NORMAL\_AP01),

[FE\_DAI\_ID\_NORMAL\_AP23] = TO\_STRING(FE\_DAI\_ID\_NORMAL\_AP23),

[FE\_DAI\_ID\_CAPTURE\_DSP] = TO\_STRING(FE\_DAI\_ID\_CAPTURE\_DSP),

[FE\_DAI\_ID\_FAST\_P] = TO\_STRING(FE\_DAI\_ID\_FAST\_P),

[FE\_DAI\_ID\_OFFLOAD] = TO\_STRING(FE\_DAI\_ID\_OFFLOAD),

[FE\_DAI\_ID\_VOICE] = TO\_STRING(FE\_DAI\_ID\_VOICE),

[FE\_DAI\_ID\_VOIP] = TO\_STRING(FE\_DAI\_ID\_VOIP),

[FE\_DAI\_ID\_FM] = TO\_STRING(FE\_DAI\_ID\_FM),

[FE\_DAI\_ID\_FM\_CAPTURE\_AP] = TO\_STRING(FE\_DAI\_ID\_FM\_CAPTURE\_AP),

[FE\_DAI\_ID\_VOICE\_CAPTURE] = TO\_STRING(FE\_DAI\_ID\_VOICE\_CAPTURE),

[FE\_DAI\_ID\_LOOP] = TO\_STRING(FE\_DAI\_ID\_LOOP),

[FE\_DAI\_ID\_A2DP\_OFFLOAD] = TO\_STRING(FE\_DAI\_ID\_A2DP\_OFFLOAD),

[FE\_DAI\_ID\_A2DP\_PCM] = TO\_STRING(FE\_DAI\_ID\_A2DP\_PCM),

[FE\_DAI\_ID\_FM\_CAP\_DSP] = TO\_STRING(FE\_DAI\_ID\_FM\_CAP\_DSP),

[FE\_DAI\_ID\_BTSCO\_CAP\_DSP] = TO\_STRING(FE\_DAI\_ID\_BTSCO\_CAP\_DSP),

[FE\_DAI\_ID\_FM\_DSP] = TO\_STRING(FE\_DAI\_ID\_FM\_DSP),

[FE\_DAI\_ID\_DUMP] = TO\_STRING(FE\_DAI\_ID\_DUMP),

[FE\_DAI\_ID\_BTSCO\_CAP\_AP] = TO\_STRING(FE\_DAI\_ID\_BTSCO\_CAP\_AP),

[FE\_DAI\_ID\_HFP] = TO\_STRING(FE\_DAI\_ID\_HFP),

[FE\_DAI\_ID\_RECOGNISE\_CAPTURE] = TO\_STRING(FE\_DAI\_ID\_RECOGNISE\_CAPTURE),

[FE\_DAI\_ID\_VOICE\_PCM\_P] = TO\_STRING(FE\_DAI\_ID\_VOICE\_PCM\_P),

[FE\_DAI\_ID\_HIFI\_P] = TO\_STRING(FE\_DAI\_ID\_HIFI\_P),

[FE\_DAI\_ID\_HIFI\_FAST\_P] = TO\_STRING(FE\_DAI\_ID\_HIFI\_FAST\_P),

};

释放fe-dai的mcdt dma

static void mcdt\_dma\_deinit(struct snd\_soc\_dai \*fe\_dai, int stream)

{

int is\_playback = stream == SNDRV\_PCM\_STREAM\_PLAYBACK ? 1 : 0;

switch (fe\_dai->id) {

case FE\_DAI\_ID\_FM:

case FE\_DAI\_ID\_OFFLOAD:

case FE\_DAI\_ID\_A2DP\_OFFLOAD:

case FE\_DAI\_ID\_VOICE:

case FE\_DAI\_ID\_FM\_DSP:

case FE\_DAI\_ID\_HFP:

default:

break;

case FE\_DAI\_ID\_CAPTURE\_DSP:

mcdt\_adc\_dma\_disable(MCDT\_CHAN\_DSP\_CAP);

break;

case FE\_DAI\_ID\_FM\_CAP\_DSP:

mcdt\_adc\_dma\_disable(MCDT\_CHAN\_DSP\_FM\_CAP);

break;

case FE\_DAI\_ID\_BTSCO\_CAP\_DSP:

mcdt\_adc\_dma\_disable(MCDT\_CHAN\_DSP\_BTSCO\_CAP);

break;

case FE\_DAI\_ID\_VOICE\_CAPTURE:

mcdt\_adc\_dma\_disable(MCDT\_CHAN\_VOICE\_CAPTURE);

break;

case FE\_DAI\_ID\_LOOP:

if (is\_playback)

mcdt\_dac\_dma\_disable(MCDT\_CHAN\_LOOP);

else

mcdt\_adc\_dma\_disable(MCDT\_CHAN\_LOOP);

break;

case FE\_DAI\_ID\_FAST\_P:

mcdt\_dac\_dma\_disable(MCDT\_CHAN\_FAST\_PLAY);

break;

case FE\_DAI\_ID\_HIFI\_FAST\_P:

mcdt\_dac\_dma\_disable(MCDT\_CHAN\_HIFI\_FAST\_PLAY);

break;

case FE\_DAI\_ID\_VOIP:

if (is\_playback)

mcdt\_dac\_dma\_disable(MCDT\_CHAN\_VOIP);

else

mcdt\_adc\_dma\_disable(MCDT\_CHAN\_VOIP);

break;

case FE\_DAI\_ID\_A2DP\_PCM:

mcdt\_dac\_dma\_disable(MCDT\_CHAN\_A2DP\_PCM);

break;

case FE\_DAI\_ID\_RECOGNISE\_CAPTURE:

mcdt\_adc\_dma\_disable(MCDT\_CHAN\_RECOGNISE\_CAPTURE);

break;

case FE\_DAI\_ID\_VOICE\_PCM\_P:

mcdt\_dac\_dma\_disable(MCDT\_CHAN\_VOICE\_PCM\_P);

break;

case FE\_DAI\_ID\_HIFI\_P:

mcdt\_dac\_dma\_disable(MCDT\_CHAN\_HIFI\_PLAY);

break;

}

}

static int mcdt\_dma\_config\_init(struct snd\_soc\_dai \*fe\_dai, int stream)

{

int uid;

int is\_playback = stream == SNDRV\_PCM\_STREAM\_PLAYBACK ? 1 : 0;

switch (fe\_dai->id) {

case FE\_DAI\_ID\_FM:

case FE\_DAI\_ID\_OFFLOAD:

case FE\_DAI\_ID\_A2DP\_OFFLOAD:

case FE\_DAI\_ID\_VOICE:

case FE\_DAI\_ID\_FM\_DSP:

case FE\_DAI\_ID\_HFP:

default:

uid = 0;

break;

case FE\_DAI\_ID\_CAPTURE\_DSP:

uid = mcdt\_adc\_dma\_enable(MCDT\_CHAN\_DSP\_CAP,

MCDT\_FULL\_WMK\_DSP\_CAP);

pcm\_dsp\_cap\_mcdt.channels[0] = uid;

break;

case FE\_DAI\_ID\_FAST\_P:

uid = mcdt\_dac\_dma\_enable(MCDT\_CHAN\_FAST\_PLAY,

MCDT\_EMPTY\_WMK\_FAST\_PLAY);

pcm\_fast\_play\_mcdt.channels[0] = uid;

break;

case FE\_DAI\_ID\_HIFI\_FAST\_P:

uid = mcdt\_dac\_dma\_enable(MCDT\_CHAN\_HIFI\_FAST\_PLAY,

MCDT\_EMPTY\_WMK\_HIFI\_FAST\_PLAY);

pcm\_hifi\_fast\_play\_mcdt.channels[0] = uid;

break;

case FE\_DAI\_ID\_VOIP:

if (is\_playback) {

uid = mcdt\_dac\_dma\_enable(MCDT\_CHAN\_VOIP,

MCDT\_EMPTY\_WMK\_VOIP);

pcm\_voip\_play\_mcdt.channels[0] = uid;

} else {

uid = mcdt\_adc\_dma\_enable(MCDT\_CHAN\_VOIP,

MCDT\_FULL\_WMK\_VOIP);

pcm\_voip\_record\_mcdt.channels[0] = uid;

}

break;

case FE\_DAI\_ID\_RECOGNISE\_CAPTURE:

uid = mcdt\_adc\_dma\_enable(MCDT\_CHAN\_RECOGNISE\_CAPTURE,

MCDT\_FULL\_WMK\_RECOGNISE\_CAPTURE);

vbc\_pcm\_recognise\_capture\_mcdt.channels[0] = uid;

break;

case FE\_DAI\_ID\_VOICE\_PCM\_P:

uid = mcdt\_dac\_dma\_enable(MCDT\_CHAN\_VOICE\_PCM\_P,

MCDT\_EMPTY\_WMK\_VOICE\_PCM\_P);

pcm\_voice\_play\_mcdt.channels[0] = uid;

break;

case FE\_DAI\_ID\_HIFI\_P:

uid = mcdt\_dac\_dma\_enable(MCDT\_CHAN\_HIFI\_PLAY,

MCDT\_EMPTY\_WMK\_HIFI\_PLAY);

pcm\_hifi\_play\_mcdt.channels[0] = uid;

break;

}

static void sprd\_dma\_config(struct snd\_pcm\_substream \*substream,

struct snd\_pcm\_hw\_params \*params, struct snd\_soc\_dai \*fe\_dai)

{

u32 rate;

int is\_playback = substream->stream == SNDRV\_PCM\_STREAM\_PLAYBACK ?

1 : 0;

switch (fe\_dai->id) {

case FE\_DAI\_ID\_FM:

case FE\_DAI\_ID\_OFFLOAD:

case FE\_DAI\_ID\_A2DP\_OFFLOAD:

case FE\_DAI\_ID\_VOICE:

case FE\_DAI\_ID\_FM\_DSP:

case FE\_DAI\_ID\_HFP:

default:

pr\_info("%s %s do not use dma\n", \_\_func\_\_,

fe\_dai\_id\_to\_str(fe\_dai->id));

break;

case FE\_DAI\_ID\_FAST\_P:

/\*fast playback\*/

pcm\_fast\_play\_mcdt.name = "VBC PCM Fast P";

pcm\_fast\_play\_mcdt.irq\_type = SPRD\_DMA\_BLK\_INT;

pcm\_fast\_play\_mcdt.desc.datawidth =

DMA\_SLAVE\_BUSWIDTH\_4\_BYTES;

pcm\_fast\_play\_mcdt.desc.fragmens\_len = MCDT\_FAST\_PLAY\_FRAGMENT;

pcm\_fast\_play\_mcdt.use\_mcdt = 1;

pcm\_fast\_play\_mcdt.dev\_paddr[0] =

mcdt\_dac\_dma\_phy\_addr(MCDT\_CHAN\_FAST\_PLAY);

pcm\_fast\_play\_mcdt.used\_dma\_channel\_name[0] = "fast\_p";

break;

case FE\_DAI\_ID\_HIFI\_FAST\_P:

/\*fast playback\*/

pcm\_hifi\_fast\_play\_mcdt.name = "VBC PCM HIFI Fast P";

pcm\_hifi\_fast\_play\_mcdt.irq\_type = SPRD\_DMA\_BLK\_INT;

pcm\_hifi\_fast\_play\_mcdt.desc.datawidth =

DMA\_SLAVE\_BUSWIDTH\_4\_BYTES;

pcm\_hifi\_fast\_play\_mcdt.desc.fragmens\_len = MCDT\_HIFI\_FAST\_PLAY\_FRAGMENT;

pcm\_hifi\_fast\_play\_mcdt.use\_mcdt = 1;

pcm\_hifi\_fast\_play\_mcdt.dev\_paddr[0] =

mcdt\_dac\_dma\_phy\_addr(MCDT\_CHAN\_HIFI\_FAST\_PLAY);

pcm\_hifi\_fast\_play\_mcdt.used\_dma\_channel\_name[0] = "hifi\_fast\_p";

break;

case FE\_DAI\_ID\_VOIP:

if (is\_playback) {

/\*voip play\*/

pcm\_voip\_play\_mcdt.name = "PCM voip play With MCDT";

pcm\_voip\_play\_mcdt.irq\_type = SPRD\_DMA\_BLK\_INT;

pcm\_voip\_play\_mcdt.desc.datawidth =

DMA\_SLAVE\_BUSWIDTH\_4\_BYTES;

pcm\_voip\_play\_mcdt.desc.fragmens\_len =

MCDT\_VOIP\_P\_FRAGMENT;

pcm\_voip\_play\_mcdt.use\_mcdt = 1;

pcm\_voip\_play\_mcdt.dev\_paddr[0] =

mcdt\_dac\_dma\_phy\_addr(MCDT\_CHAN\_VOIP);

pcm\_voip\_play\_mcdt.used\_dma\_channel\_name[0] = "voip\_p";

} else {

/\*voip capture\*/

pcm\_voip\_record\_mcdt.name = "PCM voip record With MCDT";

pcm\_voip\_record\_mcdt.irq\_type = SPRD\_DMA\_BLK\_INT;

pcm\_voip\_record\_mcdt.desc.datawidth = SPRD\_DMA\_BLK\_INT;

pcm\_voip\_record\_mcdt.desc.fragmens\_len =

MCDT\_VOIP\_C\_FRAGMENT;

pcm\_voip\_record\_mcdt.use\_mcdt = 1;

pcm\_voip\_record\_mcdt.dev\_paddr[0] =

mcdt\_adc\_dma\_phy\_addr(MCDT\_CHAN\_VOIP);

pcm\_voip\_record\_mcdt.used\_dma\_channel\_name[0] =

"voip\_c";

pcm\_voip\_record\_mcdt.use\_mcdt = 1;

}

break;

case FE\_DAI\_ID\_RECOGNISE\_CAPTURE:

/\*recognise capture\*/

vbc\_pcm\_recognise\_capture\_mcdt.name =

"VBC PCM recognise C With MCDT";

vbc\_pcm\_recognise\_capture\_mcdt.irq\_type = SPRD\_DMA\_BLK\_INT;

vbc\_pcm\_recognise\_capture\_mcdt.desc.datawidth =

DMA\_SLAVE\_BUSWIDTH\_4\_BYTES;

vbc\_pcm\_recognise\_capture\_mcdt.desc.fragmens\_len =

MCDT\_RECOGNISE\_C\_FRAGMENT;

vbc\_pcm\_recognise\_capture\_mcdt.use\_mcdt = 1;

/\* dma src address \*/

vbc\_pcm\_recognise\_capture\_mcdt.dev\_paddr[0] =

mcdt\_adc\_dma\_phy\_addr(MCDT\_CHAN\_RECOGNISE\_CAPTURE);

vbc\_pcm\_recognise\_capture\_mcdt.used\_dma\_channel\_name[0] =

"recognise\_c";

break;

case FE\_DAI\_ID\_VOICE\_PCM\_P:

/\*voice pcm play\*/

pcm\_voice\_play\_mcdt.name = "PCM voice play With MCDT";

pcm\_voice\_play\_mcdt.irq\_type = SPRD\_DMA\_BLK\_INT;

pcm\_voice\_play\_mcdt.desc.datawidth =

DMA\_SLAVE\_BUSWIDTH\_4\_BYTES;

pcm\_voice\_play\_mcdt.desc.fragmens\_len =

MCDT\_VOICE\_PCM\_P\_FRAGMENT;

pcm\_voice\_play\_mcdt.use\_mcdt = 1;

pcm\_voice\_play\_mcdt.dev\_paddr[0] =

mcdt\_dac\_dma\_phy\_addr(MCDT\_CHAN\_VOICE\_PCM\_P);

pcm\_voice\_play\_mcdt.used\_dma\_channel\_name[0] =

"voice\_pcm\_p";

break;

case FE\_DAI\_ID\_HIFI\_P:

/\*hifi playback\*/

pcm\_hifi\_play\_mcdt.name = "DSP IIS HIFI P";

pcm\_hifi\_play\_mcdt.irq\_type = SPRD\_DMA\_BLK\_INT;

pcm\_hifi\_play\_mcdt.desc.datawidth =

DMA\_SLAVE\_BUSWIDTH\_4\_BYTES;

pcm\_hifi\_play\_mcdt.desc.fragmens\_len = MCDT\_HIFI\_PLAY\_FRAGMENT;

pcm\_hifi\_play\_mcdt.use\_mcdt = 1;

pcm\_hifi\_play\_mcdt.dev\_paddr[0] =

mcdt\_dac\_dma\_phy\_addr(MCDT\_CHAN\_HIFI\_PLAY);

pcm\_hifi\_play\_mcdt.used\_dma\_channel\_name[0] = "hifi\_p";

break;

}

}

struct sprd\_pcm\_dma\_params \*get\_dma\_data\_params(struct snd\_soc\_dai \*fe\_dai,

int stream)

{

int is\_playback = stream == SNDRV\_PCM\_STREAM\_PLAYBACK ? 1 : 0;

struct sprd\_pcm\_dma\_params \*dma\_data;

switch (fe\_dai->id) {

case FE\_DAI\_ID\_OFFLOAD:

case FE\_DAI\_ID\_A2DP\_OFFLOAD:

case FE\_DAI\_ID\_VOICE:

case FE\_DAI\_ID\_FM:

case FE\_DAI\_ID\_FM\_DSP:

case FE\_DAI\_ID\_HFP:

default:

dma\_data = NULL;

break;

case FE\_DAI\_ID\_HIFI\_FAST\_P:

dma\_data = &pcm\_hifi\_fast\_play\_mcdt;

break;

case FE\_DAI\_ID\_VOIP:

dma\_data = is\_playback ? &pcm\_voip\_play\_mcdt :

&pcm\_voip\_record\_mcdt;

break;

case FE\_DAI\_ID\_VOICE\_PCM\_P:

dma\_data = &pcm\_voice\_play\_mcdt;

break;

case FE\_DAI\_ID\_HIFI\_P:

dma\_data = &pcm\_hifi\_play\_mcdt;

break;

}

return dma\_data;

}

static struct snd\_soc\_dai\_driver sprd\_fe\_dais[FE\_DAI\_ID\_MAX] = {

/\* 0: FE\_DAI\_ID\_NORMAL\_AP01 \*/

{

.id = FE\_DAI\_ID\_NORMAL\_AP01,

.name = TO\_STRING(FE\_DAI\_ID\_NORMAL\_AP01),

.probe = fe\_dai\_probe,

.playback = {

.stream\_name = "FE\_DAI\_NORMAL\_AP01\_P",

.aif\_name = "FE\_IF\_NORMAL\_AP01\_P",

.rates = SNDRV\_PCM\_RATE\_CONTINUOUS,

.formats = (SNDRV\_PCM\_FMTBIT\_S16\_LE |

SNDRV\_PCM\_FMTBIT\_S24\_LE),

.channels\_min = 1,

.channels\_max = 2,

.rate\_min = 8000,

.rate\_max = 192000,

},

.capture = {

.stream\_name = "FE\_DAI\_NORMAL\_AP01\_C",

.aif\_name = "FE\_IF\_NORMAL\_AP01\_C",

.rates = SNDRV\_PCM\_RATE\_CONTINUOUS,

.formats = (SNDRV\_PCM\_FMTBIT\_S16\_LE |

SNDRV\_PCM\_FMTBIT\_S24\_LE),

.channels\_min = 1,

.channels\_max = 2,

.rate\_min = 8000,

.rate\_max = 192000,

},

.ops = &sprd\_fe\_dai\_ops,

},

/\*17 FE\_DAI\_ID\_BTSCO\_CAP\_AP \*/

{

.id = FE\_DAI\_ID\_BTSCO\_CAP\_AP,

.name = TO\_STRING(FE\_DAI\_ID\_BTSCO\_CAP\_AP),

.probe = fe\_dai\_probe,

.capture = {

.stream\_name = "FE\_DAI\_BTCAP\_AP\_C",

.aif\_name = "FE\_IF\_BTCAP\_AP\_C",

.rates = SNDRV\_PCM\_RATE\_CONTINUOUS,

.formats = (SNDRV\_PCM\_FMTBIT\_S16\_LE |

SNDRV\_PCM\_FMTBIT\_S24\_LE),

.channels\_min = 1,

.channels\_max = 2,

.rate\_min = 8000,

.rate\_max = 192000,

},

.ops = &sprd\_fe\_dai\_ops,

},

/\*18: FE\_DAI\_ID\_VOICE\_PCM\_P\*/

{

.id = FE\_DAI\_ID\_VOICE\_PCM\_P,

.name = TO\_STRING(FE\_DAI\_ID\_VOICE\_PCM\_P),

.probe = fe\_dai\_probe,

.playback = {

.stream\_name = "FE\_DAI\_VOICE\_PCM\_P",

.aif\_name = "FE\_IF\_VOICE\_PCM\_P",

.rates = SNDRV\_PCM\_RATE\_CONTINUOUS,

.formats = (SNDRV\_PCM\_FMTBIT\_S16\_LE |

SNDRV\_PCM\_FMTBIT\_S24\_LE),

.channels\_min = 1,

.channels\_max = 2,

.rate\_min = 8000,

.rate\_max = 192000,

},

.ops = &sprd\_fe\_dai\_ops,

},

/\* 19: FE\_DAI\_ID\_CODEC\_TEST \*/

{

.id = FE\_DAI\_ID\_CODEC\_TEST,

.name = TO\_STRING(FE\_DAI\_ID\_CODEC\_TEST),

.playback = {

.stream\_name = "FE\_DAI\_CODEC\_TEST\_P",

.rates = SNDRV\_PCM\_RATE\_CONTINUOUS,

.formats = (SNDRV\_PCM\_FMTBIT\_S16\_LE |

SNDRV\_PCM\_FMTBIT\_S24\_LE),

.channels\_min = 1,

.channels\_max = 2,

.rate\_min = 8000,

.rate\_max = 192000,

},

.capture = {

.stream\_name = "FE\_DAI\_CODEC\_TEST\_C",

.rates = SNDRV\_PCM\_RATE\_CONTINUOUS,

.formats = (SNDRV\_PCM\_FMTBIT\_S16\_LE |

SNDRV\_PCM\_FMTBIT\_S24\_LE),

.channels\_min = 1,

.channels\_max = 2,

.rate\_min = 8000,

.rate\_max = 192000,

},

},

/\* 20: FE\_DAI\_ID\_HFP \*/

{

.id = FE\_DAI\_ID\_HFP,

.name = TO\_STRING(FE\_DAI\_ID\_HFP),

.probe = fe\_dai\_probe,

.playback = {

.stream\_name = "FE\_DAI\_HFP\_P",

.aif\_name = "FE\_IF\_HFP\_P",

.rates = SNDRV\_PCM\_RATE\_CONTINUOUS,

.formats = (SNDRV\_PCM\_FMTBIT\_S16\_LE |

SNDRV\_PCM\_FMTBIT\_S24\_LE),

.channels\_min = 1,

.channels\_max = 2,

.rate\_min = 8000,

.rate\_max = 192000,

},

.capture = {

.stream\_name = "FE\_DAI\_HFP\_C",

.aif\_name = "FE\_IF\_HFP\_C",

.rates = SNDRV\_PCM\_RATE\_CONTINUOUS,

.formats = (SNDRV\_PCM\_FMTBIT\_S16\_LE |

SNDRV\_PCM\_FMTBIT\_S24\_LE),

.channels\_min = 1,

.channels\_max = 2,

.rate\_min = 8000,

.rate\_max = 192000,

},

.ops = &sprd\_fe\_dai\_ops,

},

/\* 22: FE\_DAI\_ID\_HIFI\_P \*/

{

.id = FE\_DAI\_ID\_HIFI\_P,

.name = TO\_STRING(FE\_DAI\_ID\_HIFI\_P),

.probe = fe\_dai\_probe,

.playback = {

.stream\_name = "FE\_DAI\_HIFI\_P",

.aif\_name = "FE\_IF\_HIFI\_P",

.rates = SNDRV\_PCM\_RATE\_CONTINUOUS,

.formats = (SNDRV\_PCM\_FMTBIT\_S16\_LE |

SNDRV\_PCM\_FMTBIT\_S24\_LE),

.channels\_min = 1,

.channels\_max = 2,

.rate\_min = 8000,

.rate\_max = 192000,

},

.ops = &sprd\_fe\_dai\_ops,

},

/\* 23: FE\_DAI\_ID\_HIFI\_FAST\_P \*/

{

.id = FE\_DAI\_ID\_HIFI\_FAST\_P,

.name = TO\_STRING(FE\_DAI\_ID\_HIFI\_FAST\_P),

.probe = fe\_dai\_probe,

.playback = {

.stream\_name = "FE\_DAI\_HIFI\_FAST\_P",

.aif\_name = "FE\_IF\_HIFI\_FAST\_P",

.rates = SNDRV\_PCM\_RATE\_CONTINUOUS,

.formats = (SNDRV\_PCM\_FMTBIT\_S16\_LE |

SNDRV\_PCM\_FMTBIT\_S24\_LE),

.channels\_min = 1,

.channels\_max = 2,

.rate\_min = 8000,

.rate\_max = 192000,

},

.ops = &sprd\_fe\_dai\_ops,

},

};

sound\soc\sprd\dai\vbc\v4\sprd-fe-dai.h

#ifndef \_\_SPRD\_FE\_DAI\_H

#define \_\_SPRD\_FE\_DAI\_H

/\* dma hw request \*/

#define DMA\_REQ\_DA0\_DEV\_ID (1)

#define DMA\_REQ\_DA1\_DEV\_ID (2)

#define DMA\_REQ\_DA2\_DEV\_ID (17)

#define DMA\_REQ\_DA3\_DEV\_ID (18)

#define DMA\_REQ\_AD0\_DEV\_ID (3)

#define DMA\_REQ\_AD1\_DEV\_ID (4)

#define DMA\_REQ\_AD2\_DEV\_ID (15)

#define DMA\_REQ\_AD3\_DEV\_ID (16)

#define VBC\_AUDPLY01\_FRAGMENT (80)

#define VBC\_AUDPLY23\_FRAGMENT (80)

#define VBC\_AUDRCD01\_FRAGMENT (80)

#define VBC\_AUDRCD23\_FRAGMENT (80)

/\* mcdt channel 5 6 7 has been used by modem \*/

/\* FE\_DAI\_ID\_VOIP \*/

#define MCDT\_CHAN\_VOIP MCDT\_CHAN1

#define MCDT\_FULL\_WMK\_VOIP (160)

#define MCDT\_EMPTY\_WMK\_VOIP (320)

#define MCDT\_VOIP\_P\_FRAGMENT (160)

#define MCDT\_VOIP\_C\_FRAGMENT (160)

/\* FE\_DAI\_ID\_VOICE\_CAPTURE \*/

#define MCDT\_CHAN\_VOICE\_CAPTURE MCDT\_CHAN2

#define MCDT\_FULL\_WMK\_VOICE\_CAPTURE 160

#define MCDT\_VOICE\_C\_FRAGMENT 160

/\* FE\_DAI\_ID\_LOOP: reuse MCDT\_CHAN3 playback path \*/

#define MCDT\_CHAN\_LOOP MCDT\_CHAN3

#define MCDT\_LOOP\_P\_FRAGMENT 160

#define MCDT\_LOOP\_C\_FRAGMENT 160

#define MCDT\_FULL\_WMK\_LOOP 160

#define MCDT\_EMPTY\_WMK\_LOOP 160

/\* FE\_DAI\_ID\_A2DP\_PCM: reuse MCDT\_CHAN3 playback path \*/

#define MCDT\_CHAN\_A2DP\_PCM MCDT\_CHAN\_LOOP

#define MCDT\_EMPTY\_WMK\_A2DP\_PCM 320

#define MCDT\_A2DP\_PCM\_FRAGMENT 160

/\* FE\_DAI\_ID\_FAST\_P: use MCDT\_CHAN4 playback path \*/

#define MCDT\_CHAN\_FAST\_PLAY MCDT\_CHAN4

#define MCDT\_EMPTY\_WMK\_FAST\_PLAY 320

#define MCDT\_FAST\_PLAY\_FRAGMENT 160

/\* FE\_DAI\_ID\_HIFI\_FAST\_P: use MCDT\_CHAN10 playback path \*/

#define MCDT\_CHAN\_HIFI\_FAST\_PLAY MCDT\_CHAN10

#define MCDT\_EMPTY\_WMK\_HIFI\_FAST\_PLAY 320

#define MCDT\_HIFI\_FAST\_PLAY\_FRAGMENT 160

/\* FE\_DAI\_ID\_CAPTURE\_DSP: use MCDT\_CHAN4 capture path \*/

#define MCDT\_CHAN\_DSP\_CAP MCDT\_CHAN4

#define MCDT\_FULL\_WMK\_DSP\_CAP 320

#define MCDT\_DSPCAP\_FRAGMENT 80

/\* FE\_DAI\_ID\_FM\_CAP\_DSP: same as FE\_DAI\_ID\_CAPTURE\_DSP \*/

#define MCDT\_CHAN\_DSP\_FM\_CAP MCDT\_CHAN4

#define MCDT\_FULL\_WMK\_DSP\_FM\_CAP 320

#define MCDT\_DSPFMCAP\_FRAGMENT 320

/\* FE\_DAI\_ID\_BTSCO\_CAP\_DSP: same as FE\_DAI\_ID\_CAPTURE\_DSP \*/

#define MCDT\_CHAN\_DSP\_BTSCO\_CAP MCDT\_CHAN4

#define MCDT\_FULL\_WMK\_DSP\_BTSCO\_CAP 320

#define MCDT\_DSPBTSCOCAP\_FRAGMENT 320

/\* FE\_DAI\_ID\_RECOGNISE\_CAPTURE \*/

#define MCDT\_CHAN\_RECOGNISE\_CAPTURE MCDT\_CHAN2

#define MCDT\_FULL\_WMK\_RECOGNISE\_CAPTURE 320

#define MCDT\_RECOGNISE\_C\_FRAGMENT 320

/\* FE\_DAI\_ID\_VOICE\_PCM\_P \*/

#define MCDT\_CHAN\_VOICE\_PCM\_P MCDT\_CHAN1

#define MCDT\_EMPTY\_WMK\_VOICE\_PCM\_P (320)

#define MCDT\_VOICE\_PCM\_P\_FRAGMENT (160)

/\* FE\_DAI\_ID\_HIFI\_P: use MCDT\_CHAN4 playback path \*/

#define MCDT\_CHAN\_HIFI\_PLAY MCDT\_CHAN2

#define MCDT\_EMPTY\_WMK\_HIFI\_PLAY 320

#define MCDT\_HIFI\_PLAY\_FRAGMENT 160

#endif

sound\soc\sprd\dai\vbc\v4\vbc-codec.c

static const char \* const dsp\_voice\_capture\_type\_txt[] = {

/\* type 0, type 1, type 2 \*/

"VOICE\_CAPTURE\_DOWNLINK", "VOICE\_CAPTURE\_UPLINK",

"VOICE\_CAPTURE\_UPLINK\_DOWNLINK",

};

static const char \* const dsp\_voice\_pcm\_play\_mode\_txt[] = {

/\* type 0, type 1 \*/

"VOICE\_PCM\_PLAY\_UPLINK\_MIX", "VOICE\_PCM\_PLAY\_UPLINK\_ONLY",

};

static const struct soc\_enum dsp\_voice\_capture\_enum =

SPRD\_VBC\_ENUM(SND\_SOC\_NOPM, 3, dsp\_voice\_capture\_type\_txt);

static const struct soc\_enum dsp\_voice\_pcm\_play\_enum =

SPRD\_VBC\_ENUM(SND\_SOC\_NOPM, 2, dsp\_voice\_pcm\_play\_mode\_txt);

/\* FM\_MUTE \*/

static const struct soc\_enum vbc\_fm\_mute\_enum =

SPRD\_VBC\_ENUM(SND\_SOC\_NOPM, 2, mute\_unmute\_txt);

static int vbc\_fm\_mute\_get(struct snd\_kcontrol \*kcontrol,

struct snd\_ctl\_elem\_value \*ucontrol)

{

struct snd\_soc\_codec \*codec = snd\_soc\_kcontrol\_codec(kcontrol);

struct vbc\_codec\_priv \*vbc\_codec = snd\_soc\_codec\_get\_drvdata(codec);

ucontrol->value.integer.value[0] = vbc\_codec->fm\_mute.mute;

return 0;

}

static int vbc\_fm\_mute\_put(struct snd\_kcontrol \*kcontrol,

struct snd\_ctl\_elem\_value \*ucontrol)

{

u16 value;

struct soc\_enum \*texts = (struct soc\_enum \*)kcontrol->private\_value;

struct snd\_soc\_codec \*codec = snd\_soc\_kcontrol\_codec(kcontrol);

struct vbc\_codec\_priv \*vbc\_codec = snd\_soc\_codec\_get\_drvdata(codec);

struct soc\_enum \*e = (struct soc\_enum \*)kcontrol->private\_value;

u32 id = e->reg;

if (ucontrol->value.integer.value[0] >= texts->items) {

pr\_err("ERR: %s,index outof bounds error\n", \_\_func\_\_);

return -EINVAL;

}

value = ucontrol->value.enumerated.item[0];

sp\_asoc\_pr\_dbg("%s, fm\_mute %s, mute step %d\n", \_\_func\_\_,

texts->texts[value],

vbc\_codec->fm\_mute\_step.step);

vbc\_codec->fm\_mute.id = id;

vbc\_codec->fm\_mute.mute = value;

fm\_mute\_set(id, value, vbc\_codec->fm\_mute\_step.step);

return true;

}

static int vbc\_fm\_mdg\_stp\_get(struct snd\_kcontrol \*kcontrol,

struct snd\_ctl\_elem\_value \*ucontrol)

{

struct snd\_soc\_codec \*codec = snd\_soc\_kcontrol\_codec(kcontrol);

struct vbc\_codec\_priv \*vbc\_codec = snd\_soc\_codec\_get\_drvdata(codec);

ucontrol->value.integer.value[0] = vbc\_codec->fm\_mute\_step.step;

return 0;

}

static int vbc\_fm\_mdg\_stp\_put(struct snd\_kcontrol \*kcontrol,

struct snd\_ctl\_elem\_value \*ucontrol)

{

int value;

struct snd\_soc\_codec \*codec = snd\_soc\_kcontrol\_codec(kcontrol);

struct vbc\_codec\_priv \*vbc\_codec = snd\_soc\_codec\_get\_drvdata(codec);

value = ucontrol->value.integer.value[0];

sp\_asoc\_pr\_dbg("%s vbc\_fm\_mdg\_stp = %d\n",

\_\_func\_\_, value);

vbc\_codec->fm\_mute\_step.step = value;

return value;

}

static const char \* const sys\_iis\_sel\_txt[] = {

"vbc\_iis0", "vbc\_iis1", "vbc\_iis2", "vbc\_iis3", "vbc\_iism0", "ap\_iis0",

"audcp\_iis0", "audcp\_iis1"

};

static int vbc\_voice\_capture\_type\_get(struct snd\_kcontrol \*kcontrol,

struct snd\_ctl\_elem\_value \*ucontrol)

{

struct snd\_soc\_codec \*codec = snd\_soc\_kcontrol\_codec(kcontrol);

struct vbc\_codec\_priv \*vbc\_codec = snd\_soc\_codec\_get\_drvdata(codec);

ucontrol->value.integer.value[0] = vbc\_codec->voice\_capture\_type;

return 0;

}

static int vbc\_voice\_capture\_type\_put(struct snd\_kcontrol \*kcontrol,

struct snd\_ctl\_elem\_value \*ucontrol)

{

int value;

struct soc\_enum \*texts = (struct soc\_enum \*)kcontrol->private\_value;

struct snd\_soc\_codec \*codec = snd\_soc\_kcontrol\_codec(kcontrol);

struct vbc\_codec\_priv \*vbc\_codec = snd\_soc\_codec\_get\_drvdata(codec);

if (ucontrol->value.integer.value[0] >= texts->items) {

pr\_err("ERR: %s,index outof bounds error\n", \_\_func\_\_);

return -EINVAL;

}

value = ucontrol->value.enumerated.item[0];

sp\_asoc\_pr\_dbg("%s, texts->texts[%d] =%s\n",

\_\_func\_\_, value, texts->texts[value]);

vbc\_codec->voice\_capture\_type = value;

return value;

}

static int vbc\_voice\_pcm\_play\_mode\_get(struct snd\_kcontrol \*kcontrol,

struct snd\_ctl\_elem\_value \*ucontrol)

{

struct snd\_soc\_codec \*codec = snd\_soc\_kcontrol\_codec(kcontrol);

struct vbc\_codec\_priv \*vbc\_codec = snd\_soc\_codec\_get\_drvdata(codec);

ucontrol->value.integer.value[0] = vbc\_codec->voice\_pcm\_play\_mode;

return 0;

}

static int vbc\_voice\_pcm\_play\_mode\_put(struct snd\_kcontrol \*kcontrol,

struct snd\_ctl\_elem\_value \*ucontrol)

{

int value;

struct soc\_enum \*texts = (struct soc\_enum \*)kcontrol->private\_value;

struct snd\_soc\_codec \*codec = snd\_soc\_kcontrol\_codec(kcontrol);

struct vbc\_codec\_priv \*vbc\_codec = snd\_soc\_codec\_get\_drvdata(codec);

if (ucontrol->value.integer.value[0] >= texts->items) {

pr\_err("ERR: %s,index outof bounds error\n", \_\_func\_\_);

return -EINVAL;

}

value = ucontrol->value.enumerated.item[0];

sp\_asoc\_pr\_dbg("%s, texts->texts[%d] =%s\n",

\_\_func\_\_, value, texts->texts[value]);

vbc\_codec->voice\_pcm\_play\_mode = value;

return value;

}

static const struct snd\_kcontrol\_new vbc\_codec\_snd\_controls[] = {

/\* MDG \*/

SOC\_DOUBLE\_R\_EXT\_TLV("VBC DAC0 DSP MDG Set",

0, 1, VBC\_MDG\_DAC0\_DSP, MDG\_STP\_MAX\_VAL, 0,

vbc\_mdg\_get,

vbc\_mdg\_put, mdg\_tlv),

SOC\_SINGLE\_BOOL\_EXT("agdsp\_access\_a2dp\_en", 0,

vbc\_get\_agdsp\_access, vbc\_put\_agdsp\_a2dp\_access),

/\*FM MUTE\*/

SOC\_SINGLE\_EXT("VBC FM\_MUTE\_SMOOTHDG STEP", SND\_SOC\_NOPM, 0,

MAX\_12\_BIT, 0,

vbc\_fm\_mdg\_stp\_get, vbc\_fm\_mdg\_stp\_put),

SOC\_ENUM\_EXT("VBC\_FM\_UNMUTE\_SMOOTH", vbc\_fm\_mute\_enum,

vbc\_fm\_mute\_get, vbc\_fm\_mute\_put),

/\* VBC VOLUME \*/

SOC\_SINGLE\_EXT("VBC\_VOLUME", SND\_SOC\_NOPM, 0,

MAX\_32\_BIT, 0,

vbc\_volume\_get, vbc\_volume\_put),

sound\soc\sprd\dai\vbc\v4\vbc-dai.c

static const char \*dai\_id\_to\_str(int dai\_id)

{

const char \* const dai\_id\_str[BE\_DAI\_ID\_MAX] = {

[BE\_DAI\_ID\_NORMAL\_AP01\_CODEC] =

TO\_STRING(BE\_DAI\_ID\_NORMAL\_AP01\_CODEC),

[BE\_DAI\_ID\_NORMAL\_AP23\_CODEC] =

TO\_STRING(BE\_DAI\_ID\_NORMAL\_AP23\_CODEC),

[BE\_DAI\_ID\_CAPTURE\_DSP\_CODEC] =

TO\_STRING(BE\_DAI\_ID\_CAPTURE\_DSP\_CODEC),

[BE\_DAI\_ID\_NORMAL\_AP01\_P\_BTSCO] =

TO\_STRING(BE\_DAI\_ID\_NORMAL\_AP01\_P\_BTSCO),

[BE\_DAI\_ID\_NORMAL\_AP01\_P\_HIFI] =

TO\_STRING(BE\_DAI\_ID\_NORMAL\_AP01\_P\_HIFI),

[BE\_DAI\_ID\_NORMAL\_AP23\_HIFI] =

TO\_STRING(BE\_DAI\_ID\_NORMAL\_AP23\_HIFI),

[BE\_DAI\_ID\_FAST\_P\_HIFI] = TO\_STRING(BE\_DAI\_ID\_FAST\_P\_HIFI),

[BE\_DAI\_ID\_OFFLOAD\_HIFI] = TO\_STRING(BE\_DAI\_ID\_OFFLOAD\_HIFI),

[BE\_DAI\_ID\_VOICE\_HIFI] = TO\_STRING(BE\_DAI\_ID\_VOICE\_HIFI),

[BE\_DAI\_ID\_VOIP\_HIFI] = TO\_STRING(BE\_DAI\_ID\_VOIP\_HIFI),

[BE\_DAI\_ID\_FM\_HIFI] = TO\_STRING(BE\_DAI\_ID\_FM\_HIFI),

[BE\_DAI\_ID\_LOOP\_HIFI] = TO\_STRING(BE\_DAI\_ID\_LOOP\_HIFI),

[BE\_DAI\_ID\_FM\_DSP\_HIFI] = TO\_STRING(BE\_DAI\_ID\_FM\_DSP\_HIFI),

[BE\_DAI\_ID\_HFP] = TO\_STRING(BE\_DAI\_ID\_HFP),

[BE\_DAI\_ID\_RECOGNISE\_CAPTURE] =

TO\_STRING(BE\_DAI\_ID\_RECOGNISE\_CAPTURE),

[BE\_DAI\_ID\_VOICE\_PCM\_P] = TO\_STRING(BE\_DAI\_ID\_VOICE\_PCM\_P),

[BE\_DAI\_ID\_NORMAL\_AP01\_P\_SMTPA] =

TO\_STRING(BE\_DAI\_ID\_NORMAL\_AP01\_P\_SMTPA),

[BE\_DAI\_ID\_NORMAL\_AP23\_SMTPA] =

TO\_STRING(BE\_DAI\_ID\_NORMAL\_AP23\_SMTPA),

[BE\_DAI\_ID\_FAST\_P\_SMTPA] = TO\_STRING(BE\_DAI\_ID\_FAST\_P\_SMTPA),

[BE\_DAI\_ID\_OFFLOAD\_SMTPA] = TO\_STRING(BE\_DAI\_ID\_OFFLOAD\_SMTPA),

[BE\_DAI\_ID\_VOICE\_SMTPA] = TO\_STRING(BE\_DAI\_ID\_VOICE\_SMTPA),

[BE\_DAI\_ID\_VOIP\_SMTPA] = TO\_STRING(BE\_DAI\_ID\_VOIP\_SMTPA),

[BE\_DAI\_ID\_FM\_SMTPA] = TO\_STRING(BE\_DAI\_ID\_FM\_SMTPA),

[BE\_DAI\_ID\_LOOP\_SMTPA] = TO\_STRING(BE\_DAI\_ID\_LOOP\_SMTPA),

[BE\_DAI\_ID\_FM\_DSP\_SMTPA] = TO\_STRING(BE\_DAI\_ID\_FM\_DSP\_SMTPA),

[BE\_DAI\_ID\_HIFI\_P] = TO\_STRING(BE\_DAI\_ID\_HIFI\_P),

[BE\_DAI\_ID\_HIFI\_FAST\_P] = TO\_STRING(BE\_DAI\_ID\_HIFI\_FAST\_P),

};

static const char \*scene\_id\_to\_str(int scene\_id)

{

const char \*scene\_id\_str[VBC\_DAI\_ID\_MAX] = {

[VBC\_DAI\_ID\_NORMAL\_AP01] = TO\_STRING(VBC\_DAI\_ID\_NORMAL\_AP01),

[VBC\_DAI\_ID\_NORMAL\_AP23] = TO\_STRING(VBC\_DAI\_ID\_NORMAL\_AP23),

[VBC\_DAI\_ID\_CAPTURE\_DSP] = TO\_STRING(VBC\_DAI\_ID\_CAPTURE\_DSP),

[VBC\_DAI\_ID\_FAST\_P] = TO\_STRING(VBC\_DAI\_ID\_FAST\_P),

[VBC\_DAI\_ID\_OFFLOAD] = TO\_STRING(VBC\_DAI\_ID\_OFFLOAD),

[VBC\_DAI\_ID\_VOICE] = TO\_STRING(VBC\_DAI\_ID\_VOICE),

[VBC\_DAI\_ID\_VOIP] = TO\_STRING(VBC\_DAI\_ID\_VOIP),

[VBC\_DAI\_ID\_FM] = TO\_STRING(VBC\_DAI\_ID\_FM),

[VBC\_DAI\_ID\_FM\_CAPTURE\_AP] =

TO\_STRING(VBC\_DAI\_ID\_FM\_CAPTURE\_AP),

[VBC\_DAI\_ID\_VOICE\_CAPTURE] =

TO\_STRING(VBC\_DAI\_ID\_VOICE\_CAPTURE),

[VBC\_DAI\_ID\_LOOP] = TO\_STRING(VBC\_DAI\_ID\_LOOP),

[VBC\_DAI\_ID\_PCM\_A2DP] = TO\_STRING(VBC\_DAI\_ID\_PCM\_A2DP),

[VBC\_DAI\_ID\_OFFLOAD\_A2DP] = TO\_STRING(VBC\_DAI\_ID\_OFFLOAD\_A2DP),

[VBC\_DAI\_ID\_BT\_CAPTURE\_AP] =

TO\_STRING(VBC\_DAI\_ID\_BT\_CAPTURE\_AP),

[VBC\_DAI\_ID\_FM\_CAPTURE\_DSP] =

TO\_STRING(VBC\_DAI\_ID\_FM\_CAPTURE\_DSP),

[VBC\_DAI\_ID\_BT\_SCO\_CAPTURE\_DSP] =

TO\_STRING(VBC\_DAI\_ID\_BT\_SCO\_CAPTURE\_DSP),

[VBC\_DAI\_ID\_FM\_DSP] = TO\_STRING(VBC\_DAI\_ID\_FM\_DSP),

[VBC\_DAI\_ID\_HFP] = TO\_STRING(VBC\_DAI\_ID\_HFP),

[VBC\_DAI\_ID\_RECOGNISE\_CAPTURE] =

TO\_STRING(VBC\_DAI\_ID\_RECOGNISE\_CAPTURE),

[VBC\_DAI\_ID\_VOICE\_PCM\_P] = TO\_STRING(VBC\_DAI\_ID\_VOICE\_PCM\_P),

[AUDCP\_DAI\_ID\_HIFI] = TO\_STRING(AUDCP\_DAI\_ID\_HIFI),

[AUDCP\_DAI\_ID\_FAST] = TO\_STRING(AUDCP\_DAI\_ID\_FAST),

};

static int check\_be\_dai\_id(int be\_dai\_id)

{

int scene\_id;

switch (be\_dai\_id) {

case BE\_DAI\_ID\_NORMAL\_AP01\_CODEC:

case BE\_DAI\_ID\_NORMAL\_AP01\_USB:

case BE\_DAI\_ID\_NORMAL\_AP01\_P\_BTSCO:

case BE\_DAI\_ID\_NORMAL\_AP01\_P\_HIFI:

case BE\_DAI\_ID\_NORMAL\_AP01\_P\_SMTPA:

case BE\_DAI\_ID\_DUMP:

scene\_id = VBC\_DAI\_ID\_NORMAL\_AP01;

break;

case BE\_DAI\_ID\_NORMAL\_AP23\_USB:

case BE\_DAI\_ID\_NORMAL\_AP23\_HIFI:

case BE\_DAI\_ID\_NORMAL\_AP23\_SMTPA:

scene\_id = VBC\_DAI\_ID\_NORMAL\_AP23;

break;

case BE\_DAI\_ID\_OFFLOAD\_USB:

case BE\_DAI\_ID\_OFFLOAD\_HIFI:

case BE\_DAI\_ID\_OFFLOAD\_SMTPA:

scene\_id = VBC\_DAI\_ID\_OFFLOAD;

break;

case BE\_DAI\_ID\_VOICE\_HIFI:

case BE\_DAI\_ID\_VOICE\_SMTPA:

scene\_id = VBC\_DAI\_ID\_VOICE;

break;

case BE\_DAI\_ID\_VOIP\_HIFI:

case BE\_DAI\_ID\_VOIP\_SMTPA:

scene\_id = VBC\_DAI\_ID\_VOIP;

break;

case BE\_DAI\_ID\_FM\_HIFI:

case BE\_DAI\_ID\_FM\_SMTPA:

scene\_id = VBC\_DAI\_ID\_FM;

break;

case BE\_DAI\_ID\_LOOP\_HIFI:

case BE\_DAI\_ID\_LOOP\_SMTPA:

scene\_id = VBC\_DAI\_ID\_LOOP;

break;

case BE\_DAI\_ID\_FM\_DSP\_USB:

case BE\_DAI\_ID\_FM\_DSP\_HIFI:

case BE\_DAI\_ID\_FM\_DSP\_SMTPA:

scene\_id = VBC\_DAI\_ID\_FM\_DSP;

break;

case BE\_DAI\_ID\_HFP:

scene\_id = VBC\_DAI\_ID\_HFP;

break;

case BE\_DAI\_ID\_RECOGNISE\_CAPTURE:

scene\_id = VBC\_DAI\_ID\_RECOGNISE\_CAPTURE;

break;

case BE\_DAI\_ID\_VOICE\_PCM\_P:

scene\_id = VBC\_DAI\_ID\_VOICE\_PCM\_P;

break;

case BE\_DAI\_ID\_HIFI\_P:

scene\_id = AUDCP\_DAI\_ID\_HIFI;

break;

case BE\_DAI\_ID\_HIFI\_FAST\_P:

scene\_id = AUDCP\_DAI\_ID\_FAST;

break;

}

static int get\_startup\_scene\_dac\_id(int scene\_id)

{

int dac\_id;

switch (scene\_id) {

case VBC\_DAI\_ID\_NORMAL\_AP01:

dac\_id = VBC\_DA0;

break;

case VBC\_DAI\_ID\_NORMAL\_AP23:

dac\_id = VBC\_DA0;

break;

case VBC\_DAI\_ID\_FM\_DSP:

dac\_id = VBC\_DA0;

break;

case VBC\_DAI\_ID\_HFP:

dac\_id = VBC\_DA1;

break;

case VBC\_DAI\_ID\_VOICE\_PCM\_P:

/\* not used \*/

dac\_id = 0;

break;

default:

pr\_err("invalid scene\_id = %d\n", scene\_id);

dac\_id = 0;

break;

}

static int get\_startup\_scene\_adc\_id(int scene\_id)

{

int adc\_id;

switch (scene\_id) {

case VBC\_DAI\_ID\_NORMAL\_AP01:

adc\_id = VBC\_AD0;

break;

case VBC\_DAI\_ID\_HFP:

adc\_id = VBC\_AD2;

break;

case VBC\_DAI\_ID\_RECOGNISE\_CAPTURE:

adc\_id = VBC\_AD0;

break;

case VBC\_DAI\_ID\_VOICE\_PCM\_P:

/\* not used \*/

adc\_id = 0;

break;

default:

pr\_err("invalid scene\_id = %d\n", scene\_id);

adc\_id = VBC\_AD0;

break;

}

static void fill\_hifi\_shutdown\_data(int scene\_id, int stream,

struct snd\_pcm\_hifi\_stream \*hifi\_shutdown\_info)

{

hifi\_shutdown\_info->id = scene\_id;

hifi\_shutdown\_info->stream = stream;

hifi\_shutdown\_info->enable = 0;

pr\_info("%s enable: %d, scene\_id: %d, stream: %d", \_\_func\_\_,

hifi\_shutdown\_info->enable, hifi\_shutdown\_info->id,

hifi\_shutdown\_info->stream);

}

void fill\_hifi\_dsp\_hw\_data(int scene\_id, int stream, int chan\_cnt, int rate, int fmt,

struct sprd\_vbc\_stream\_hw\_paras \*hifi\_data)

{

hifi\_data->stream\_info.id= scene\_id;

hifi\_data->stream\_info.stream = stream;

hifi\_data->hw\_params\_info.channels = chan\_cnt;

hifi\_data->hw\_params\_info.format = fmt;

hifi\_data->hw\_params\_info.rate = rate\_to\_src\_mode(rate);

pr\_info("%s id %d, stream %d, channel %d, fmt %d, rate\_src\_mode %d",

hifi\_data->stream\_info.id, hifi\_data->stream\_info.stream,

hifi\_data->hw\_params\_info.channels,

hifi\_data->hw\_params\_info.format,

hifi\_data->hw\_params\_info.format,

hifi\_data->hw\_params\_info.rate);

}

static void fill\_hifi\_startup\_data(int scene\_id, int stream,

struct snd\_pcm\_hifi\_stream \*hifi\_startup\_info)

{

hifi\_startup\_info->id = scene\_id;

hifi\_startup\_info->stream = stream;

hifi\_startup\_info->enable = 1;

pr\_info("%s startup enable: %d, scene\_id %d, stream %d",

\_\_func\_\_, hifi\_startup\_info->enable,

hifi\_startup\_info->id, hifi\_startup\_info->stream);

}

static int hifi\_dsp\_trigger(int scene\_id, int stream, int up\_down)

{

int ret;

ret = hifi\_func\_trigger(scene\_id, stream, up\_down);

if (ret < 0) {

pr\_err("vbc\_dsp\_func\_trigger return error\n");

return ret;

}

return 0;

}

static void hifi\_hw\_params(int scene\_id, int stream, int chan\_cnt, u32 rate, int data\_fmt)

{

struct sprd\_vbc\_stream\_hw\_paras hifi\_data;

int ret;

memset(&hifi\_data, 0, sizeof(struct sprd\_vbc\_stream\_hw\_paras));

fill\_hifi\_dsp\_hw\_data(scene\_id, stream, chan\_cnt, rate, data\_fmt,

&hifi\_data);

ret = hifi\_dsp\_func\_hwparam(scene\_id, stream, &hifi\_data);

if (ret < 0) {

pr\_err("HIFI\_func\_hwparam return error, scene\_id: %d\n", scene\_id);

return;

}

}

static void hifi\_shutdown(int scene\_id, int stream)

{

int ret;

struct snd\_pcm\_hifi\_stream hifi\_shutdown\_info;

ret = agdsp\_access\_enable();

if (ret) {

pr\_err("%s, agdsp\_access\_enable failed!\n", \_\_func\_\_);

return;

}

memset(&hifi\_shutdown\_info, 0,

sizeof(struct snd\_pcm\_hifi\_stream));

fill\_hifi\_shutdown\_data(scene\_id, stream, &hifi\_shutdown\_info);

ret = hifi\_func\_shutdown(scene\_id, stream, &hifi\_shutdown\_info);

if (ret < 0) {

agdsp\_access\_disable();

return;

}

agdsp\_access\_disable();

}

static int hifi\_startup(int scene\_id, int stream)

{

int ret = 0;

struct snd\_pcm\_hifi\_stream hifi\_startup\_info;

ret = agdsp\_access\_enable();

if (ret) {

pr\_err("%s:agdsp\_access\_enable:error:%d", \_\_func\_\_, ret);

return ret;

}

memset(&hifi\_startup\_info, 0, sizeof(struct snd\_pcm\_hifi\_stream));

fill\_hifi\_startup\_data(scene\_id, stream, &hifi\_startup\_info);

ret = hifi\_func\_startup(scene\_id, stream, &hifi\_startup\_info);

if (ret < 0) {

pr\_err("vbc\_dsp\_func\_startup return error");

agdsp\_access\_disable();

return ret;

}

agdsp\_access\_disable();

return 0;

}

/\* voice pcm \*/

static int scene\_voice\_pcm\_startup(struct snd\_pcm\_substream \*substream,

struct snd\_soc\_dai \*dai)

{

int stream = substream->stream;

int scene\_id = VBC\_DAI\_ID\_VOICE\_PCM\_P;

int be\_dai\_id = dai->id;

struct vbc\_codec\_priv \*vbc\_codec = dev\_get\_drvdata(dai->dev);

int ret = 0;

pr\_info("%s dai:%s(%d) scene:%s %s\n", \_\_func\_\_,

dai\_id\_to\_str(be\_dai\_id),

be\_dai\_id, scene\_id\_to\_str(scene\_id), stream\_to\_str(stream));

if (scene\_id != check\_be\_dai\_id(be\_dai\_id)) {

pr\_err("%s check\_be\_dai\_id failed\n", \_\_func\_\_);

return -EINVAL;

}

if (!vbc\_codec)

return 0;

startup\_lock\_mtx(scene\_id, stream);

startup\_add\_ref(scene\_id, stream);

if (startup\_get\_ref(scene\_id, stream) == 1) {

set\_scene\_flag(scene\_id, stream);

}

startup\_unlock\_mtx(scene\_id, stream);

return ret;

}

static void scene\_voice\_pcm\_shutdown(struct snd\_pcm\_substream \*substream,

struct snd\_soc\_dai \*dai)

{

int stream = substream->stream;

int scene\_id = VBC\_DAI\_ID\_VOICE\_PCM\_P;

int be\_dai\_id = dai->id;

struct vbc\_codec\_priv \*vbc\_codec = dev\_get\_drvdata(dai->dev);

pr\_info("%s dai:%s(%d) scene:%s %s\n", \_\_func\_\_,

dai\_id\_to\_str(be\_dai\_id),

be\_dai\_id, scene\_id\_to\_str(scene\_id), stream\_to\_str(stream));

if (scene\_id != check\_be\_dai\_id(be\_dai\_id)) {

pr\_err("%s check\_be\_dai\_id failed\n", \_\_func\_\_);

return;

}

if (!vbc\_codec)

return;

startup\_lock\_mtx(scene\_id, stream);

startup\_dec\_ref(scene\_id, stream);

if (startup\_get\_ref(scene\_id, stream) == 0) {

clr\_scene\_flag(scene\_id, stream);

dsp\_vbc\_voice\_pcm\_play\_set(false,

vbc\_codec->voice\_pcm\_play\_mode);

}

startup\_unlock\_mtx(scene\_id, stream);

}

static int scene\_voice\_pcm\_hw\_params(struct snd\_pcm\_substream \*substream,

struct snd\_pcm\_hw\_params \*params, struct snd\_soc\_dai \*dai)

{

unsigned int rate;

int data\_fmt = VBC\_DAT\_L16;

int stream = substream->stream;

int scene\_id = VBC\_DAI\_ID\_VOICE\_PCM\_P;

struct vbc\_codec\_priv \*vbc\_codec = dev\_get\_drvdata(dai->dev);

int chan\_cnt;

pr\_info("%s dai:%s(%d) scene:%s %s\n", \_\_func\_\_,

dai\_id\_to\_str(dai->id),

dai->id, scene\_id\_to\_str(scene\_id), stream\_to\_str(stream));

if (scene\_id != check\_be\_dai\_id(dai->id)) {

pr\_err("%s check\_be\_dai\_id failed\n", \_\_func\_\_);

return -EINVAL;

}

if (!vbc\_codec)

return 0;

switch (params\_format(params)) {

case SNDRV\_PCM\_FORMAT\_S16\_LE:

data\_fmt = VBC\_DAT\_L16;

break;

case SNDRV\_PCM\_FORMAT\_S24\_LE:

data\_fmt = VBC\_DAT\_L24;

break;

default:

pr\_err("%s, ERR:VBC not support data fmt =%d", \_\_func\_\_,

data\_fmt);

break;

}

chan\_cnt = params\_channels(params);

rate = params\_rate(params);

pr\_info("%s data\_fmt=%s, chan=%u, rate =%u\n", \_\_func\_\_,

vbc\_data\_fmt\_to\_str(data\_fmt), chan\_cnt, rate);

if (chan\_cnt > 2)

pr\_warn("%s channel count invalid\n", \_\_func\_\_);

hw\_param\_lock\_mtx(scene\_id, stream);

hw\_param\_add\_ref(scene\_id, stream);

hw\_param\_unlock\_mtx(scene\_id, stream);

return 0;

}

static int scene\_voice\_pcm\_hw\_free(struct snd\_pcm\_substream \*substream,

struct snd\_soc\_dai \*dai)

{

int stream = substream->stream;

int scene\_id = VBC\_DAI\_ID\_VOICE\_PCM\_P;

struct vbc\_codec\_priv \*vbc\_codec = dev\_get\_drvdata(dai->dev);

pr\_info("%s dai:%s(%d) scene:%s %s\n", \_\_func\_\_, dai\_id\_to\_str(dai->id),

dai->id, scene\_id\_to\_str(scene\_id), stream\_to\_str(stream));

if (scene\_id != check\_be\_dai\_id(dai->id)) {

pr\_err("%s check\_be\_dai\_id failed\n", \_\_func\_\_);

return -EINVAL;

}

if (!vbc\_codec)

return 0;

hw\_param\_lock\_mtx(scene\_id, stream);

hw\_param\_dec\_ref(scene\_id, stream);

hw\_param\_unlock\_mtx(scene\_id, stream);

return 0;

}

static int scene\_voice\_pcm\_trigger(struct snd\_pcm\_substream \*substream,

int cmd, struct snd\_soc\_dai \*dai)

{

int up\_down;

int stream = substream->stream;

int scene\_id = VBC\_DAI\_ID\_VOICE\_PCM\_P;

int ret;

struct vbc\_codec\_priv \*vbc\_codec = dev\_get\_drvdata(dai->dev);

pr\_info("%s dai:%s(%d) scene:%s %s, cmd=%d\n", \_\_func\_\_,

dai\_id\_to\_str(dai->id),

dai->id, scene\_id\_to\_str(scene\_id), stream\_to\_str(stream), cmd);

if (scene\_id != check\_be\_dai\_id(dai->id)) {

pr\_err("%s check\_be\_dai\_id failed\n", \_\_func\_\_);

return -EINVAL;

}

if (!vbc\_codec)

return 0;

up\_down = triggered\_flag(cmd);

/\* default ret is 0 \*/

ret = 0;

if (up\_down == 1) {

trigger\_lock\_spin(scene\_id, stream);

trigger\_add\_ref(scene\_id, stream);

if (trigger\_get\_ref(scene\_id, stream) == 1) {

dsp\_vbc\_voice\_pcm\_play\_set(true,

vbc\_codec->voice\_pcm\_play\_mode);

}

trigger\_unlock\_spin(scene\_id, stream);

} else {

trigger\_lock\_spin(scene\_id, stream);

trigger\_dec\_ref(scene\_id, stream);

trigger\_unlock\_spin(scene\_id, stream);

}

return ret;

}

static struct snd\_soc\_dai\_ops voice\_pcm\_ops = {

.startup = scene\_voice\_pcm\_startup,

.shutdown = scene\_voice\_pcm\_shutdown,

.hw\_params = scene\_voice\_pcm\_hw\_params,

.trigger = scene\_voice\_pcm\_trigger,

.hw\_free = scene\_voice\_pcm\_hw\_free,

};

/\* HFP \*/

static int scene\_hfp\_startup(struct snd\_pcm\_substream \*substream,

struct snd\_soc\_dai \*dai)

{

int stream = substream->stream;

int scene\_id = VBC\_DAI\_ID\_HFP;

int be\_dai\_id = dai->id;

struct vbc\_codec\_priv \*vbc\_codec = dev\_get\_drvdata(dai->dev);

int ret = 0;

pr\_debug("%s dai:%s(%d) scene:%s %s\n", \_\_func\_\_,

dai\_id\_to\_str(be\_dai\_id),

be\_dai\_id, scene\_id\_to\_str(scene\_id),

stream\_to\_str(stream));

if (scene\_id != check\_be\_dai\_id(be\_dai\_id)) {

pr\_err("%s check\_be\_dai\_id failed\n", \_\_func\_\_);

return -EINVAL;

}

if (!vbc\_codec)

return 0;

startup\_lock\_mtx(scene\_id, stream);

startup\_add\_ref(scene\_id, stream);

if (startup\_get\_ref(scene\_id, stream) == 1) {

ret = dsp\_startup(vbc\_codec, scene\_id, stream);

if (ret)

startup\_dec\_ref(scene\_id, stream);

else

set\_scene\_flag(scene\_id, stream);

}

startup\_unlock\_mtx(scene\_id, stream);

return ret;

}

static void scene\_hfp\_shutdown(struct snd\_pcm\_substream \*substream,

struct snd\_soc\_dai \*dai)

{

int stream = substream->stream;

int scene\_id = VBC\_DAI\_ID\_HFP;

int be\_dai\_id = dai->id;

struct vbc\_codec\_priv \*vbc\_codec = dev\_get\_drvdata(dai->dev);

pr\_debug("%s dai:%s(%d) scene:%s %s\n", \_\_func\_\_,

dai\_id\_to\_str(be\_dai\_id),

be\_dai\_id, scene\_id\_to\_str(scene\_id),

stream\_to\_str(stream));

if (scene\_id != check\_be\_dai\_id(be\_dai\_id)) {

pr\_err("%s check\_be\_dai\_id failed\n", \_\_func\_\_);

return;

}

if (!vbc\_codec)

return;

startup\_lock\_mtx(scene\_id, stream);

startup\_dec\_ref(scene\_id, stream);

if (startup\_get\_ref(scene\_id, stream) == 0) {

clr\_scene\_flag(scene\_id, stream);

dsp\_shutdown(vbc\_codec, scene\_id, stream);

}

startup\_unlock\_mtx(scene\_id, stream);

}

static int scene\_hfp\_hw\_params(struct snd\_pcm\_substream \*substream,

struct snd\_pcm\_hw\_params \*params, struct snd\_soc\_dai \*dai)

{

unsigned int rate;

int data\_fmt = VBC\_DAT\_L16;

int stream = substream->stream;

int scene\_id = VBC\_DAI\_ID\_HFP;

struct vbc\_codec\_priv \*vbc\_codec = dev\_get\_drvdata(dai->dev);

int chan\_cnt;

pr\_debug("%s dai:%s(%d) scene:%s %s\n", \_\_func\_\_,

dai\_id\_to\_str(dai->id),

dai->id, scene\_id\_to\_str(scene\_id),

stream\_to\_str(stream));

if (scene\_id != check\_be\_dai\_id(dai->id)) {

pr\_err("%s check\_be\_dai\_id failed\n", \_\_func\_\_);

return -EINVAL;

}

if (!vbc\_codec)

return 0;

switch (params\_format(params)) {

case SNDRV\_PCM\_FORMAT\_S16\_LE:

data\_fmt = VBC\_DAT\_L16;

break;

case SNDRV\_PCM\_FORMAT\_S24\_LE:

data\_fmt = VBC\_DAT\_L24;

break;

default:

pr\_err("%s, ERR:VBC not support data fmt =%d", \_\_func\_\_,

data\_fmt);

break;

}

chan\_cnt = params\_channels(params);

rate = params\_rate(params);

pr\_debug("%s data\_fmt=%s, chan=%u, rate =%u\n", \_\_func\_\_,

vbc\_data\_fmt\_to\_str(data\_fmt), chan\_cnt, rate);

if (chan\_cnt > 2)

pr\_warn("%s channel count invalid\n", \_\_func\_\_);

hw\_param\_lock\_mtx(scene\_id, stream);

hw\_param\_add\_ref(scene\_id, stream);

if (hw\_param\_get\_ref(scene\_id, stream) == 1) {

dsp\_hw\_params(vbc\_codec, scene\_id, stream,

chan\_cnt, rate, data\_fmt);

}

hw\_param\_unlock\_mtx(scene\_id, stream);

return 0;

}

static int scene\_hfp\_hw\_free(struct snd\_pcm\_substream \*substream,

struct snd\_soc\_dai \*dai)

{

int stream = substream->stream;

int scene\_id = VBC\_DAI\_ID\_HFP;

struct vbc\_codec\_priv \*vbc\_codec = dev\_get\_drvdata(dai->dev);

pr\_debug("%s dai:%s(%d) scene:%s %s\n", \_\_func\_\_,

dai\_id\_to\_str(dai->id),

dai->id, scene\_id\_to\_str(scene\_id),

stream\_to\_str(stream));

if (scene\_id != check\_be\_dai\_id(dai->id)) {

pr\_err("%s check\_be\_dai\_id failed\n", \_\_func\_\_);

return -EINVAL;

}

if (!vbc\_codec)

return 0;

hw\_param\_lock\_mtx(scene\_id, stream);

hw\_param\_dec\_ref(scene\_id, stream);

hw\_param\_unlock\_mtx(scene\_id, stream);

return 0;

}

static int scene\_hfp\_trigger(struct snd\_pcm\_substream \*substream, int cmd,

struct snd\_soc\_dai \*dai)

{

int up\_down;

int stream = substream->stream;

int scene\_id = VBC\_DAI\_ID\_HFP;

int ret;

struct vbc\_codec\_priv \*vbc\_codec = dev\_get\_drvdata(dai->dev);

pr\_debug("%s dai:%s(%d) scene:%s %s, cmd=%d\n", \_\_func\_\_,

dai\_id\_to\_str(dai->id),

dai->id, scene\_id\_to\_str(scene\_id),

stream\_to\_str(stream), cmd);

if (scene\_id != check\_be\_dai\_id(dai->id)) {

pr\_err("%s check\_be\_dai\_id failed\n", \_\_func\_\_);

return -EINVAL;

}

if (!vbc\_codec)

return 0;

up\_down = triggered\_flag(cmd);

/\* default ret is 0 \*/

ret = 0;

if (up\_down == 1) {

trigger\_lock\_spin(scene\_id, stream);

trigger\_add\_ref(scene\_id, stream);

if (trigger\_get\_ref(scene\_id, stream) == 1)

ret = dsp\_trigger(vbc\_codec, scene\_id, stream, up\_down);

trigger\_unlock\_spin(scene\_id, stream);

} else {

trigger\_lock\_spin(scene\_id, stream);

trigger\_dec\_ref(scene\_id, stream);

trigger\_unlock\_spin(scene\_id, stream);

}

return ret;

}

static struct snd\_soc\_dai\_ops hfp\_ops = {

.startup = scene\_hfp\_startup,

.shutdown = scene\_hfp\_shutdown,

.hw\_params = scene\_hfp\_hw\_params,

.trigger = scene\_hfp\_trigger,

.hw\_free = scene\_hfp\_hw\_free,

};

static int scene\_hifi\_startup(struct snd\_pcm\_substream \*substream,

struct snd\_soc\_dai \*dai)

{

int stream = substream->stream;

int scene\_id = AUDCP\_DAI\_ID\_HIFI;

int be\_dai\_id = dai->id;

int ret = 0;

pr\_info("%s dai:%s(%d) scene:%s %s\n", \_\_func\_\_,

dai\_id\_to\_str(be\_dai\_id),

be\_dai\_id, scene\_id\_to\_str(scene\_id), stream\_to\_str(stream));

if (scene\_id != check\_be\_dai\_id(be\_dai\_id)) {

pr\_err("%s check\_be\_dai\_id failed\n", \_\_func\_\_);

return -EINVAL;

}

startup\_lock\_mtx(scene\_id, stream);

startup\_add\_ref(scene\_id, stream);

if (startup\_get\_ref(scene\_id, stream) == 1) {

ret = hifi\_startup(scene\_id, stream);

if (ret)

startup\_dec\_ref(scene\_id, stream);

else

set\_scene\_flag(scene\_id, stream);

}

startup\_unlock\_mtx(scene\_id, stream);

return ret;

}

static void scene\_hifi\_shutdown(struct snd\_pcm\_substream \*substream,

struct snd\_soc\_dai \*dai)

{

int stream = substream->stream;

int scene\_id = AUDCP\_DAI\_ID\_HIFI;

int be\_dai\_id = dai->id;

pr\_info("%s dai:%s(%d) scene:%s %s\n", \_\_func\_\_,

dai\_id\_to\_str(be\_dai\_id),

be\_dai\_id, scene\_id\_to\_str(scene\_id), stream\_to\_str(stream));

if (scene\_id != check\_be\_dai\_id(be\_dai\_id)) {

pr\_err("%s check\_be\_dai\_id failed\n", \_\_func\_\_);

return;

}

startup\_lock\_mtx(scene\_id, stream);

startup\_dec\_ref(scene\_id, stream);

if (startup\_get\_ref(scene\_id, stream) == 0) {

clr\_scene\_flag(scene\_id, stream);

hifi\_shutdown(scene\_id, stream);

}

startup\_unlock\_mtx(scene\_id, stream);

}

static int scene\_hifi\_hw\_params(struct snd\_pcm\_substream \*substream,

struct snd\_pcm\_hw\_params \*params, struct snd\_soc\_dai \*dai)

{

unsigned int rate;

int data\_fmt = VBC\_DAT\_L16;

int stream = substream->stream;

int scene\_id = AUDCP\_DAI\_ID\_HIFI;

int chan\_cnt;

pr\_info("%s dai:%s(%d) scene:%s %s\n", \_\_func\_\_, dai\_id\_to\_str(dai->id),

dai->id, scene\_id\_to\_str(scene\_id), stream\_to\_str(stream));

if (scene\_id != check\_be\_dai\_id(dai->id)) {

pr\_err("%s check\_be\_dai\_id failed\n", \_\_func\_\_);

return -EINVAL;

}

switch (params\_format(params)) {

case SNDRV\_PCM\_FORMAT\_S16\_LE:

data\_fmt = VBC\_DAT\_L16;

break;

case SNDRV\_PCM\_FORMAT\_S24\_LE:

data\_fmt = VBC\_DAT\_L24;

break;

default:

pr\_err("%s, ERR:VBC not support data fmt =%d", \_\_func\_\_,

data\_fmt);

break;

}

chan\_cnt = params\_channels(params);

rate = params\_rate(params);

pr\_info("%s data\_fmt=%s, chan=%u, rate =%u\n", \_\_func\_\_,

vbc\_data\_fmt\_to\_str(data\_fmt), chan\_cnt, rate);

if (chan\_cnt > 2)

pr\_warn("%s channel count invalid\n", \_\_func\_\_);

hw\_param\_lock\_mtx(scene\_id, stream);

hw\_param\_add\_ref(scene\_id, stream);

if (hw\_param\_get\_ref(scene\_id, stream) == 1) {

hifi\_hw\_params(scene\_id, stream,

chan\_cnt, rate, data\_fmt);

}

hw\_param\_unlock\_mtx(scene\_id, stream);

return 0;

}

static int scene\_hifi\_hw\_free(struct snd\_pcm\_substream \*substream,

struct snd\_soc\_dai \*dai)

{

int stream = substream->stream;

int scene\_id = AUDCP\_DAI\_ID\_HIFI;

pr\_info("%s dai:%s(%d) scene:%s %s\n", \_\_func\_\_, dai\_id\_to\_str(dai->id),

dai->id, scene\_id\_to\_str(scene\_id), stream\_to\_str(stream));

if (scene\_id != check\_be\_dai\_id(dai->id)) {

pr\_err("%s check\_be\_dai\_id failed\n", \_\_func\_\_);

return -EINVAL;

}

hw\_param\_lock\_mtx(scene\_id, stream);

hw\_param\_dec\_ref(scene\_id, stream);

hw\_param\_unlock\_mtx(scene\_id, stream);

return 0;

}

static int scene\_hifi\_trigger(struct snd\_pcm\_substream \*substream, int cmd,

struct snd\_soc\_dai \*dai)

{

int up\_down;

int stream = substream->stream;

int scene\_id = AUDCP\_DAI\_ID\_HIFI;

int ret;

pr\_info("%s dai:%s(%d) scene:%s %s, cmd=%d\n", \_\_func\_\_,

dai\_id\_to\_str(dai->id),

dai->id, scene\_id\_to\_str(scene\_id), stream\_to\_str(stream), cmd);

if (scene\_id != check\_be\_dai\_id(dai->id)) {

pr\_err("%s check\_be\_dai\_id failed\n", \_\_func\_\_);

return -EINVAL;

}

up\_down = triggered\_flag(cmd);

ret = 0;

if (up\_down == 1) {

trigger\_lock\_spin(scene\_id, stream);

trigger\_add\_ref(scene\_id, stream);

if (trigger\_get\_ref(scene\_id, stream) == 1)

ret = hifi\_dsp\_trigger(scene\_id, stream, up\_down);

trigger\_unlock\_spin(scene\_id, stream);

} else {

trigger\_lock\_spin(scene\_id, stream);

trigger\_dec\_ref(scene\_id, stream);

trigger\_unlock\_spin(scene\_id, stream);

}

return ret;

}

static struct snd\_soc\_dai\_ops hifi\_ops = {

.startup = scene\_hifi\_startup,

.shutdown = scene\_hifi\_shutdown,

.hw\_params = scene\_hifi\_hw\_params,

.trigger = scene\_hifi\_trigger,

.hw\_free = scene\_hifi\_hw\_free,

};

static int scene\_hifi\_fast\_startup(struct snd\_pcm\_substream \*substream,

struct snd\_soc\_dai \*dai)

{

int stream = substream->stream;

int scene\_id = AUDCP\_DAI\_ID\_FAST;

int be\_dai\_id = dai->id;

int ret = 0;

pr\_info("%s dai:%s(%d) scene:%s %s\n", \_\_func\_\_,

dai\_id\_to\_str(be\_dai\_id),

be\_dai\_id, scene\_id\_to\_str(scene\_id), stream\_to\_str(stream));

if (scene\_id != check\_be\_dai\_id(be\_dai\_id)) {

pr\_err("%s check\_be\_dai\_id failed\n", \_\_func\_\_);

return -EINVAL;

}

startup\_lock\_mtx(scene\_id, stream);

startup\_add\_ref(scene\_id, stream);

if (startup\_get\_ref(scene\_id, stream) == 1) {

ret = hifi\_startup(scene\_id, stream);

if (ret)

startup\_dec\_ref(scene\_id, stream);

else

set\_scene\_flag(scene\_id, stream);

}

startup\_unlock\_mtx(scene\_id, stream);

return ret;

}

static void scene\_hifi\_fast\_shutdown(struct snd\_pcm\_substream \*substream,

struct snd\_soc\_dai \*dai)

{

int stream = substream->stream;

int scene\_id = AUDCP\_DAI\_ID\_FAST;

int be\_dai\_id = dai->id;

pr\_info("%s dai:%s(%d) scene:%s %s\n", \_\_func\_\_,

dai\_id\_to\_str(be\_dai\_id),

be\_dai\_id, scene\_id\_to\_str(scene\_id), stream\_to\_str(stream));

if (scene\_id != check\_be\_dai\_id(be\_dai\_id)) {

pr\_err("%s check\_be\_dai\_id failed\n", \_\_func\_\_);

return;

}

startup\_lock\_mtx(scene\_id, stream);

startup\_dec\_ref(scene\_id, stream);

if (startup\_get\_ref(scene\_id, stream) == 0) {

clr\_scene\_flag(scene\_id, stream);

hifi\_shutdown(scene\_id, stream);

}

startup\_unlock\_mtx(scene\_id, stream);

}

static int scene\_hifi\_fast\_hw\_params(struct snd\_pcm\_substream \*substream,

struct snd\_pcm\_hw\_params \*params, struct snd\_soc\_dai \*dai)

{

unsigned int rate;

int data\_fmt = VBC\_DAT\_L16;

int stream = substream->stream;

int scene\_id = AUDCP\_DAI\_ID\_FAST;

int chan\_cnt;

pr\_info("%s dai:%s(%d) scene:%s %s\n", \_\_func\_\_, dai\_id\_to\_str(dai->id),

dai->id, scene\_id\_to\_str(scene\_id), stream\_to\_str(stream));

if (scene\_id != check\_be\_dai\_id(dai->id)) {

pr\_err("%s check\_be\_dai\_id failed\n", \_\_func\_\_);

return -EINVAL;

}

switch (params\_format(params)) {

case SNDRV\_PCM\_FORMAT\_S16\_LE:

data\_fmt = VBC\_DAT\_L16;

break;

case SNDRV\_PCM\_FORMAT\_S24\_LE:

data\_fmt = VBC\_DAT\_L24;

break;

default:

pr\_err("%s, ERR:VBC not support data fmt =%d", \_\_func\_\_,

data\_fmt);

break;

}

chan\_cnt = params\_channels(params);

rate = params\_rate(params);

pr\_info("%s data\_fmt=%s, chan=%u, rate =%u\n", \_\_func\_\_,

vbc\_data\_fmt\_to\_str(data\_fmt), chan\_cnt, rate);

if (chan\_cnt > 2)

pr\_warn("%s channel count invalid\n", \_\_func\_\_);

hw\_param\_lock\_mtx(scene\_id, stream);

hw\_param\_add\_ref(scene\_id, stream);

if (hw\_param\_get\_ref(scene\_id, stream) == 1) {

hifi\_hw\_params(scene\_id, stream,

chan\_cnt, rate, data\_fmt);

}

hw\_param\_unlock\_mtx(scene\_id, stream);

return 0;

}

static int scene\_hifi\_fast\_hw\_free(struct snd\_pcm\_substream \*substream,

struct snd\_soc\_dai \*dai)

{

int stream = substream->stream;

int scene\_id = AUDCP\_DAI\_ID\_FAST;

pr\_info("%s dai:%s(%d) scene:%s %s\n", \_\_func\_\_, dai\_id\_to\_str(dai->id),

dai->id, scene\_id\_to\_str(scene\_id), stream\_to\_str(stream));

if (scene\_id != check\_be\_dai\_id(dai->id)) {

pr\_err("%s check\_be\_dai\_id failed\n", \_\_func\_\_);

return -EINVAL;

}

hw\_param\_lock\_mtx(scene\_id, stream);

hw\_param\_dec\_ref(scene\_id, stream);

hw\_param\_unlock\_mtx(scene\_id, stream);

return 0;

}

static int scene\_hifi\_fast\_trigger(struct snd\_pcm\_substream \*substream, int cmd,

struct snd\_soc\_dai \*dai)

{

int up\_down;

int stream = substream->stream;

int scene\_id = AUDCP\_DAI\_ID\_FAST;

int ret;

pr\_info("%s dai:%s(%d) scene:%s %s, cmd=%d\n", \_\_func\_\_,

dai\_id\_to\_str(dai->id),

dai->id, scene\_id\_to\_str(scene\_id), stream\_to\_str(stream), cmd);

if (scene\_id != check\_be\_dai\_id(dai->id)) {

pr\_err("%s check\_be\_dai\_id failed\n", \_\_func\_\_);

return -EINVAL;

}

up\_down = triggered\_flag(cmd);

ret = 0;

if (up\_down == 1) {

trigger\_lock\_spin(scene\_id, stream);

trigger\_add\_ref(scene\_id, stream);

if (trigger\_get\_ref(scene\_id, stream) == 1)

ret = hifi\_dsp\_trigger(scene\_id, stream, up\_down);

trigger\_unlock\_spin(scene\_id, stream);

} else {

trigger\_lock\_spin(scene\_id, stream);

trigger\_dec\_ref(scene\_id, stream);

trigger\_unlock\_spin(scene\_id, stream);

}

return ret;

}

static struct snd\_soc\_dai\_ops hifi\_fast\_ops = {

.startup = scene\_hifi\_fast\_startup,

.shutdown = scene\_hifi\_fast\_shutdown,

.hw\_params = scene\_hifi\_fast\_hw\_params,

.trigger = scene\_hifi\_fast\_trigger,

.hw\_free = scene\_hifi\_fast\_hw\_free,

};

static struct snd\_soc\_dai\_driver vbc\_dais[BE\_DAI\_ID\_MAX] = {

/\* 0: BE\_DAI\_ID\_NORMAL\_AP01\_CODEC \*/

{

.name = TO\_STRING(BE\_DAI\_ID\_NORMAL\_AP01\_CODEC),

.id = BE\_DAI\_ID\_NORMAL\_AP01\_CODEC,

.playback = {

.stream\_name = "BE\_DAI\_NORMAL\_AP01\_CODEC\_P",

.aif\_name = "BE\_IF\_NORMAL\_AP01\_CODEC\_P",

.channels\_min = 1,

.channels\_max = 2,

.rates = SNDRV\_PCM\_RATE\_CONTINUOUS,

.rate\_max = 192000,

.formats = SPRD\_VBC\_DAI\_PCM\_FORMATS,

},

.capture = {

.stream\_name = "BE\_DAI\_NORMAL\_AP01\_CODEC\_C",

.aif\_name = "BE\_IF\_NORMAL\_AP01\_CODEC\_C",

.channels\_min = 1,

.channels\_max = 2,

.rates = SNDRV\_PCM\_RATE\_CONTINUOUS,

.rate\_max = 192000,

.formats = SPRD\_VBC\_DAI\_PCM\_FORMATS,

},

.probe = sprd\_dai\_vbc\_probe,

.ops = &normal\_ops,

.resume = normal\_resume,

.suspend = normal\_suspend,

.bus\_control = true,

},

/\* 34: BE\_DAI\_ID\_NORMAL\_AP01\_P\_HIFI \*/

{

.name = TO\_STRING(BE\_DAI\_ID\_NORMAL\_AP01\_P\_HIFI),

.id = BE\_DAI\_ID\_NORMAL\_AP01\_P\_HIFI,

.playback = {

.stream\_name = "BE\_DAI\_ID\_NORMAL\_AP01\_P\_HIFI",

.aif\_name = "BE\_IF\_NORMAL\_AP01\_HIFI\_P",

.channels\_min = 1,

.channels\_max = 2,

.rates = SNDRV\_PCM\_RATE\_CONTINUOUS,

.rate\_max = 192000,

.formats = SPRD\_VBC\_DAI\_PCM\_FORMATS,

},

.probe = sprd\_dai\_vbc\_probe,

.ops = &normal\_ops,

},

/\* 35: BE\_DAI\_ID\_NORMAL\_AP23\_HIFI \*/

{

.name = TO\_STRING(BE\_DAI\_ID\_NORMAL\_AP23\_HIFI),

.id = BE\_DAI\_ID\_NORMAL\_AP23\_HIFI,

.playback = {

.stream\_name = "BE\_DAI\_ID\_NORMAL\_AP23\_HIFI",

.aif\_name = "BE\_IF\_ID\_NORMAL\_AP23\_HIFI",

.channels\_min = 1,

.channels\_max = 2,

.rates = SNDRV\_PCM\_RATE\_CONTINUOUS,

.rate\_max = 192000,

.formats = SPRD\_VBC\_DAI\_PCM\_FORMATS,

},

.probe = sprd\_dai\_vbc\_probe,

.ops = &normal\_ap23\_ops,

},

/\* 36: BE\_DAI\_ID\_FAST\_P\_HIFI \*/

{

.name = TO\_STRING(BE\_DAI\_ID\_FAST\_P\_HIFI),

.id = BE\_DAI\_ID\_FAST\_P\_HIFI,

.playback = {

.stream\_name = "BE\_DAI\_ID\_FAST\_P\_HIFI",

.aif\_name = "BE\_IF\_ID\_FAST\_P\_HIFI",

.channels\_min = 1,

.channels\_max = 2,

.rates = SNDRV\_PCM\_RATE\_CONTINUOUS,

.rate\_max = 192000,

.formats = SPRD\_VBC\_DAI\_PCM\_FORMATS,

},

.probe = sprd\_dai\_vbc\_probe,

.ops = &fast\_ops,

},

/\* 37: BE\_DAI\_ID\_OFFLOAD\_HIFI \*/

{

.name = TO\_STRING(BE\_DAI\_ID\_OFFLOAD\_HIFI),

.id = BE\_DAI\_ID\_OFFLOAD\_HIFI,

.playback = {

.stream\_name = "BE\_DAI\_ID\_OFFLOAD\_HIFI",

.aif\_name = "BE\_IF\_ID\_OFFLOAD\_HIFI",

.channels\_min = 1,

.channels\_max = 2,

.rates = SNDRV\_PCM\_RATE\_CONTINUOUS,

.rate\_max = 192000,

.formats = SPRD\_VBC\_DAI\_PCM\_FORMATS,

},

.probe = sprd\_dai\_vbc\_probe,

.ops = &offload\_ops,

},

/\* 38: BE\_DAI\_ID\_VOICE\_HIFI \*/

{

.name = TO\_STRING(BE\_DAI\_ID\_VOICE\_HIFI),

.id = BE\_DAI\_ID\_VOICE\_HIFI,

.playback = {

.stream\_name = "BE\_DAI\_ID\_VOICE\_HIFI",

.aif\_name = "BE\_IF\_ID\_VOICE\_HIFI",

.channels\_min = 1,

.channels\_max = 2,

.rates = SNDRV\_PCM\_RATE\_CONTINUOUS,

.rate\_max = 192000,

.formats = SPRD\_VBC\_DAI\_PCM\_FORMATS,

},

.probe = sprd\_dai\_vbc\_probe,

.ops = &voice\_ops,

},

/\* 39: BE\_DAI\_ID\_VOIP\_HIFI \*/

{

.name = TO\_STRING(BE\_DAI\_ID\_VOIP\_HIFI),

.id = BE\_DAI\_ID\_VOIP\_HIFI,

.playback = {

.stream\_name = "BE\_DAI\_ID\_VOIP\_HIFI",

.aif\_name = "BE\_IF\_ID\_VOIP\_HIFI",

.channels\_min = 1,

.channels\_max = 2,

.rates = SNDRV\_PCM\_RATE\_CONTINUOUS,

.rate\_max = 192000,

.formats = SPRD\_VBC\_DAI\_PCM\_FORMATS,

},

.probe = sprd\_dai\_vbc\_probe,

.ops = &voip\_ops,

},

/\* 40: BE\_DAI\_ID\_FM\_HIFI \*/

{

.name = TO\_STRING(BE\_DAI\_ID\_FM\_HIFI),

.id = BE\_DAI\_ID\_FM\_HIFI,

.playback = {

.stream\_name = "BE\_DAI\_ID\_FM\_HIFI",

.aif\_name = "BE\_IF\_ID\_FM\_HIFI",

.channels\_min = 1,

.channels\_max = 2,

.rates = SNDRV\_PCM\_RATE\_CONTINUOUS,

.rate\_max = 192000,

.formats = SPRD\_VBC\_DAI\_PCM\_FORMATS,

},

.probe = sprd\_dai\_vbc\_probe,

.ops = &fm\_ops,

},

/\* 41: BE\_DAI\_ID\_LOOP\_HIFI \*/

{

.name = TO\_STRING(BE\_DAI\_ID\_LOOP\_HIFI),

.id = BE\_DAI\_ID\_LOOP\_HIFI,

.playback = {

.stream\_name = "BE\_DAI\_ID\_LOOP\_HIFI",

.aif\_name = "BE\_IF\_ID\_LOOP\_HIFI",

.channels\_min = 1,

.channels\_max = 2,

.rates = SNDRV\_PCM\_RATE\_CONTINUOUS,

.rate\_max = 192000,

.formats = SPRD\_VBC\_DAI\_PCM\_FORMATS,

},

.probe = sprd\_dai\_vbc\_probe,

.ops = &loop\_ops,

},

/\* 42: BE\_DAI\_ID\_FM\_DSP\_HIFI \*/

{

.name = TO\_STRING(BE\_DAI\_ID\_FM\_DSP\_HIFI),

.id = BE\_DAI\_ID\_FM\_DSP\_HIFI,

.playback = {

.stream\_name = "BE\_DAI\_ID\_FM\_DSP\_HIFI",

.aif\_name = "BE\_IF\_ID\_FM\_DSP\_HIFI",

.channels\_min = 1,

.channels\_max = 2,

.rates = SNDRV\_PCM\_RATE\_CONTINUOUS,

.rate\_max = 192000,

.formats = SPRD\_VBC\_DAI\_PCM\_FORMATS,

},

.probe = sprd\_dai\_vbc\_probe,

.ops = &fm\_dsp\_ops,

},

/\* 43: BE\_DAI\_ID\_HFP \*/

{

.name = TO\_STRING(BE\_DAI\_ID\_HFP),

.id = BE\_DAI\_ID\_HFP,

.playback = {

.stream\_name = "BE\_DAI\_HFP\_P",

.aif\_name = "BE\_IF\_HFP\_P",

.channels\_min = 1,

.channels\_max = 2,

.rates = SNDRV\_PCM\_RATE\_CONTINUOUS,

.rate\_max = 192000,

.formats = SPRD\_VBC\_DAI\_PCM\_FORMATS,

},

.capture = {

.stream\_name = "BE\_DAI\_HFP\_C",

.aif\_name = "BE\_IF\_HFP\_C",

.channels\_min = 1,

.channels\_max = 2,

.rates = SNDRV\_PCM\_RATE\_CONTINUOUS,

.rate\_max = 192000,

.formats = SPRD\_VBC\_DAI\_PCM\_FORMATS,

},

.probe = sprd\_dai\_vbc\_probe,

.ops = &hfp\_ops,

},

/\* 44: BE\_DAI\_ID\_RECOGNISE\_CAPTURE \*/

{

.name = TO\_STRING(BE\_DAI\_ID\_RECOGNISE\_CAPTURE),

.id = BE\_DAI\_ID\_RECOGNISE\_CAPTURE,

.capture = {

.stream\_name = "BE\_DAI\_CAP\_RECOGNISE\_CODEC\_C",

.aif\_name = "BE\_IF\_CAP\_RECOGNISE\_CODEC\_C",

.channels\_min = 1,

.channels\_max = 2,

.rates = SNDRV\_PCM\_RATE\_CONTINUOUS,

.rate\_max = 192000,

.formats = SPRD\_VBC\_DAI\_PCM\_FORMATS,

},

.probe = sprd\_dai\_vbc\_probe,

.ops = &recognise\_capture\_ops,

},

/\* 45: BE\_DAI\_ID\_VOICE\_PCM\_P \*/

{

.name = TO\_STRING(BE\_DAI\_ID\_VOICE\_PCM\_P),

.id = BE\_DAI\_ID\_VOICE\_PCM\_P,

.playback = {

.stream\_name = "BE\_DAI\_VOICE\_PCM\_P",

.aif\_name = "BE\_IF\_VOICE\_PCM\_P",

.channels\_min = 1,

.channels\_max = 2,

.rates = SNDRV\_PCM\_RATE\_CONTINUOUS,

.rate\_max = 192000,

.formats = SPRD\_VBC\_DAI\_PCM\_FORMATS,

},

.probe = sprd\_dai\_vbc\_probe,

.ops = &voice\_pcm\_ops,

},

/\* 55: BE\_DAI\_ID\_HIFI\_P \*/

{

.name = TO\_STRING(BE\_DAI\_ID\_HIFI\_P),

.id = BE\_DAI\_ID\_HIFI\_P,

.playback = {

.stream\_name = "BE\_DAI\_HIFI\_P",

.aif\_name = "BE\_IF\_HIFI\_P",

.channels\_min = 1,

.channels\_max = 2,

.rates = SNDRV\_PCM\_RATE\_CONTINUOUS,

.rate\_max = 192000,

.formats = SPRD\_VBC\_DAI\_PCM\_FORMATS,

},

.probe = sprd\_dai\_vbc\_probe,

.ops = &hifi\_ops,

},

/\* 56: BE\_DAI\_ID\_HIFI\_FAST\_P \*/

{

.name = TO\_STRING(BE\_DAI\_ID\_HIFI\_FAST\_P),

.id = BE\_DAI\_ID\_HIFI\_FAST\_P,

.playback = {

.stream\_name = "BE\_DAI\_HIFI\_FAST\_P",

.aif\_name = "BE\_IF\_HIFI\_FAST\_P",

.channels\_min = 1,

.channels\_max = 2,

.rates = SNDRV\_PCM\_RATE\_CONTINUOUS,

.rate\_max = 192000,

.formats = SPRD\_VBC\_DAI\_PCM\_FORMATS,

},

.probe = sprd\_dai\_vbc\_probe,

.ops = &hifi\_fast\_ops,

},

};

static int vbc\_drv\_probe(struct platform\_device \*pdev)

{

int ret;

struct vbc\_codec\_priv \*vbc\_codec = NULL;

pr\_info("%s: to setup vbc dt\n", \_\_func\_\_);

/\* 1. probe CODEC \*/

ret = sprd\_vbc\_codec\_probe(pdev);

if (ret < 0)

goto probe\_err;

vbc\_codec = platform\_get\_drvdata(pdev);

/\*

\* should first call sprd\_vbc\_codec\_probe

\* because we will call platform\_get\_drvdata(pdev)

\*/

ret = vbc\_of\_setup(pdev);

if (ret < 0) {

pr\_err("%s: failed to setup vbc dt, ret=%d\n", \_\_func\_\_, ret);

return -ENODEV;

}

aud\_ipc\_ch\_open(AMSG\_CH\_DSP\_GET\_PARAM\_FROM\_SMSG\_NOREPLY);

aud\_ipc\_ch\_open(AMSG\_CH\_VBC\_CTL);

aud\_ipc\_ch\_open(AMSG\_CH\_DSP\_HIFI);

/\* 2. probe DAIS \*/

ret = snd\_soc\_register\_codec(&pdev->dev, &sprd\_vbc\_codec, vbc\_dais,

ARRAY\_SIZE(vbc\_dais));

sound\soc\sprd\dai\vbc\v4\vbc-phy-v4.c

/\* SND\_KCTL\_TYPE\_FM\_MUTE \*/

int fm\_mute\_set(int id, u16 mute, u16 step)

{

int ret;

struct vbc\_fm\_mute\_step mute\_step = { };

struct vbc\_fm\_mute\_en\_para fm\_mute\_en = { };

struct vbc\_fm\_mute\_para mute\_p = { };

mute\_step.step = step;

fm\_mute\_en.id = id;

fm\_mute\_en.enable = FM\_MUTE\_ENABLE;

mute\_p.id = id;

mute\_p.mute = mute;

ret = aud\_send\_cmd(AMSG\_CH\_VBC\_CTL, SND\_KCTL\_TYPE\_FM\_MDG\_STP,

-1, SND\_VBC\_DSP\_IO\_KCTL\_SET, &mute\_step,

sizeof(struct vbc\_fm\_mute\_step), AUDIO\_SIPC\_WAIT\_FOREVER);

if (ret < 0)

pr\_err("%s, Failed to set, ret: %d\n", \_\_func\_\_, ret);

ret = aud\_send\_cmd(AMSG\_CH\_VBC\_CTL, SND\_KCTL\_TYPE\_FM\_MUTE\_EN,

-1, SND\_VBC\_DSP\_IO\_KCTL\_SET,

&fm\_mute\_en, sizeof(struct vbc\_fm\_mute\_en\_para),

AUDIO\_SIPC\_WAIT\_FOREVER);

if (ret < 0)

pr\_err("%s, Failed to set, ret: %d\n", \_\_func\_\_, ret);

ret = aud\_send\_cmd(AMSG\_CH\_VBC\_CTL, SND\_KCTL\_TYPE\_FM\_MUTE,

-1, SND\_VBC\_DSP\_IO\_KCTL\_SET, &mute\_p,

sizeof(struct vbc\_fm\_mute\_para), AUDIO\_SIPC\_WAIT\_FOREVER);

if (ret < 0)

pr\_err("%s, Failed to set, ret: %d\n", \_\_func\_\_, ret);

return 0;

}

/\* SND\_KCTL\_TYPE\_VOICE\_MIX\_UL \*/

int dsp\_vbc\_voice\_pcm\_play\_set(bool enable, int mode)

{

int ret;

struct vbc\_voice\_pcm\_play\_t play\_mode;

sp\_asoc\_pr\_dbg("%s enable =%d, mode = %d\n",

\_\_func\_\_, enable, mode);

play\_mode.mix\_pcm\_enable = enable;

play\_mode.mix\_pcm\_mode = mode;

/\* send audio cmd \*/

sp\_asoc\_pr\_dbg("cmd=%d, parameter0=%d\n", SND\_VBC\_DSP\_IO\_KCTL\_SET,

SND\_KCTL\_TYPE\_VOICE\_MIX\_UL);

ret = aud\_send\_cmd\_no\_wait\_param(AMSG\_CH\_VBC\_CTL,

SND\_KCTL\_TYPE\_VOICE\_MIX\_UL, -1, SND\_VBC\_DSP\_IO\_KCTL\_SET,

&play\_mode, sizeof(struct vbc\_voice\_pcm\_play\_t));

if (ret < 0)

pr\_err("%s, Failed to set, ret: %d\n", \_\_func\_\_, ret);

return 0;

}

int hifi\_func\_startup(int scene\_id, int stream,

struct snd\_pcm\_hifi\_stream \*hifi\_startup\_info)

{

int ret;

ret = aud\_send\_cmd(AMSG\_CH\_DSP\_HIFI, scene\_id, stream,

SND\_VBC\_DSP\_FUNC\_STARTUP,

hifi\_startup\_info, sizeof(struct snd\_pcm\_hifi\_stream),

AUDIO\_SIPC\_WAIT\_FOREVER);

if (ret < 0)

return -EIO;

return 0;

}

int hifi\_func\_shutdown(int scene\_id, int stream,

struct snd\_pcm\_hifi\_stream \*hifi\_shutdown\_info)

{

int ret;

ret = aud\_send\_cmd(AMSG\_CH\_DSP\_HIFI, scene\_id, stream,

SND\_VBC\_DSP\_FUNC\_SHUTDOWN,

hifi\_shutdown\_info, sizeof(struct snd\_pcm\_hifi\_stream),

AUDIO\_SIPC\_WAIT\_FOREVER);

if (ret < 0)

return -EIO;

return 0;

}

int hifi\_dsp\_func\_hwparam(int scene\_id, int stream,

struct sprd\_vbc\_stream\_hw\_paras \*hifi\_data)

{

int ret;

ret = aud\_send\_cmd(AMSG\_CH\_DSP\_HIFI, scene\_id, stream,

SND\_VBC\_DSP\_FUNC\_HW\_PARAMS, hifi\_data,

sizeof(struct sprd\_vbc\_stream\_hw\_paras),

AUDIO\_SIPC\_WAIT\_FOREVER);

if (ret < 0)

return -EIO;

return 0;

}

int hifi\_func\_trigger(int id, int stream, int up\_down)

{

int ret;

/\* send audio cmd \*/

ret = aud\_send\_cmd\_no\_wait(AMSG\_CH\_DSP\_HIFI,

SND\_VBC\_DSP\_FUNC\_HW\_TRIGGER, id, stream,

up\_down, 0);

if (ret < 0)

return -EIO;

return 0;

}

sound\soc\sprd\dai\vbc\v4\vbc-phy-v4.h

nt dsp\_call\_mute\_set(int id, u16 mute);

int fm\_mute\_set(int id, u16 mute, u16 step);

int dsp\_vbc\_iis\_tx\_width\_set(int id, u32 width);

int dsp\_vbc\_iis\_tx\_lr\_mod\_set(int id, u32 lr\_mod);

int dsp\_vbc\_iis\_rx\_width\_set(int id, u32 width);

int dsp\_vbc\_iis\_rx\_lr\_mod\_set(int id, u32 lr\_mod);

int dsp\_vbc\_mst\_sel\_type\_set(int id, u32 type);

int dsp\_vbc\_iis\_master\_start(u32 enable);

void dsp\_vbc\_iis\_master\_width\_set(u32 iis\_width);

int dsp\_vbc\_mainmic\_path\_set(int type, int val);

int dsp\_ivsence\_func(int enable, int iv\_adc\_id);

int dsp\_vbc\_voice\_pcm\_play\_set(bool enable, int mode);

int vbc\_dsp\_func\_startup(int scene\_id, int stream,

struct sprd\_vbc\_stream\_startup\_shutdown \*startup\_info);

int vbc\_dsp\_func\_shutdown(int scene\_id, int stream,

struct sprd\_vbc\_stream\_startup\_shutdown \*shutdown\_info);

int vbc\_dsp\_func\_hwparam(int scene\_id, int stream,

struct sprd\_vbc\_stream\_hw\_paras \*hw\_data);

int vbc\_dsp\_func\_trigger(int id, int stream, int up\_down);

int aud\_dig\_iis\_master(struct snd\_soc\_card \*card, int setting);

int pm\_shutdown(void);

int hifi\_dsp\_func\_hwparam(int scene\_id, int stream,

struct sprd\_vbc\_stream\_hw\_paras \*hifi\_data);

int hifi\_func\_startup(int scene\_id, int stream,

struct snd\_pcm\_hifi\_stream \*hifi\_startup\_info);

int hifi\_func\_shutdown(int scene\_id, int stream,

struct snd\_pcm\_hifi\_stream \*hifi\_shutdown\_info);

int hifi\_func\_trigger(int id, int stream, int up\_down);

#endif /\* \_\_VBC\_V4\_PHY\_DRV\_H \*/

sound\soc\sprd\platform\sprd-dmaengine-pcm.c

static int is\_no\_pcm\_dai(int fe\_dai\_id)

{

int ret;

switch (fe\_dai\_id) {

case FE\_DAI\_ID\_VOICE:

case FE\_DAI\_ID\_FM:

case FE\_DAI\_ID\_FM\_DSP:

case FE\_DAI\_ID\_CODEC\_TEST:

case FE\_DAI\_ID\_HFP:

ret = 1;

break;

default:

ret = 0;

break;

}

return ret;

}

sound\soc\sprd\platform\sprd-platform-pcm-routing.c

enum SPRD\_BE\_SWITCH {

S\_NORMAL\_AP01\_P\_CODEC = 0,

S\_NORMAL\_AP01\_C\_CODEC,

S\_NORMAL\_AP23\_P\_CODEC,

S\_NORMAL\_AP23\_C\_CODEC,

S\_CAPTURE\_DSP\_CODEC,

S\_FAST\_P\_CODEC,

S\_OFFLOAD\_CODEC,

S\_VOICE\_P\_CODEC,

S\_VOICE\_C\_CODEC,

S\_VOIP\_P\_CODEC,

S\_VOIP\_C\_CODEC,

S\_FM\_CODEC,

S\_LOOP\_P\_CODEC,

S\_LOOP\_C\_CODEC,

S\_FM\_DSP\_CODEC,

S\_NORMAL\_AP01\_P\_USB,

S\_NORMAL\_AP01\_C\_USB,

S\_NORMAL\_AP23\_P\_USB,

S\_NORMAL\_AP23\_C\_USB,

S\_CAPTURE\_DSP\_USB,

S\_FAST\_P\_USB,

S\_OFFLOAD\_USB,

S\_VOICE\_P\_USB,

S\_VOICE\_C\_USB,

S\_VOIP\_P\_USB,

S\_VOIP\_C\_USB,

S\_FM\_USB,

S\_LOOP\_P\_USB,

S\_LOOP\_C\_USB,

S\_FM\_DSP\_USB,

S\_OFFLOAD\_A2DP,

S\_PCM\_A2DP,

S\_VOICE\_P\_BT,

S\_VOICE\_C\_BT,

S\_VOIP\_P\_BT,

S\_VOIP\_C\_BT,

S\_LOOP\_P\_BT,

S\_LOOP\_C\_BT,

S\_CAPTURE\_BT,

S\_FAST\_P\_BT,

S\_NORMAL\_AP01\_P\_BT,

S\_NORMAL\_AP01\_P\_HIFI,

S\_NORMAL\_AP23\_P\_HIFI,

S\_FAST\_P\_HIFI,

S\_OFFLOAD\_HIFI,

S\_VOICE\_P\_HIFI,

S\_VOIP\_P\_HIFI,

S\_FM\_HIFI,

S\_LOOP\_P\_HIFI,

S\_FM\_DSP\_HIFI,

S\_VOICE\_CAP\_C,

S\_FM\_CAP\_C,

S\_FM\_CAP\_DSP\_C,

S\_BTSCO\_CAP\_DSP\_C,

S\_VBC\_DUMP,

S\_CODEC\_TEST\_C,

S\_CODEC\_TEST\_P,

S\_HFP\_P,

S\_HFP\_C,

S\_CAPTURE\_RECOGNISE\_CODEC,

S\_VOICE\_PCM\_P,

S\_NORMAL\_AP01\_P\_SMTPA,

S\_NORMAL\_AP23\_P\_SMTPA,

S\_FAST\_P\_SMTPA,

S\_OFFLOAD\_SMTPA,

S\_VOICE\_P\_SMTPA,

S\_VOIP\_P\_SMTPA,

S\_FM\_SMTPA,

S\_LOOP\_P\_SMTPA,

S\_FM\_DSP\_SMTPA,

S\_HIFI\_P,

S\_DSP\_HIFI\_FAST\_P,

S\_SWITCH\_CASE\_MAX,

};

static const struct snd\_kcontrol\_new sprd\_audio\_be\_switch[S\_SWITCH\_CASE\_MAX] = {

[S\_NORMAL\_AP01\_P\_CODEC] = SOC\_DAPM\_SINGLE("SWITCH", SND\_SOC\_NOPM, 0,

1, 0),

[S\_NORMAL\_AP01\_P\_BT] = SOC\_DAPM\_SINGLE("SWITCH", SND\_SOC\_NOPM, 0,

1, 0),

[S\_NORMAL\_AP01\_P\_HIFI] = SOC\_DAPM\_SINGLE("SWITCH", SND\_SOC\_NOPM,

0, 1, 0),

[S\_NORMAL\_AP23\_P\_HIFI] = SOC\_DAPM\_SINGLE("SWITCH", SND\_SOC\_NOPM,

0, 1, 0),

[S\_FAST\_P\_HIFI] = SOC\_DAPM\_SINGLE("SWITCH", SND\_SOC\_NOPM, 0, 1, 0),

[S\_OFFLOAD\_HIFI] = SOC\_DAPM\_SINGLE("SWITCH", SND\_SOC\_NOPM, 0, 1, 0),

[S\_VOICE\_P\_HIFI] = SOC\_DAPM\_SINGLE("SWITCH", SND\_SOC\_NOPM, 0, 1, 0),

[S\_VOIP\_P\_HIFI] = SOC\_DAPM\_SINGLE("SWITCH", SND\_SOC\_NOPM, 0, 1, 0),

[S\_FM\_HIFI] = SOC\_DAPM\_SINGLE("SWITCH", SND\_SOC\_NOPM, 0, 1, 0),

[S\_LOOP\_P\_HIFI] = SOC\_DAPM\_SINGLE("SWITCH", SND\_SOC\_NOPM, 0, 1, 0),

[S\_FM\_DSP\_HIFI] = SOC\_DAPM\_SINGLE("SWITCH", SND\_SOC\_NOPM, 0, 1, 0),

[S\_VOICE\_CAP\_C] = SOC\_DAPM\_SINGLE("SWITCH", SND\_SOC\_NOPM, 0,

1, 0),

[S\_FM\_CAP\_C] = SOC\_DAPM\_SINGLE("SWITCH", SND\_SOC\_NOPM, 0,

1, 0),

[S\_FM\_CAP\_DSP\_C] = SOC\_DAPM\_SINGLE("SWITCH", SND\_SOC\_NOPM, 0,

1, 0),

[S\_BTSCO\_CAP\_DSP\_C] = SOC\_DAPM\_SINGLE("SWITCH", SND\_SOC\_NOPM, 0,

1, 0),

[S\_VBC\_DUMP] = SOC\_DAPM\_SINGLE("SWITCH", SND\_SOC\_NOPM, 0,

1, 0),

[S\_CODEC\_TEST\_C] = SOC\_DAPM\_SINGLE("SWITCH", SND\_SOC\_NOPM, 0,

1, 0),

[S\_CODEC\_TEST\_P] = SOC\_DAPM\_SINGLE("SWITCH", SND\_SOC\_NOPM, 0,

1, 0),

[S\_HFP\_P] = SOC\_DAPM\_SINGLE("SWITCH", SND\_SOC\_NOPM, 0,

1, 0),

[S\_HFP\_C] = SOC\_DAPM\_SINGLE("SWITCH", SND\_SOC\_NOPM, 0,

1, 0),

[S\_CAPTURE\_RECOGNISE\_CODEC] = SOC\_DAPM\_SINGLE("SWITCH", SND\_SOC\_NOPM, 0,

1, 0),

[S\_VOICE\_PCM\_P] = SOC\_DAPM\_SINGLE("SWITCH", SND\_SOC\_NOPM, 0,

1, 0),

[S\_NORMAL\_AP01\_P\_SMTPA] = SOC\_DAPM\_SINGLE("SWITCH", SND\_SOC\_NOPM,

0, 1, 0),

[S\_NORMAL\_AP23\_P\_SMTPA] = SOC\_DAPM\_SINGLE("SWITCH", SND\_SOC\_NOPM,

0, 1, 0),

[S\_FAST\_P\_SMTPA] = SOC\_DAPM\_SINGLE("SWITCH", SND\_SOC\_NOPM, 0, 1, 0),

[S\_OFFLOAD\_SMTPA] = SOC\_DAPM\_SINGLE("SWITCH", SND\_SOC\_NOPM, 0, 1, 0),

[S\_VOICE\_P\_SMTPA] = SOC\_DAPM\_SINGLE("SWITCH", SND\_SOC\_NOPM, 0, 1, 0),

[S\_VOIP\_P\_SMTPA] = SOC\_DAPM\_SINGLE("SWITCH", SND\_SOC\_NOPM, 0, 1, 0),

[S\_FM\_SMTPA] = SOC\_DAPM\_SINGLE("SWITCH", SND\_SOC\_NOPM, 0, 1, 0),

[S\_LOOP\_P\_SMTPA] = SOC\_DAPM\_SINGLE("SWITCH", SND\_SOC\_NOPM, 0, 1, 0),

[S\_FM\_DSP\_SMTPA] = SOC\_DAPM\_SINGLE("SWITCH", SND\_SOC\_NOPM, 0, 1, 0),

[S\_HIFI\_P] = SOC\_DAPM\_SINGLE("SWITCH", SND\_SOC\_NOPM, 0, 1, 0),

[S\_DSP\_HIFI\_FAST\_P] = SOC\_DAPM\_SINGLE("SWITCH", SND\_SOC\_NOPM, 0, 1, 0),

};

static const struct snd\_soc\_dapm\_widget sprd\_pcm\_routing\_widgets[] = {

/\* Frontend AIF \*/

SND\_SOC\_DAPM\_AIF\_IN("FE\_IF\_NORMAL\_AP01\_P", "FE\_DAI\_NORMAL\_AP01\_P",

0, 0, 0, 0),

SND\_SOC\_DAPM\_AIF\_OUT("FE\_IF\_NORMAL\_AP01\_C", "FE\_DAI\_NORMAL\_AP01\_C",

0, 0, 0, 0),

SND\_SOC\_DAPM\_AIF\_IN("FE\_IF\_NORMAL\_AP23\_P", "FE\_DAI\_NORMAL\_AP23\_P",

0, 0, 0, 0),

SND\_SOC\_DAPM\_AIF\_OUT("FE\_IF\_NORMAL\_AP23\_C", "FE\_DAI\_NORMAL\_AP23\_C",

0, 0, 0, 0),

SND\_SOC\_DAPM\_AIF\_OUT("FE\_IF\_CAP\_DSP\_C", "FE\_DAI\_CAP\_DSP\_C", 0, 0, 0, 0),

SND\_SOC\_DAPM\_AIF\_IN("FE\_IF\_FAST\_P", "FE\_DAI\_FAST\_P", 0, 0, 0, 0),

SND\_SOC\_DAPM\_AIF\_IN("FE\_IF\_HIFI\_FAST\_P", "FE\_DAI\_ID\_HIFI\_FAST\_P", 0, 0, 0, 0),

SND\_SOC\_DAPM\_AIF\_IN("FE\_IF\_OFFLOAD\_P", "FE\_DAI\_OFFLOAD\_P", 0, 0, 0, 0),

SND\_SOC\_DAPM\_AIF\_OUT("FE\_IF\_DUMP\_C", "FE\_DAI\_DUMP\_C", 0, 0, 0, 0),

SND\_SOC\_DAPM\_AIF\_OUT("FE\_IF\_BTCAP\_AP\_C", "FE\_DAI\_BTCAP\_AP\_C",

0, 0, 0, 0),

SND\_SOC\_DAPM\_AIF\_IN("FE\_IF\_HFP\_P", "FE\_DAI\_HFP\_P", 0, 0, 0, 0),

SND\_SOC\_DAPM\_AIF\_OUT("FE\_IF\_HFP\_C", "FE\_DAI\_HFP\_C", 0, 0, 0, 0),

SND\_SOC\_DAPM\_AIF\_OUT("FE\_IF\_RECOGNISE\_CAP\_C", "FE\_DAI\_RECOGNISE\_CAP\_C",

0, 0, 0, 0),

SND\_SOC\_DAPM\_AIF\_IN("FE\_IF\_VOICE\_PCM\_P", "FE\_DAI\_VOICE\_PCM\_P",

0, 0, 0, 0),

SND\_SOC\_DAPM\_AIF\_IN("FE\_IF\_HIFI\_P", "FE\_DAI\_HIFI\_P", 0, 0, 0, 0),

/\* Backend AIF \*/

SND\_SOC\_DAPM\_AIF\_IN("BE\_IF\_NORMAL\_AP01\_CODEC\_P",

"BE\_DAI\_NORMAL\_AP01\_CODEC\_P", 0, 0, 0, 0),

SND\_SOC\_DAPM\_AIF\_IN("BE\_IF\_NORMAL\_AP01\_BTSCO\_P",

"BE\_DAI\_NORMAL\_AP01\_BTSCO\_P", 0, 0, 0, 0),

SND\_SOC\_DAPM\_AIF\_IN("BE\_IF\_NORMAL\_AP01\_HIFI\_P",

"BE\_DAI\_ID\_NORMAL\_AP01\_P\_HIFI",

0, 0, 0, 0),

SND\_SOC\_DAPM\_AIF\_IN("BE\_IF\_ID\_NORMAL\_AP23\_HIFI",

"BE\_DAI\_ID\_NORMAL\_AP23\_HIFI",

0, 0, 0, 0),

SND\_SOC\_DAPM\_AIF\_IN("BE\_IF\_ID\_FAST\_P\_HIFI", "BE\_DAI\_ID\_FAST\_P\_HIFI",

0, 0, 0, 0),

SND\_SOC\_DAPM\_AIF\_IN("BE\_IF\_ID\_OFFLOAD\_HIFI", "BE\_DAI\_ID\_OFFLOAD\_HIFI",

0, 0, 0, 0),

SND\_SOC\_DAPM\_AIF\_IN("BE\_IF\_ID\_VOICE\_HIFI", "BE\_DAI\_ID\_VOICE\_HIFI",

0, 0, 0, 0),

SND\_SOC\_DAPM\_AIF\_IN("BE\_IF\_ID\_VOIP\_HIFI", "BE\_DAI\_ID\_VOIP\_HIFI",

0, 0, 0, 0),

SND\_SOC\_DAPM\_AIF\_IN("BE\_IF\_ID\_FM\_HIFI", "BE\_DAI\_ID\_FM\_HIFI",

0, 0, 0, 0),

SND\_SOC\_DAPM\_AIF\_IN("BE\_IF\_ID\_LOOP\_HIFI", "BE\_DAI\_ID\_LOOP\_HIFI",

0, 0, 0, 0),

SND\_SOC\_DAPM\_AIF\_IN("BE\_IF\_ID\_FM\_DSP\_HIFI", "BE\_DAI\_ID\_FM\_DSP\_HIFI",

0, 0, 0, 0),

SND\_SOC\_DAPM\_AIF\_OUT("BE\_IF\_VOICE\_CAP\_C", "BE\_DAI\_VOICE\_CAP\_C",

0, 0, 0, 0),

SND\_SOC\_DAPM\_AIF\_OUT("BE\_IF\_CAP\_FM\_CAP\_C", "BE\_DAI\_FM\_CAP\_C",

0, 0, 0, 0),

SND\_SOC\_DAPM\_AIF\_OUT("BE\_IF\_CAP\_DSP\_FM\_C", "BE\_DAI\_CAP\_DSP\_FM\_C",

0, 0, 0, 0),

SND\_SOC\_DAPM\_AIF\_OUT("BE\_IF\_CAP\_DSP\_BTSCO\_C", "BE\_DAI\_CAP\_DSP\_BTSCO\_C",

0, 0, 0, 0),

SND\_SOC\_DAPM\_AIF\_OUT("BE\_IF\_DUMP\_C", "BE\_DAI\_DUMP\_C", 0, 0, 0, 0),

SND\_SOC\_DAPM\_AIF\_IN("BE\_IF\_HFP\_P", "BE\_DAI\_HFP\_P", 0, 0, 0, 0),

SND\_SOC\_DAPM\_AIF\_OUT("BE\_IF\_HFP\_C", "BE\_DAI\_HFP\_C", 0, 0, 0, 0),

SND\_SOC\_DAPM\_AIF\_OUT("BE\_IF\_CAP\_RECOGNISE\_CODEC\_C",

"BE\_DAI\_CAP\_RECOGNISE\_CODEC\_C", 0, 0, 0, 0),

SND\_SOC\_DAPM\_AIF\_IN("BE\_IF\_VOICE\_PCM\_P", "BE\_DAI\_VOICE\_PCM\_P",

0, 0, 0, 0),

SND\_SOC\_DAPM\_AIF\_IN("BE\_IF\_NORMAL\_AP01\_SMTPA\_P",

"BE\_DAI\_ID\_NORMAL\_AP01\_P\_SMTPA",

0, 0, 0, 0),

SND\_SOC\_DAPM\_SWITCH("S\_NORMAL\_AP01\_P\_BT", SND\_SOC\_NOPM,

0, 0, &sprd\_audio\_be\_switch[S\_NORMAL\_AP01\_P\_BT]),

SND\_SOC\_DAPM\_SWITCH\_E("S\_NORMAL\_AP01\_P\_HIFI", SND\_SOC\_NOPM, 0, 0,

&sprd\_audio\_be\_switch[S\_NORMAL\_AP01\_P\_HIFI],

be\_switch\_evt,

SND\_SOC\_DAPM\_PRE\_PMU | SND\_SOC\_DAPM\_POST\_PMD),

SND\_SOC\_DAPM\_SWITCH\_E("S\_NORMAL\_AP23\_P\_HIFI", SND\_SOC\_NOPM, 0, 0,

&sprd\_audio\_be\_switch[S\_NORMAL\_AP23\_P\_HIFI],

be\_switch\_evt,

SND\_SOC\_DAPM\_PRE\_PMU | SND\_SOC\_DAPM\_POST\_PMD),

SND\_SOC\_DAPM\_SWITCH\_E("S\_FAST\_P\_HIFI", SND\_SOC\_NOPM, 0, 0,

&sprd\_audio\_be\_switch[S\_FAST\_P\_HIFI],

be\_switch\_evt,

SND\_SOC\_DAPM\_PRE\_PMU | SND\_SOC\_DAPM\_POST\_PMD),

SND\_SOC\_DAPM\_SWITCH\_E("S\_OFFLOAD\_HIFI", SND\_SOC\_NOPM, 0, 0,

&sprd\_audio\_be\_switch[S\_OFFLOAD\_HIFI],

be\_switch\_evt,

SND\_SOC\_DAPM\_PRE\_PMU | SND\_SOC\_DAPM\_POST\_PMD),

SND\_SOC\_DAPM\_SWITCH\_E("S\_VOICE\_P\_HIFI", SND\_SOC\_NOPM, 0, 0,

&sprd\_audio\_be\_switch[S\_VOICE\_P\_HIFI],

be\_switch\_evt,

SND\_SOC\_DAPM\_PRE\_PMU | SND\_SOC\_DAPM\_POST\_PMD),

SND\_SOC\_DAPM\_SWITCH\_E("S\_VOIP\_P\_HIFI", SND\_SOC\_NOPM, 0, 0,

&sprd\_audio\_be\_switch[S\_VOIP\_P\_HIFI],

be\_switch\_evt,

SND\_SOC\_DAPM\_PRE\_PMU | SND\_SOC\_DAPM\_POST\_PMD),

SND\_SOC\_DAPM\_SWITCH\_E("S\_FM\_HIFI", SND\_SOC\_NOPM, 0, 0,

&sprd\_audio\_be\_switch[S\_FM\_HIFI],

be\_switch\_evt,

SND\_SOC\_DAPM\_PRE\_PMU | SND\_SOC\_DAPM\_POST\_PMD),

SND\_SOC\_DAPM\_SWITCH\_E("S\_LOOP\_P\_HIFI", SND\_SOC\_NOPM, 0, 0,

&sprd\_audio\_be\_switch[S\_LOOP\_P\_HIFI],

be\_switch\_evt,

SND\_SOC\_DAPM\_PRE\_PMU | SND\_SOC\_DAPM\_POST\_PMD),

SND\_SOC\_DAPM\_SWITCH\_E("S\_FM\_DSP\_HIFI", SND\_SOC\_NOPM, 0, 0,

&sprd\_audio\_be\_switch[S\_FM\_DSP\_HIFI],

be\_switch\_evt,

SND\_SOC\_DAPM\_PRE\_PMU | SND\_SOC\_DAPM\_POST\_PMD),

SND\_SOC\_DAPM\_SWITCH("S\_VOICE\_CAP\_C", SND\_SOC\_NOPM,

0, 0, &sprd\_audio\_be\_switch[S\_VOICE\_CAP\_C]),

SND\_SOC\_DAPM\_SWITCH("S\_FM\_CAP\_C", SND\_SOC\_NOPM,

0, 0, &sprd\_audio\_be\_switch[S\_FM\_CAP\_C]),

SND\_SOC\_DAPM\_SWITCH("S\_FM\_CAP\_DSP\_C", SND\_SOC\_NOPM,

0, 0, &sprd\_audio\_be\_switch[S\_FM\_CAP\_DSP\_C]),

SND\_SOC\_DAPM\_SWITCH("S\_BTSCO\_CAP\_DSP\_C", SND\_SOC\_NOPM,

0, 0, &sprd\_audio\_be\_switch[S\_BTSCO\_CAP\_DSP\_C]),

SND\_SOC\_DAPM\_SWITCH("S\_VBC\_DUMP", SND\_SOC\_NOPM,

0, 0, &sprd\_audio\_be\_switch[S\_VBC\_DUMP]),

SND\_SOC\_DAPM\_SWITCH("S\_CODEC\_TEST\_C", SND\_SOC\_NOPM,

0, 0, &sprd\_audio\_be\_switch[S\_CODEC\_TEST\_C]),

SND\_SOC\_DAPM\_SWITCH("S\_CODEC\_TEST\_P", SND\_SOC\_NOPM,

0, 0, &sprd\_audio\_be\_switch[S\_CODEC\_TEST\_P]),

SND\_SOC\_DAPM\_SWITCH("S\_HFP\_P", SND\_SOC\_NOPM,

0, 0, &sprd\_audio\_be\_switch[S\_HFP\_P]),

SND\_SOC\_DAPM\_SWITCH("S\_HFP\_C", SND\_SOC\_NOPM,

0, 0, &sprd\_audio\_be\_switch[S\_HFP\_C]),

SND\_SOC\_DAPM\_SWITCH("S\_CAPTURE\_RECOGNISE\_CODEC", SND\_SOC\_NOPM,

0, 0, &sprd\_audio\_be\_switch[S\_CAPTURE\_RECOGNISE\_CODEC]),

SND\_SOC\_DAPM\_SWITCH("S\_VOICE\_PCM\_P", SND\_SOC\_NOPM,

0, 0, &sprd\_audio\_be\_switch[S\_VOICE\_PCM\_P]),

SND\_SOC\_DAPM\_SWITCH("S\_DSP\_HIFI\_FAST\_P", SND\_SOC\_NOPM,

0, 0, &sprd\_audio\_be\_switch[S\_DSP\_HIFI\_FAST\_P]),

};

**sound\soc\sprd\platform\sprd-platform-pcm-routing.h**

**/\* FE dai id\*/**

**enum {**

**FE\_DAI\_ID\_NORMAL\_AP01 = 0,**

**FE\_DAI\_ID\_NORMAL\_AP23,**

**FE\_DAI\_ID\_CAPTURE\_DSP,**

**FE\_DAI\_ID\_FAST\_P,**

**FE\_DAI\_ID\_OFFLOAD,**

**FE\_DAI\_ID\_VOICE,**

**FE\_DAI\_ID\_VOIP,**

**FE\_DAI\_ID\_FM,**

**FE\_DAI\_ID\_FM\_CAPTURE\_AP,**

**FE\_DAI\_ID\_VOICE\_CAPTURE,**

**FE\_DAI\_ID\_LOOP,**

**FE\_DAI\_ID\_A2DP\_OFFLOAD,**

**FE\_DAI\_ID\_A2DP\_PCM,**

**FE\_DAI\_ID\_FM\_CAP\_DSP,**

**FE\_DAI\_ID\_BTSCO\_CAP\_DSP,**

**FE\_DAI\_ID\_FM\_DSP,**

**FE\_DAI\_ID\_DUMP,**

**FE\_DAI\_ID\_BTSCO\_CAP\_AP,**

**FE\_DAI\_ID\_VOICE\_PCM\_P,**

**FE\_DAI\_ID\_CODEC\_TEST,**

**FE\_DAI\_ID\_HFP,**

**FE\_DAI\_ID\_RECOGNISE\_CAPTURE,**

**FE\_DAI\_ID\_HIFI\_P,**

**FE\_DAI\_ID\_HIFI\_FAST\_P,**

**FE\_DAI\_ID\_MAX**

**};**

**/\* BE dais id\*/**

**enum {**

**/\* codec \*/**

**BE\_DAI\_ID\_NORMAL\_AP01\_CODEC = 0,**

**BE\_DAI\_ID\_NORMAL\_AP23\_CODEC,**

**BE\_DAI\_ID\_CAPTURE\_DSP\_CODEC,**

**BE\_DAI\_ID\_FAST\_P\_CODEC,**

**BE\_DAI\_ID\_OFFLOAD\_CODEC,**

**BE\_DAI\_ID\_VOICE\_CODEC,**

**BE\_DAI\_ID\_VOIP\_CODEC,**

**BE\_DAI\_ID\_FM\_CODEC,**

**BE\_DAI\_ID\_LOOP\_CODEC,**

**BE\_DAI\_ID\_FM\_DSP\_CODEC,**

**/\* usb \*/**

**BE\_DAI\_ID\_NORMAL\_AP01\_USB,**

**BE\_DAI\_ID\_NORMAL\_AP23\_USB,**

**BE\_DAI\_ID\_CAPTURE\_DSP\_USB,**

**BE\_DAI\_ID\_FAST\_P\_USB,**

**BE\_DAI\_ID\_OFFLOAD\_USB,**

**BE\_DAI\_ID\_VOICE\_USB,**

**BE\_DAI\_ID\_VOIP\_USB,**

**BE\_DAI\_ID\_FM\_USB,**

**BE\_DAI\_ID\_LOOP\_USB,**

**BE\_DAI\_ID\_FM\_DSP\_USB,**

**/\* bt \*/**

**BE\_DAI\_ID\_OFFLOAD\_A2DP,**

**BE\_DAI\_ID\_PCM\_A2DP,**

**BE\_DAI\_ID\_VOICE\_BT,**

**BE\_DAI\_ID\_VOIP\_BT,**

**BE\_DAI\_ID\_LOOP\_BT,**

**BE\_DAI\_ID\_CAPTURE\_BT,**

**BE\_DAI\_ID\_CAPTURE\_DSP\_BTSCO,**

**BE\_DAI\_ID\_FAST\_P\_BTSCO,**

**BE\_DAI\_ID\_NORMAL\_AP01\_P\_BTSCO,**

**/\* hifi \*/**

**BE\_DAI\_ID\_NORMAL\_AP01\_P\_HIFI,**

**BE\_DAI\_ID\_NORMAL\_AP23\_HIFI,**

**BE\_DAI\_ID\_FAST\_P\_HIFI,**

**BE\_DAI\_ID\_OFFLOAD\_HIFI,**

**BE\_DAI\_ID\_VOICE\_HIFI,**

**BE\_DAI\_ID\_VOIP\_HIFI,**

**BE\_DAI\_ID\_FM\_HIFI,**

**BE\_DAI\_ID\_LOOP\_HIFI,**

**BE\_DAI\_ID\_FM\_DSP\_HIFI,**

**/\* common \*/**

**BE\_DAI\_ID\_VOICE\_CAPTURE,**

**BE\_DAI\_ID\_FM\_CAPTURE,**

**BE\_DAI\_ID\_FM\_CAPTURE\_DSP,**

**BE\_DAI\_ID\_DUMP,**

**BE\_DAI\_ID\_DUMMY\_VBC\_DAI\_NOTBE,**

**BE\_DAI\_ID\_HFP,**

**BE\_DAI\_ID\_RECOGNISE\_CAPTURE,**

**BE\_DAI\_ID\_VOICE\_PCM\_P,**

**/\* smart pa \*/**

**BE\_DAI\_ID\_NORMAL\_AP01\_P\_SMTPA,**

**BE\_DAI\_ID\_NORMAL\_AP23\_SMTPA,**

**BE\_DAI\_ID\_FAST\_P\_SMTPA,**

**BE\_DAI\_ID\_OFFLOAD\_SMTPA,**

**BE\_DAI\_ID\_VOICE\_SMTPA,**

**BE\_DAI\_ID\_VOIP\_SMTPA,**

**BE\_DAI\_ID\_FM\_SMTPA,**

**BE\_DAI\_ID\_LOOP\_SMTPA,**

**BE\_DAI\_ID\_FM\_DSP\_SMTPA,**

**BE\_DAI\_ID\_HIFI\_P,**

**BE\_DAI\_ID\_HIFI\_FAST\_P,**

**BE\_DAI\_ID\_MAX**

**};**

1:ActivityManager: START u0 {cmp=com.sprd.validationtools/.itemstest.PhoneLoopBackTest (has extras)} from uid 1000, pid 5346

2:am\_create\_activity: [0,178983590,48,com.sprd.validationtools/.itemstest.PhoneLoopBackTest,NULL,NULL,NULL,0]

3:PowerController: packageName:com.sprd.validationtools state:MOVE\_TO\_BACKGROUND user:0

4:ActivityManager: ->startActivity for ActivityRecord{aab12a6 u0 com.sprd.validationtools/.itemstest.PhoneLoopBackTest t48} result:START\_SUCCESS

5:am\_set\_resumed\_activity: [0,com.sprd.validationtools/.itemstest.PhoneLoopBackTest,minimalResumeActivityLocked]

6:am\_set\_resumed\_activity: [0,com.sprd.validationtools/.itemstest.PhoneLoopBackTest,minimalResumeActivityLocked]

7:PhoneLoopBackTest: onCreate mAuxDeviceSupport=true

8:validation\_jni: get\_audio\_whale\_loopback\_flag...AUDIO\_WHALE\_LOOPBACK=:1

9:PhoneLoopBackTest: === create thread to execute PhoneLoopBackTest start command! ===

10:am\_on\_resume\_called: [0,com.sprd.validationtools.itemstest.PhoneLoopBackTest,RESUME\_ACTIVITY]

Modem 开启语音通话的在hal层执行:

Voice\_call.c (g:\code-pj\source\t710\vendor\modules\audio\whale)

pcm\_start(voice\_out->pcm\_modem\_ul)---->上行

pcm\_start(voice\_out->pcm\_modem\_dl)---->下行

对应到内核的执行:

Sprd-dmaengine-pcm.c (sound\soc\sprd\platform)

sprd\_pcm\_trigger//实际是启动DMA传输

--->case SNDRV\_PCM\_TRIGGER\_START:

--->rtd->cookie[i] = dmaengine\_submit(

rtd->dma\_tx\_des[i]);

dev\_laohua\_test\_out\_thread

/\* config mixer \*/

|dsp\_sleep\_ctrl(adev->dev\_ctl->agdsp\_ctl,true);

|set\_usecase(adev->dev\_ctl, UC\_NORMAL\_PLAYBACK, true);

|set\_usecase(adev->dev\_ctl, UC\_NORMAL\_PLAYBACK, true);

|switch\_vbc\_route(adev->dev\_ctl,out\_test\_info->devices);

|select\_devices\_new(adev->dev\_ctl,UC\_NORMAL\_PLAYBACK, out\_test\_info->devices,false,true,true, true);

|/\* play \*/

|out\_test\_info->config.format = PCM\_FORMAT\_S16\_LE;

|out\_test\_info->config.period\_size = 1024 ;

|out\_test\_info->config.period\_count = 4 ;

|out\_test\_info->pcm = pcm\_open(0, 0, PCM\_OUT | PCM\_MMAP | PCM\_NOIRQ |PCM\_MONOTONIC, &out\_test\_info->config);