聯陽半導體
ITE Tech. Inc.

iTE SDK

Streaming 架構說明文件

V0.9

ITE TECH. INC.

修訂記錄

修訂日期	修訂說明	頁次
2015/05/22	初建版本 V0.9	

目錄

1. 前言

IT9079 目前提供支援 H.264 baseline profile (1280 x 720) 的 hardware decoder。在 software SDK 內，我們提供了一套架構於 linphone (http://www.linphone.org/) 和 mediastreamer2(http://www.linphone.org/technical-corner/mediastreamer2/overview)之上特別針對 SIP 以及 RTP 做處理的 player。底下描述這套 player 的架構，以及 hardware video decoder 在其中扮演的角色。

1.1 編寫目的

讓相關開發人員快速理解 streaming 技術

1.2 適用人員

軟體應用程式、驅動程式開發者。

2. Streaming

SDK 所提供的 streaming 是架構於 linphone 和 mediastreamer2 之上特別針對 SIP 以及 RTP 做處理的 player。linphone 和 mediastreamer2 都是開源的程式碼, 在網路上都可找到相關的資料. linphone 是一个轻量级 voip 客户端，linphone 的架构设计十分的清晰，其底层音视频引擎 mediastream2 是一个独立的模块，基于它可以很容易的实现各种音视频的应用。

2.1 linphone 整體架構介紹

linphone 拥有自己的用户接口和核心引擎(音频/视频引擎)，允许在相同的函数基础上建立不同的用户接口。

- 用户接口
  命令行接口(linphonec, linphonecsh)
- liblinphone，核心引擎：提供了高度容易扩展的函数接口，它是一个功能强大的 SIP VoIP video SDK，我们可以基于该框架实现任意的语音/视频应用。
  liblinphone 基于下列组件实现：
  mediastream2，功能强大的音/视多媒体 SDK，用于创建和处理音视频流
  oRTP,简单的 RTP 处理库。
  eXosip，SIP UA 库，其基于 libosip2 实现。

linphone 和 liblinphone 都是使用 C 语言实现的，詳細資料請參照官網：http://www.linphone.org/eng/documentation/dev/

2.2 Mediastreamer2 用法介紹

mediastreamer2 是一个功能强大的，轻量级流处理引擎，特别适合于音视频电话应用方面的开发。它能够完成音视频流的捕获、接收、发送、编码、解码、渲染等功能。

特性：

1. 可以捕获、重放多种基于各种平台音频架构(ALSA, AudioUnits, AudioQueue, WaveApi)。
2. 接收、发送 RTP 数据包。
3. 支持如下的音视频编解码格式：音频 **speex, G711, GSM, iLBC, AMR, AMR-WB, G722, SILK, G729；** 视频： **H263, theora, MPEG4, H264 and VP8.**
4. 支持音频的 wav 格式文件的读取和保存。
5. 捕获摄像头的 YUV 图片格式(基于平台独立的 API)。
6. 优化，渲染 YUV 图片。
7. 立体声输出。
8. 自定义声音侦测。
9. 回声消除，基于 speex 库 和 Android 上的 webrtc AEC 机制。
10. 支持音频会议。
11. 支持音频参数均衡。
12. 音量控制，自动增益控制。
13. 具有 IEC NAT 穿越动能。

14. 自适应比特率控制算法：根据接收的 RTCP 反馈信息自动适应编码速率。
mediastreamer2 以扩展插件的形式支持 H264, ILBC, SILK, AMR, AMR-WB and G729。

设计和原理：
mediastreamer2 中每一个处理实体都包含一个 MSFilter 结构，每一个 MSFiter 有一个或者若干个输入和输出，通过这些输入和输出可以将各个 MSFilter 连接起来。
下面为一个简单的例子：

1. MSRtpRecv：接收网络上的数据包，解包后将它们输出到下一个 MSFilter。
2. MSSpeexDec：接收输入的音频数据包(假设音频用 speex 编码)，解码并输出到下一个 MSFilter。
3. MSFileRec:接收输入的语音数据并保存为 wav 格式文件(假设输入为 16bit 线性 PCM)。

MSFilter 可以被串接成一个 filter 链，我们接收网络上的 RTP 包，然后解码并保存成一个 wav 文件。
MSRtpRecv --> MSSpeexDec --> MSFileRec

媒体处理中的调度对象为 MSTicker，它一个独立的线程，其每 10ms 被唤醒一次，然后它会处理他所管理的媒体链的数据。几个 MSTicker 可以同时运行，例如，一个负责处理音频，一个负责处理视频，或在不同的处理器中运行不同的 MSTicker。

以上是 Linphone 官方网站上的介绍，下面我们实现一个本地录音机实例来具体说明 MSFilter 到底是如何工作的。

本地录音机的实现

1.Filter 的分类和定义：
从源码目录我们可以看到 Filter 大概分为以下几种，音频，视频和其他。

音频 Filter：

```
├─── aac-eld.c
├─── alaw.c          ITU-G.711 alaw encoder & decoder
├─── ulaw.c          ITU-G.711 ulaw encoder & decoder
├─── l16.c           L16 dummy encoder & decoder
├─── msg722.c        G.722 wideband encoder & decoder
├─── msopus.c        opus encoder & decoder
├─── msspeex.c       speex encoder & decoder
├─── gsm.c           GSM encoder & decoder
├─── oss.c           Sound playback & playback filter for OSS drivers
├─── alsa.c          Sound playback & playback filter ALSA Driver
├─── macsnd.c        Sound playback & playback filter for OSX Audio Unit
├─── arts.c          Sound playback & playback filter aRts Driver
├─── pasnd.c         Sound playback & playback filter Port Audio
├─── winsnd.c        Sound playback & playback for Windows Sound drivers
├─── aqsnd.c         Sound playback & playback for OSX AudioQueueService
├─── pulseaudio.c    Sound input & output plugin based on PulseAudio
```

├─── audiomixer.c A filter that mixes down 16 bit sample audio streams
├─── chanadapt.c A filter that converts from mono to stereo and vice versa
├─── dtmfgen.c DTMF generator
├─── equalizer.c Parametric sound equalizer
├─── genericplc.c Generic PLC
├─── msconf.c A filter to make conferencing
├─── msfileplayer.c Raw files and wav reader
├─── msfilerec.c Wav file recorder
├─── msresample.c Audio resampler
├─── msvolume.c A filter that controls and measure sound volume
├─── tonedetector.c Custom tone detection filter
├─── speexec.c Echo canceller using speex library(回声消除)
├─── webrtc_aec.c Echo canceller using WebRTC library(回声消除)

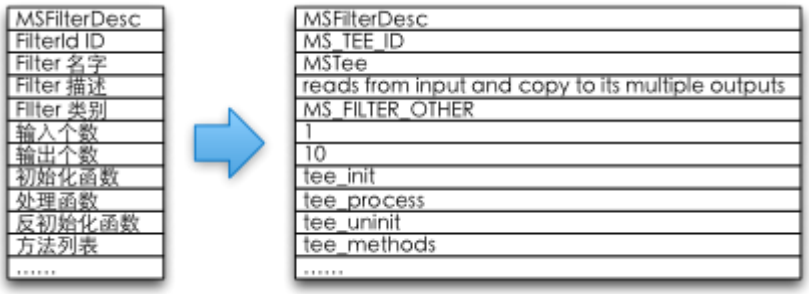
视频 Filter：
├─── h264dec.c H264 decoder
├─── videodec.c H.263 ，MPEG4 ，MJEPG ，snow decoder
├─── videoenc.c H.263 ，MPEG4 ，MJEPG ，snow encoder
├─── theora.c theora encoder & decoder
├─── vp8.c VP8 encoder & decoder
├─── drawdib-display.c A video display based on windows DrawDib api
├─── extdisplay.c A display filter sending the buffers to draw to the upper layer
├─── x11video.c A video display using X11+Xv
├─── glxvideo.c A video display using GL (glx)
├─── videoout.c A SDL-based video display
├─── jpegwriter.c Take a video snapshot as jpg file
├─── mire.c outputs synthetic moving picture
├─── nowebcam.c A filter that outputs a static image.
├─── pixconv.c A pixel format converter
├─── sizeconv.c a small video size converter
├─── wincevideods.c A video4windows compatible source filter to stream pictures
├─── msv4l2.c A filter to grab pictures from Video4Linux2-powered cameras
├─── msv4l.c A video4linux compatible source filter to stream pictures
├─── winvideo2.c A video for windows (vfw.h) based source filter to grab pictures
├─── winvideo.c A video4windows compatible source filter to stream pictures
├─── winvideods.c A video4windows compatible source filter to stream pictures

其他 Filter：
├─── msrtp.c RTP input & ouput
├─── tee.c Reads from input and copy to its multiple outputs
├─── join.c Send several inputs to one output
├─── itc.c Inter ticker communication

└──── void.c　　Trashes its input (useful for terminating some graphs)

核心组件
├──── eventqueue.c　事件队列
├──── mscommon.c　初始化等
├──── msfilter.c　Filter 定义
├──── msqueue.c　队列结构
├──── mssndcard.c　声卡处理
├──── msticker.c　定时器
├──── mswebcam.c　摄像头处理
└──── mtu.c　MTU

每个 Filter 都定义有一个描述信息，所有 Filter 的描述在 ms_base_filter_descs[]和 ms_voip_filter_descs[]两个数组中定义，下面以 Tee Filter 为例说明，后面我们会用到。



我们可以根据 ID 或者描述构造一个 Filter 对象，如：
MSFilter *tee=ms_filter_new(MS_TEE_ID)；

要回收此对象，如：
ms_filter_destroy(tee);

或者调用此对象的某种方法，如：
ms_filter_call_method(tee,MS_TEE_UNMUTE,&pin);

方法都是在上面 tee_methods 数组中定义的,如 Tee 只有两种方法
```
static MSFilterMethod tee_methods[]={
    {   MS_TEE_MUTE  ,    tee_mute    },
    {   MS_TEE_UNMUTE   ,tee_unmute     },
    {   0    ,    NULL          }
};
```

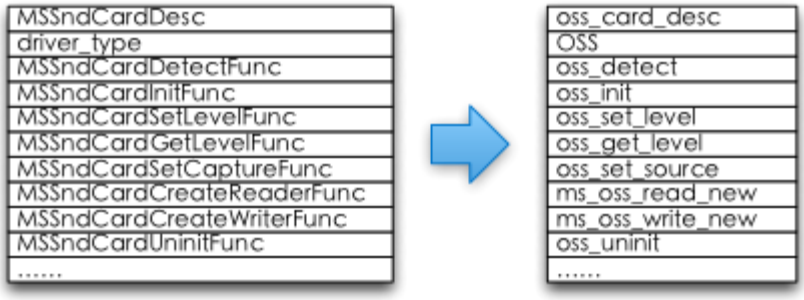
tee_init 在构造对象时执行一次，tee_uninit 在回收时执行一次，tee_process 在每次调度时执行。

2.声卡的管理和使用

声卡的处理在核心组件 mssndcard.c 中统一管理和使用。
在文件中的开头定义了声卡的全局管理句柄，在库初始化时会把所有可用的声卡挂载到这个链表上。

```
static MSSndCardManager *scm=NULL；
struct _MSSndCardManager{
        MSList *cards;
        MSList *descs;
};
```

和 Filter 类似，每种声卡也有一个描述信息，所有声卡的描述在 ms_snd_card_descs[] 数组中定义，相当一个容器：



ms_init() 函数在初始化库的时候依次注册 ms_snd_card_manager_register_desc 列表中支持的驱动，执行相应的 etect 函数侦测声卡，然后构造声卡对象并调用对象的 Init 函数；

当我们需要一个采集或者回放的 Filter 时，可以直接调用：

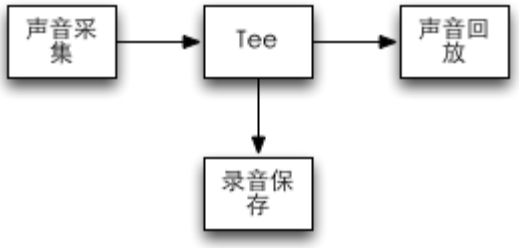
```
MSSndCard                                                    *card_capture
=ms_snd_card_manager_get_default_capture_card(ms_snd_card_manager_get());
```

ms_snd_card_manager_get() 返回一个全局的声卡管理句柄，
ms_snd_card_manager_get_default_capture_card() 扫描全局的管理句柄中注册的声卡设备，并返回第一个可用的声卡对象。

继续调用声卡对象的 CreateReader 方法，可以得到的声音采集的 MSFilte 对象。

```
MSFilter *filte_capture=ms_snd_card_create_reader(card_capture)；
```

3.设计 Filter 链
现在我们来设计我们的 Filter 链，有些文档叫 Graph.

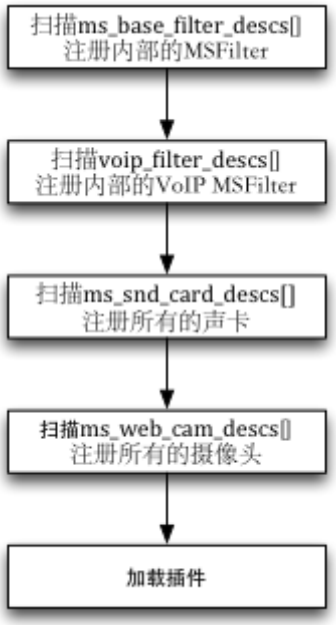


总共需要 4 个 Filter，采集，复制，保存和回放，Tee Filter 在上文已经简单了解过了，他可以把一份数据拷贝到多出。

要使用 Filter 要先进行 mediastream2 和 oRTP 的初始化，mediastream2 库依赖于 oRTP 库，先初始化 oRTP 库:
ortp_init();

然后初始化 mediastream2 库：
ms_init();

我们先简单看一下，他的初始化都大概做了些什么？



ms_base_filter_descs[]和 ms_voip_filter_descs[]数组上面提到过，定义了所有的 Filter，ms_snd_card_descs[]上面也提到过，定义了所有的声卡驱动，此处就是把所有的定义注册到系统中，摄像头的管理和使用方式其实和声卡是类似的，在后面的视频流实例中我们会用到。最后是加载插件，像我们后面要用到的 H.264 编码器，在程序中是没有的，前面提到过，是以插件的形式提供的，其实就是动态库的形式。所有的东西都注册完毕，我们

就可以创建我们要用的 Filter 了。

1.创建 Filter

```
MSSndCard                                                    *card_capture=
ms_snd_card_manager_get_default_capture_card(ms_snd_card_manager_get())；
MSSndCard                                                   *card_playback=
ms_snd_card_manager_get_default_playback_card(ms_snd_card_manager_get())；

MSFilter *capture=ms_snd_card_create_reader(card_capture);
MSFilter *playback=ms_snd_card_create_writer(card_playback);
MSFilter *tee=ms_filter_new(MS_TEE_ID);
MSFilter *rec=ms_filter_new(MS_FILE_REC_ID);
```

2.设置 Filter

```
int rate=48000;
ms_filter_call_method (capture, MS_FILTER_SET_SAMPLE_RATE,&rate);
ms_filter_call_method (playback, MS_FILTER_SET_SAMPLE_RATE,&rate);
ms_filter_call_method (rec,MS_FILTER_SET_SAMPLE_RATE, &rate);
ms_filter_call_method(rec,MS_FILE_REC_OPEN,"test.wav");
ms_filter_call_method_noarg(rec,MS_FILE_REC_START);
```

3.连接 Filter

```
ms_filter_link(capture,0,tee,0);
ms_filter_link(tee,0, playback,0);
ms_filter_link(tee,1,rec,0);
```

4.开始调度

```
MSTicker *ticker=ms_ticker_new();
ms_ticker_attach(ticker, capture);
```

此后每隔 10ms，就会对 Filter 处理一次。

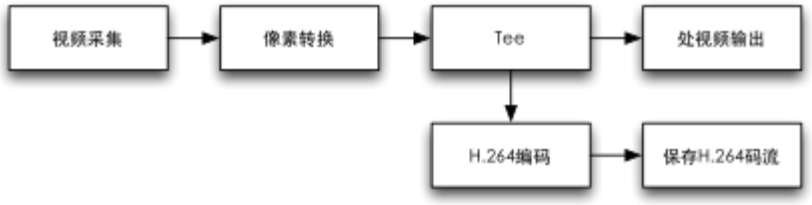
5.停止调度，解除连接并资源回收

```
ms_filter_call_method_noarg(rec,MS_FILE_REC_CLOSE); //很重要，重写 wav 文件头
ms_ticker_detach(ticker, capture);
ms_ticker_destroy(ticker);
ms_filter_unlink(capture,0,tee,0);
ms_filter_unlink(tee,0, playback,0);
ms_filter_unlink(tee,1,rec,0);
ms_filter_destroy(capture);
ms_filter_destroy(playback);
ms_filter_destroy(tee);
ms_filter_destroy(rec);
```

到此，一个有本地录音回放的小程序就实现了。下面用一个视频回放并录制 h.264 裸码流的实例来说明 mediastream2 中对视频流是如何处理的。

本地录像机的实现

1.和本地录音机一样，还是先设计我们的 Graph。



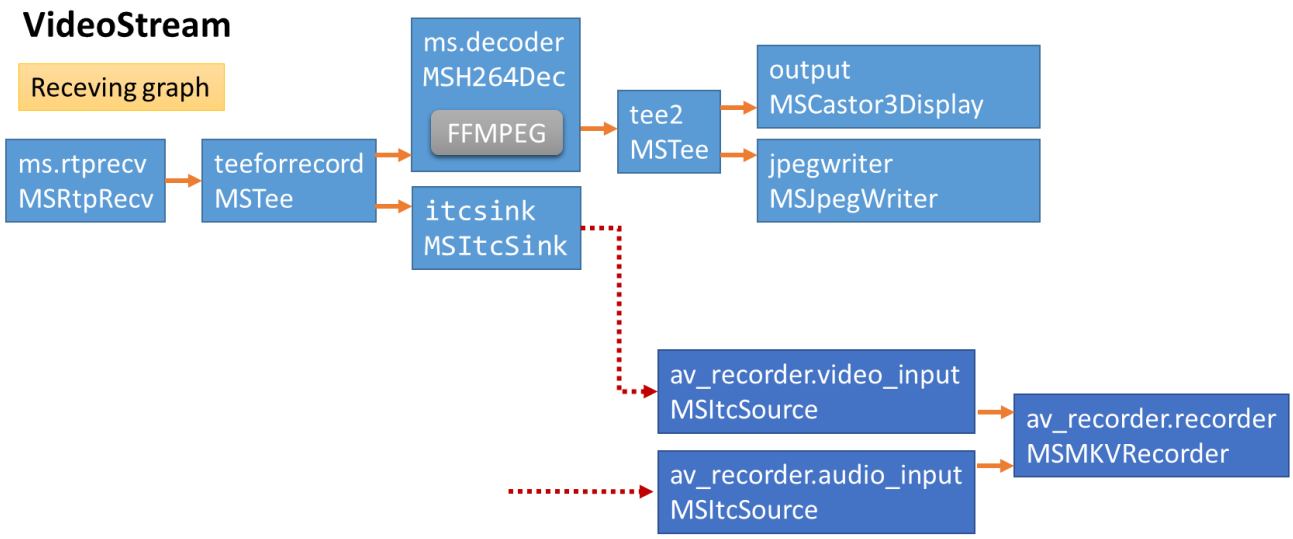
2.3 iTE Filters

SDK 針對 mediastreamer2 開發了基於 MSFilter 架構的 filters :

MSH264Dec: 處理 H.264 硬件解碼的模組
MSCastor3Display: 影像顯示的模組
MSCastor3SndRead: 處理聲音輸入的模組
MSCastor3SndWriter: 處理聲音輸出的模組
MSSpeexEC: 處理回音消除模組
MSJpegWriter: 處理 JPEG 硬件編碼的模組
MSMkvRecorder: 處理 AV recorder 模組

3. Video Streaming

iTE 可視對講室內機，基於 mediastreamer2 架構所建立的 video stream graph 如下圖:

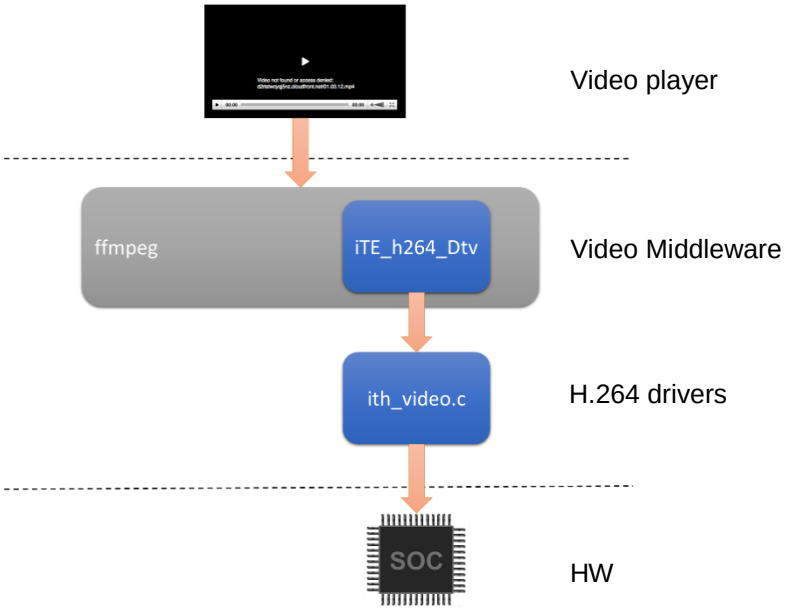


請參考/sdk/share/mediastreamer2/videostream.c
其中 filter MSH264Dec 主要處理 H264 硬體解碼的模組，其中所使用的 API 是基於 FFMPEG 來完成的, FFMPEG 為一個開源的程式碼，其主要的目的是將 H264 stream parse 出來，然後輸入給 H264 HW decoder 來解碼. 詳細資料可參考 FFMPEG 官網: http://www.ffmpeg.org.

Filter MSCastor3Display 主要處理將 Decode 出來的影像資料顯示到 LCD 螢幕上
Filter MSMKVRecorder 則是處理儲存 AV stream 到在地儲存裝置

3.1 Hardware video decoder

此節描述 hardware video decoder 相關 source code 放置的目錄位置。在 SDK 內，操作 hardware video decoder 相關的 code 是被包裝成 ffmpeg library 底下的一個 AVCodec，其主要 source code 分別放置在 sdk\share\ffmpeg\libavcodec\iTE_h264_Dtv 目錄內以及 sdk\driver\ith\ith_video.c。其中 ith_video.c 是負責直接存取 hardware register。而 sdk\share\ffmpeg\libavcodec\iTE_h264_Dtv 則是負責將 hardware video decoder 包裝成 ffmpeg 內一個 AVCodec。任何上層的 player 都應透過 ffmpeg 公開的 interface 來使用 hardware video decoder。強烈建議使用者將 h/w video deocder 當成 ffmpeg 內的一個 AVCodec 來使用，直接使用 ffmpeg 公開的 interface，使用方式可以參考"iTE Media Player 開發指南"文件。



3.2 FFMPEG AVCODEC

sdk\share\ffmpeg\libavcodec\iTE_h264_Dtv 是負責將 hardware video decoder 包裝成 ffmpeg 內一個 AVCodec。使用者可以參考 sdk\share\ffmpeg\libavcodec\iTE_h264_Dtv\avc_video_decoder.c 中 AVCodec structure，透過 avcodec_init()做初始化及 avcodec_register_all()註冊，將 h/w video decoder 包裝成 ffmpeg 內的一個 codec。

```
#if CONFIG_ITE_H264_DTV_DECODER
#if defined(_MSC_VER)
AVCodec ff_iTE_h264_Dtv_decoder = {
    "iTE_h264_Dtv",
    AVMEDIA_TYPE_VIDEO,
    CODEC_ID_H264,
    sizeof(H264Context),
    iTE_h264_Dtv_decode_init,
    NULL,
    iTE_h264_Dtv_decode_end,
    iTE_h264_Dtv_decode_frame,
    /*CODEC_CAP_DRAW_HORIZ_BAND |*/ CODEC_CAP_DR1 | CODEC_CAP_DELAY |
      CODEC_CAP_SLICE_THREADS | CODEC_CAP_FRAME_THREADS,
    NULL,
    iTE_h264_Dtv_decode_flush/*flush_dpb*/,
    NULL,
    NULL,
    "H.264 / AVC / MPEG-4 AVC / MPEG-4 part 10",
    NULL,
    NULL,
```

```
    NULL,
    0,
    &h264_class/*,
    NULL_IF_CONFIG_SMALL(profiles),
    ONLY_IF_THREADS_ENABLED(decode_init_thread_copy),
    ONLY_IF_THREADS_ENABLED(decode_update_thread_context),*/
};
#else // !defined (_MSC_VER)
AVCodec ff_iTE_h264_Dtv_decoder = {
    .name           = "iTE_h264_Dtv",
    .type           = AVMEDIA_TYPE_VIDEO,
    .id             = CODEC_ID_H264,
    .priv_data_size      = sizeof(H264Context),
    .init           = iTE_h264_Dtv_decode_init,
    .close          = iTE_h264_Dtv_decode_end,
    .decode         = iTE_h264_Dtv_decode_frame,
    .capabilities      = /*CODEC_CAP_DRAW_HORIZ_BAND |*/ CODEC_CAP_DR1 | CODEC_CAP_DELAY |
      CODEC_CAP_SLICE_THREADS | CODEC_CAP_FRAME_THREADS,
    .flush          = iTE_h264_Dtv_decode_flush,
    .long_name          = "H.264 / AVC / MPEG-4 AVC / MPEG-4 part 10",
    .priv_class         = &h264_class,
};
#endif // _MSC_VER
#endif // CONFIG_ITE_H264_DTV_DECODER
```

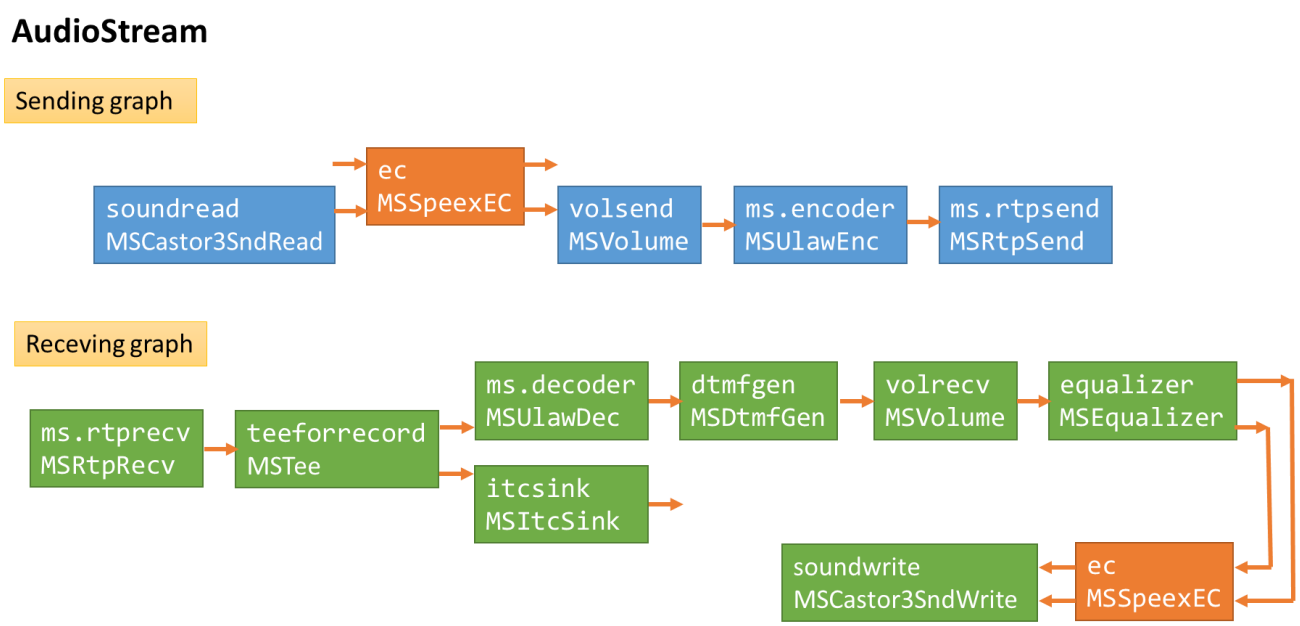
如果使用者想更進一步了解如何實作 AVCodec 及運作細節的部份，可以參考 ffmpeg 官網提供的文件說明，官網網址: https://www.ffmpeg.org/documentation.html 。
除了提供 ffmpeg 接口的部分，這個目錄內的 code 還提供如何控制 h/w 解碼部分及準備 h/w 解碼的資料，主要是將收到的 h.264 bitstream parse 到 slice layer 並轉成 hardware 所需 format。相關程式碼可以參考 sdk\share\ffmpeg\libavcodec\iTE_h264_Dtv\avc.c。

3.3 H264 Hardware Decoder

sdk\driver\ith\ith_video.c 是負責直接存取 hardware register。提供 API 供上層調用
ex.
ithVideoOpen()：初始化硬件。
ithVideoWait()：等待解碼完成。
ithVideoFire()：啟動硬件解碼。
ithVideoClose()：關閉硬件。
ithVideoQuery(): 查詢硬件狀態。

4. Audio Streaming

iTE 可視對講室內機，基於 mediastream2 架構所建立的 Audio stream graph 如下圖:



請參考/sdk/share/mediastreamer2/audiostream.c
Audio streaming 包含兩個 graphs，一個是傳送部分，另一個是接收部分.
MSCastor3SndRead Filter 主要是接收來自 IIS 介面的資料
MSCastor3SndWriter Filter 主要是將 audio 資料送到 IIS 介面
MSSpeexEC Filter 是處理回音消除的模組，有別於 mediastreamer2 上是使用 speex ec 技術，此部分是使用 ITE 內建的回音消除 DSP 來完成.

4.1 I2S APIs

在/sdk/driver/i2s/* 提供 I2S 相關的 driver code 與 APIs, 負責直接存取 I2S hardware registers.

Init 初始化

```
void i2s_init_DAC(STRC_I2S_SPEC *i2s_spec);
void i2s_init_ADC(STRC_I2S_SPEC *i2s_spec);
```

Deinit

```
void i2s_deinit_ADC(void);
void i2s_deinit_DAC(void);
```

Pause 暫停

```
void i2s_pause_ADC(int pause);
void i2s_pause_DAC(int pause);
```

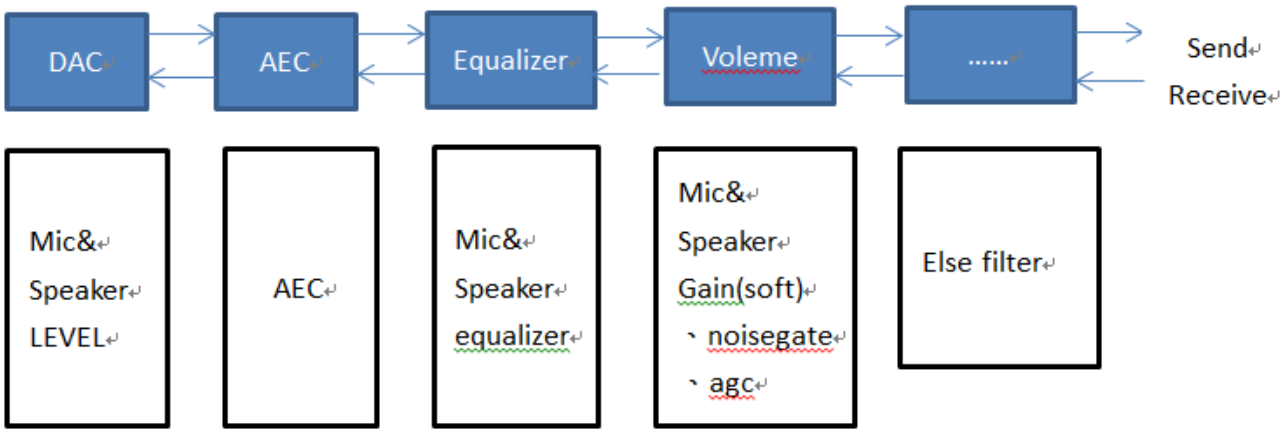
聲音大小設置(volperc =0~100)

```
int i2s_set_direct_volperc(unsigned volperc)          //speaker
int i2s_ADC_set_direct_volstep(unsigned volstep)      //mic
```

mute

```
void i2s_mute_DAC(int mute)
void i2s_mute_ADC(int mute)
```

4.2 Audiostream filter



當通話過程中雙方聲音互相傳播,聲音的大小調整、回音消除、等化器、語音壓縮、混入聲音等可藉由 audiostream filter 模組達成，詳情請查閱。

聲音大小調整 :/sdk/share/mediastreamer2/msvolume.c
等化器 :/sdk/share/mediastreamer2/equalizer.c
混入聲音 :/sdk/share/mediastreamer2/mixvoice.c
語音壓縮 :/sdk/share/mediastreamer2/ulaw.c、alaw.c
回音消除 :/sdk/share/mediastreamer2/sbc_aec.c

亦可自行增加所需要的聲音 filter 藉由串連 filter 達成所需要的 DSP。

連接 filter 部分請參閱/sdk/share/mediastreamer2/audiostream.c

```
int audio_stream_start_full(….){
…
      ms_connection_helper_start(&h);
      ms_connection_helper_link(&h,stream->soundread,-1,0);
      if (stream->ec)
            ms_connection_helper_link(&h,stream->ec,1,1);
      if (stream->equalizerMIC)
            ms_connection_helper_link(&h,stream->equalizerMIC,0,0);
      if (stream->volsend)
            ms_connection_helper_link(&h,stream->volsend,0,0);
      ms_connection_helper_link(&h,stream->ms.encoder,0,0);
      ms_connection_helper_link(&h,stream->ms.rtpsend,0,-1);.
…
}
```

簡單說明此 audiostream 代表由 mic 送至網路中所經過的 filter。

soundread->AEC-> equalizerMIC-> volsend-> encoder-> rtpsend

5. JPEG Codec

iTE 可視對講室內機 SDK 提供 JPEG 硬體編碼與解碼功能.

5.1 JPEG encoder

當室內機接通來電時，或是按下"快照"按鍵時，系統會將當時的影像擷取下來，並啟動 JPEG 壓縮，再將此 JPEG 檔儲存在指定位置. iTE 可視對講室內機 SDK 利用在 mediastream2 的 video stream graph 中增加一個 MSJpegWriter filter 來達成此目的，請參考/sdk/share/mediastreamer2/jpegwriter.c

5.2 JPEG decoder

當在室內機查看留言紀錄，其中可能包含 JEPG 檔案，或是從管理中心送來的社區服務訊息包含 JPEG 檔案，系統會啟動 JPEG 解壓縮，將 JPEG 還原成影像資料再輸出是螢幕. iTE 可視對講室內機 SDK 提供 ituIconLoadJpegData() ， ituJpegLoad() APIs 來達成此目的. 請參考 /sdk/share/itu/itu_icon.c 與 /sdk/share/itu/itu_jpeg.c

5.3 JPEG Hardware Codec

sdk\driver\jpg* 提供相關 Jpeg driver code，負責直接存取 hardware register。Ite_jpg.c 提供 API 供上層調用 ex.

iteJpg_CreateHandle(): 產生一個 Handler
iteJpg_DestroyHandle(): 釋放 Handler
iteJpg_SetStreamInfo(): 設定 stream information
iteJpg_Parsing(): 解析
iteJpg_Setup(): Jpeg 設定
iteJpg_Process(): 執行 HW Jpeg 硬體
iteJpg_SetBaseOutInfo():設定輸出 information
iteJpg_Reset(): HW.重置
iteJpg_WaitIdle(): 等待 HW 執行完成