vpu_api.h 接口说明文档

目录

1. RK 支持的编解码类型

只支持H264 与VP8 编解码。

2. 头文件与库文件

需要头文件：

- vpu_api.h

需要库文件：

- librk_vpuapi.so
- libvpu.so

3. 结构体介绍

3.1 编解码枚举定义

编码输入的图像枚举类型：

```
typedef enum
{
    VPU_H264ENC_YUV420_PLANAR = 0,      /* YYYY... UUUU... VVVV */
    VPU_H264ENC_YUV420_SEMIPLANAR = 1,  /* YYYY... UVUVUV...    */
    VPU_H264ENC_YUV422_INTERLEAVED_YUYV = 2, /* YUYVYUYV...         */
    VPU_H264ENC_YUV422_INTERLEAVED_UYVY = 3, /* UYVYUYVY...         */
    VPU_H264ENC_RGB565 = 4, /* 16-bit RGB          */
    VPU_H264ENC_BGR565 = 5, /* 16-bit RGB          */
    VPU_H264ENC_RGB555 = 6, /* 15-bit RGB          */
    VPU_H264ENC_BGR555 = 7, /* 15-bit RGB          */
    VPU_H264ENC_RGB444 = 8, /* 12-bit RGB          */
    VPU_H264ENC_BGR444 = 9, /* 12-bit RGB          */
    VPU_H264ENC_RGB888 = 10,    /* 24-bit RGB          */
    VPU_H264ENC_BGR888 = 11,    /* 24-bit RGB          */
    VPU_H264ENC_RGB101010 = 12, /* 30-bit RGB          */
    VPU_H264ENC_BGR101010 = 13  /* 30-bit RGB          */
} H264EncPictureType;
```

编解码类型结构体：

```
typedef enum OMX_RK_VIDEO_CODINGTYPE
{
    OMX_RK_VIDEO_CodingUnused,     /**< Value when coding is N/A */
    OMX_RK_VIDEO_CodingAutoDetect, /**< Autodetection of coding type */
    OMX_RK_VIDEO_CodingMPEG2,      /**< AKA: H.262 */
```

```
    OMX_RK_VIDEO_CodingH263,       /**< H.263 */
    OMX_RK_VIDEO_CodingMPEG4,      /**< MPEG-4 */
    OMX_RK_VIDEO_CodingWMV,  /**< Windows Media Video (WMV1,WMV2,WMV3)*/
    OMX_RK_VIDEO_CodingRV,         /**< all versions of Real Video */
    OMX_RK_VIDEO_CodingAVC,        /**< H.264/AVC */
    OMX_RK_VIDEO_CodingMJPEG,      /**< Motion JPEG */
    OMX_RK_VIDEO_CodingVP8,                /**< VP8 */
    OMX_RK_VIDEO_CodingVP9,                /**< VP9 */
    OMX_RK_VIDEO_CodingVC1=0x01000000,
/**<WindowsMediaVideo(WMV1,WMV2,WMV3)*/
    OMX_RK_VIDEO_CodingFLV1,       /**< Sorenson H.263 */
    OMX_RK_VIDEO_CodingDIVX3,              /**< DIVX3 */
    OMX_RK_VIDEO_CodingVP6,
    OMX_RK_VIDEO_CodingHEVC,       /**< H.265/HEVC */
    OMX_RK_VIDEO_CodingKhronosExtensions = 0x6F000000,
/**< Reserved region for introducing Khronos Standard Extensions */
OMX_RK_VIDEO_CodingVendorStartUnused = 0x7F000000,
/**< Reserved region for introducing Vendor Extensions */
    OMX_RK_VIDEO_CodingMax = 0x7FFFFFFF
} OMX_RK_VIDEO_CODINGTYPE;
```

编解码类别结构体：

```
typedef enum CODEC_TYPE
{
    CODEC_NONE,
    CODEC_DECODER,/*Decoding*/
    CODEC_ENCODER,/*Encoding*/
    CODEC_BUTT,
} CODEC_TYPE;
```

编解码错误返回值类型：

```
typedef enum VPU_API_ERR
{
    VPU_API_OK = 0,
    VPU_API_ERR_UNKNOW = -1,
    VPU_API_ERR_BASE = -1000,
    VPU_API_ERR_LIST_STREAM        =     VPU_API_ERR_BASE - 1,
    VPU_API_ERR_INIT               =     VPU_API_ERR_BASE - 2,
    VPU_API_ERR_VPU_CODEC_INIT     =     VPU_API_ERR_BASE - 3,
    VPU_API_ERR_STREAM             =     VPU_API_ERR_BASE - 4,
    VPU_API_ERR_FATAL_THREAD       =     VPU_API_ERR_BASE - 5,
    VPU_API_EOS_STREAM_REACHED     =     VPU_API_ERR_BASE - 11,
    VPU_API_ERR_BUTT,
} VPU_API_ERR;
```

3.2 编解码结构体定义

视频码流包信息结构体：

```
typedef struct VideoPacket
{
    RK_S64 pts;              /* with unit of us*/
    RK_S64 dts;              /* with unit of us*/
    RK_U8 *data;         /* start address of stream data per frame*/
    RK_S32 size;             /* length of the stream per frame*/
    RK_U32 capability;    /* NOTE：  0->no packet loss 1       packet loss*/
    RK_U32 nFlags;
} VideoPacket_t;
```

输出存放解码后的图像结构体：

```
typedef struct DecoderOut
{
    RK_U8 *data;              /*image data after decoding*/
    RK_U32 size;              /*image size after decoding*/
    RK_S64 timeUs;
    RK_S32 nFlags;
} DecoderOut_t;
```

编码图像输入结构体：

```
typedef struct EncInputStream
{
    RK_U8 *buf;      /* image data after encoding */
    RK_S32 size;   /* image size after encoding *
    RK_U32 bufPhyAddr; /*image's physical address after encoding*/
    RK_S64 timeUs;
    RK_U32 nFlags;
} EncInputStream_t;
```

编码码流输出结构体：

```
typedef struct EncoderOut
{
    RK_U8 *data;   /*stream data per frame */
    RK_S32 size;    /*stream data per frame*/
    RK_S64 timeUs;
    RK_S32 keyFrame; /*key frame*/
} EncoderOut_t;
```

编码参数结构体：

```
typedef struct EncParameter
{
    int width;
    int height;
    int rc_mode;
    int bitRate;
    int framerate;
    int  qp;
    int  enableCabac;
    int  cabacInitIdc;
    int  format;
    int  intraPicRate;
    int  framerateout;
    int  profileIdc;
    int  levelIdc;
    int  reserved[3];
} EncParameter_t;
```

编码参数结构体： --> /*未用到不需要设置*/

```
typedef struct
{
    int width;
    int height;
    int rc_mode;
    int bitRate;
    int framerate;
    int  qp;
    int  reserved[10];
} EncParams1;
```

编解码额外参数: --> /*未用到不需要设置*/

```
typedef struct EXtraCfg {
    RK_S32 vc1extra_size;
    RK_S32 vp6codeid;
    RK_S32 tsformat;
    RK_U32 reserved[20];
} EXtraCfg_t;
```

VPU 编解码上下文结构体：

```
typedef struct VpuCodecContext
{
    void* vpuApiObj;
    CODEC_TYPE codecType;
```

```
    OMX_RK_VIDEO_CODINGTYPE videoCoding;
    RK_U32 width;
    RK_U32 height;
    RK_U8 *extradata;
    RK_S32 extradata_size;
    RK_U8  enableparsing;
    RK_S32 no_thread;
    EXtraCfg_t extra_cfg;
    void* private_data;
    /*
     ** 1: error state(not working)  0: working
    */
    RK_S32 decoder_err;
    /**
     * Allocate and initialize an VpuCodecContext.
     *
     * @param ctx The context of vpu api, allocated in this function.
     * @param extraData The extra data of codec, some codecs need / can
     *        use extradata like Huffman tables, also live VC1 codec can
     *        use extradata to initialize itself.
     * @param extra_size The size of extra data.
     *
     * @return 0 for init success, others for failure.
     * note: check whether ctx has been allocated success after you do init.
     */
    RK_S32 (*init)(struct VpuCodecContext *ctx, RK_U8 *extraData, RK_U32 extra_size);
    /**
     * @return 0 for decode success, others for failure.
     */
    RK_S32 (*decode)(struct VpuCodecContext *ctx, VideoPacket_t *pkt, DecoderOut_t *aDecOut);
    /**
     * @return 0 for encode success, others for failure.
     */
    RK_S32 (*encode)(struct VpuCodecContext *ctx, EncInputStream_t *aEncInStrm,
EncoderOut_t *aEncOut);
    /**
     * flush codec while do fast forward playing.
     *
     * @return 0 for flush success, others for failure.
     */
    RK_S32 (*flush)(struct VpuCodecContext *ctx);
    RK_S32 (*control)(struct VpuCodecContext *ctx, VPU_API_CMD cmdType, void* param);
    /**
```

```
    *seperate the decode function to two function
    *
    */
    RK_S32 (*decode_sendstream)(struct VpuCodecContext *ctx, VideoPacket_t *pkt);
    RK_S32 (*decode_getframe)(struct VpuCodecContext *ctx, DecoderOut_t *aDecOut);
    RK_S32 (*encoder_sendframe)(struct VpuCodecContext *ctx,  EncInputStream_t *aEncInStrm);
    RK_S32 (*encoder_getstream)(struct VpuCodecContext *ctx, EncoderOut_t *aEncOut);
} VpuCodecContext_t;
/* allocated vpu codec context */
#ifdef __cplusplus
extern "C"
{
#endif
RK_S32 vpu_open_context(struct VpuCodecContext **ctx);
#ifdef __cplusplus
}
#endif
#ifdef __cplusplus
extern "C"
{
#endif
RK_S32 vpu_close_context(struct VpuCodecContext **ctx);
#ifdef __cplusplus
}
#endif
```

4. 解码调用流程

解码不大于720P30fps 情况下的设置示例：

```
VpuCodecContext_t *_vpu_ctx = NULL;
```

4.1 解码创建过程

(1) 创建_vpu_ctx

```
int ret = vpu_open_context(&_vpu_ctx);
if (ret < 0 || (_vpu_ctx == NULL))
{
        ERROR ("can not create context"); return -1;
}
```

(2) 初始化

```
_vpu_ctx->width = 1280;
_vpu_ctx->height = 720;/*resolution720P*/
_vpu_ctx->videoCoding =              OMX_RK_VIDEO_CodingAVC;/*H264*/
```

```
_vpu_ctx->codecType = CODEC_DECODER;/*decoding*/
ret = _vpu_ctx->init(_vpu_ctx, NULL, 0); if (ret < 0)
{
    ERROR (" decoder init failed");
    return -1;
}
```

4.2 解码过程

```
VideoPacket_t pkt;
DecoderOut_t        picture;
VPU_FRAME* _vpu_fm;
memset(&pkt, 0, sizeof(VideoPacket_t));
memset(&picture, 0, sizeof(DecoderOut_t));
pkt.data = coded_data_ptr;
pkt.size = frame_len;

picture.data = (RK_U8*)_vpu_fm; picture.size = 0;
int ret = _vpu_ctx->decode(_vpu_ctx, &pkt, &picture);
if (ret != 0)
{
      ALOGE("[ER] failed to decode this frame"); return -1;
}
if (_vpu_fm->vpumem.vir_addr == NULL)
{
      解码出数据，需要缓存2帧数据，解码需要小于30fps数据。
}
```

注意:数据使用完后需要释放

```
if (_vpu_fm->vpumem.vir_addr != NULL)
{
        VPUMemLink(&_vpu_fm->vpumem);
        VPUFreeLinear(&_vpu_fm->vpumem);
        memset(_vpu_fm, 0, sizeof(VPU_FRAME));
}
```

注：decoder_getstream和decoder_sendframe与decode接口功能相同；

4.3 解码销毁过程

```
if (_vpu_ctx != NULL)
{
      int res = vpu_close_context(&_vpu_ctx);
      _vpu_ctx = NULL;
}
```

5. 编码调用流程

编码不大于720P30fps 的设置示例：

VpuCodecContext_t *_vpu_ctx = NULL;

5.1 编码创建过程

(1) 创建_vpu_ctx

```
int ret = vpu_open_context(&_vpu_ctx);
if (ret < 0 || (_vpu_ctx == NULL))
{
        ERROR ("can not create context"); return -1;
}
```

(2) 初始化

```
EncParameter_t  *_vpu_enc_param;
_vpu_enc_param = (EncParameter_t*)malloc(sizeof(EncParameter_t)); memset(_vpu_enc_param,
0, sizeof(EncParameter_t));
_vpu_ctx->width = 1280;
_vpu_ctx->height = 720;/*分辨率720P*/
_vpu_ctx->videoCoding =  OMX_RK_VIDEO_CodingAVC;/*H264*/
_vpu_ctx->codecType = CODEC_ENCODER;/*编码*/
_vpu_enc_param->bitRate = 512;/*比特率*/
_vpu_enc_param->enableCabac = false;/*使能cabac*/
_vpu_enc_param->cabacInitIdc = 0;
_vpu_enc_param->format = 0;
_vpu_enc_param->framerate =30;/*帧率*/
_vpu_enc_param->width = 1280;
_vpu_enc_param->height = 720;
_vpu_enc_param->profileIdc = profile;
_vpu_enc_param->intraPicRate =_gop_len;
_vpu_enc_param->levelIdc = level_id;
_vpu_enc_param->rc_mode = 1;
_vpu_enc_param->qp = 35;
_vpu_ctx->private_data    =    _vpu_enc_param;    ret    =
_vpu_ctx->init(_vpu_ctx, NULL, 0); if (ret < 0)
{
    ERROR (" decoder init failed");
    return -1;
}
```

/*编码参数控制，获取编码参数*/

```
EncParameter_t vpug;
_vpu_ctx->control(_vpu_ctx,VPU_API_ENC_GETCFG,(void*)&vpug);
```

```
/*编码参数控制，设置编码参数*/
vpug.rc_mode = 1;
_vpu_ctx->control(_vpu_ctx,VPU_API_ENC_SETCFG,(void*)&vpug);
/*编码参数控制，设置编码类型*/
H264EncPictureType encType = VPU_H264ENC_YUV420_SEMIPLANAR; //nv12
res=_vpu_ctx->control(_vpu_ctx,VPU_API_ENC_SETFORMAT, (void*)
&encType); if
(res < 0)
{
    ERROR ("init control failed!");
    return -1;
}
```

5.2 编码过程

```
EncInputStream_t input;/*输入图片信息*/
input.buf = pic.viraddr /*图像地址*/
input.size = size;/*图像数据尺寸*/
res = _vpu_ctx->encoder_sendframe(_vpu_ctx, &input);
if (res != 0)
{
      ERROR ("enc send frame to vpu error"); return -1;
}
EncoderOut_t output;/*编码后的码流数据*/
 memset(&output, 0, sizeof(EncoderOut_t));
output.data = NULL;
output.size = 0;            /*码流尺寸*/
res = _vpu_ctx->encoder_getstream(_vpu_ctx, &output);
if (res != 0)
{
      ERROR ("enc get frame to vpu error"); return -1;
}
```

注：encoder_getstream 和encoder_sendframe 与encoder接口功能相同；

5.3 编码销毁过程

```
if (_vpu_ctx != NULL)
{
      res = _vpu_ctx->flush(_vpu_ctx);
      if (res != 0)
      {
```

```
        ERROR ("enc flush vpu error"); return -1;
    }
    res = vpu_close_context(&_vpu_ctx); if (res != 0)
    {
        ERROR ("enc close vpu error"); return -1;
    }
    _vpu_ctx = NULL;
}
```

注：丢包需要申请I帧，RK硬件编码不支持多slice编码。