密级状态：绝密（　　）　秘密（　　）　内部（ ）　公开（ √ ）

RK3399_Android7.1_MPI_Demo_说明文档

（第二系统产品部，技术部）

文件状态： [　] 正在修改 [√] 正式发布	当前版本：	V1.0
	作　　者：	王剑辉
	完成日期：	2018-04-09
	审　　核：	邓训金、陈海燕
	完成日期：	2018-04-10

福州瑞芯微电子股份有限公司

Fuzhou　Rockchips　Semiconductor　Co . , Ltd

版 本 历 史

版本号	作者	修改日期	修改说明	备注
V1.0	王剑辉	2018-04-09	初始版本	

目 录

1 mpi_multi_test 简介

mpi_multi_test 是基于 mpp api 接口开发的测试 demo。Mpp api 是 Rockchip 算法组新开发的一套编解码 api。mpi_multi_test 支持同时多分辨率的多路编码和多路解码，只需要修改传递的参数。

接口支持：

- h264 和 jpeg 编码
- mpeg2,mpeg4,h263,h264,h265,vp8,vp9,avs+解码

2 mpi_multi_test 编译

2.1 mpi_multi_test 源码路径

多路编解码 demo 源码路径：external/mpp-demo。重要源码目录说明：

```
├── build
|   ├── android
|   |   ├── android.toolchain.cmake 编译规则文件
|   |   ├── arm
|   |   |   ├── make-Android.bash 编译脚本文件
├── doc 说明文档
├── inc 头文件
├── mpp 编解码基础库文件
├── osal 系统层文件
├── test
|   ├── mpi_multi_test.c 多路编解码 demo 源码
├── tools 调试分析工具
├── utils 公用方法文件
```

2.2 mpi_multi_test 源码编译

编译命令：

cd external/mpp-demo/build/android/arm/

./make-Android.bash

编译完成后，会生成 so 库和可执行文件：

test/mpi_multi_test

mpp/legacy/libvpu_demo.so

mpp/libmpp_demo.so

重命名 so 库名称：

mv mpp/legacy/libvpu_demo.so mpp/legacy/libvpu.so

mv mpp/libmpp_demo.so mpp/libmpp.so

2.3 推送 so 库和可执行文件到 3399 开发板

adb root;adb disable-verity;adb reboot

adb root;adb remount

adb push mpi_multi_test /system/bin/

adb push libvpu.so /system/lib/

adb push libmpp.so /system/lib/

adb shell sync

3 多路解码

3.1 Mpi_multi_test 使用和参数说明

测试多分辨率的多路 H264 解码（写文件）：

```
./mpi_multi_test
{i=/data/640x480.h264:o=/data/640x360.h264.yuv:w=640:h=360:t=7:p=10:q=1}
```

```
{i=/data/1280x720.h264:o=/data/1280x720.h264.yuv:w=1280:h=720:t=7:p=1:q=1}

{i=/data/1920x1080.h264:o=/data/1920x1080.h264.yuv:w=1920:h=1080:t=7:p=1:q=1}
```

参数说明：（参数之间用:隔开）

i 指定待解码的文件

o 指定报错解码后的文件（当测试 vpu 解码性能时，不需要填，排除写 flash 影响）

w 视频宽（多分辨率同时解码时，必须填写）

h 视频高（多分辨率同时解码时，必须填写）

p 同时几路解码

q 编码或者解码，q=1 表示解码，q=2 表示编码

t 指定协议类型，7 代表的是 H.264，具体支持以下几种：

```
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,            /**< Windows Media Video
(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_CodingAVS,                     /**< AVS+ */
    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;
```

测试多分辨率的多路 H264 解码（丢弃解码后的数据，排除 io 性能影响解码）：

```
./mpi_multi_test {i=/data/640x480.h264:w=640:h=360:t=7:p=10:q=1}
{i=/data/1280x720.h264: w=1280:h=720:t=7:p=1:q=1} {i=/data/1920x1080.h264:
w=1920:h=1080:t=7:p=1:q=1}
```

mpi_multi_test 多路解码是通过创建多个线程来实现解码，每路解码里面又分为两个线程：send packet 线程和 get frame 线程。

Send packet 线程：调用 decode_sendstream 将待解码的 packet 传入给解码器，Packet 不需要外部分帧，mpp 内部会自行组帧。

Get frame 线程：调用 decode_getframe 从解码器获取解码后的 frame，decode_getframe 为阻塞接口，还未能获取到 frame 时阻塞。

3.2 解码逻辑

如何判断解码全部结束？

送解码时依赖的 packet 结构是 struct VideoPacket_t，通过设置 VideoPacket_t 的 nFlags 成员在 packet 中打上结束标志，在解码到最后一帧时 decode_getframe 会返回 -1011 作为标识。

3.3 宽高对齐

以输入源为 1080p 为例：

1. H264 解码后，输出的图像宽 1920 高 1088(h264 宽高均 16 对齐)；

2. H265 解码后，输出的图像宽 2160 高 1088(h265 宽 256 对齐，高 1088 对齐)；

4 多路编码

4.1 mpi_multi_test 使用和参数说明

测试多分辨率的多路 H264 编码:（写文件）

```
./ mpi_multi_test
{i=/data/640x480.yuv:o=/data/640x480.yuv.h264:w=640:h=480:f=4:n=100:t=7:p=1:q=2}
{i=/data/1280x720.yuv:o=/data/1280x720.yuv.h264:w=1280:h=720:f=4:n=100:t=7:p=1:q=2}
{i=/data/1920x1080.yuv:o=/data/1920x1080.yuv.h264:w=1920:h=1080:f=4:n=100:t=7:p=1:
q=2}
```

参数说明:（参数之间用:隔开）

i 指定待编码的文件

o 指定报错解码后的文件（当测试 vpu 解码性能时，不需要填，排除写 flash 影响）

w 视频宽

h 视频高

p 同时几路编码，p=1 表示一路编码

q 编码或者解码，q=1 表示解码，q=2 表示编码

t 指定编码后协议类型，7 代表的是 H.264，8 代表的是 JPEG，参考 2.1 章节说明

f 是输入图像的格式，当原始数据为 YUV420P 格式时，f 设为 4；原始数据为 NV12 时，f 设置为 5。具体指出以下几种：

```
typedef enum {
    MPP_FMT_YUV420SP          = MPP_FRAME_FMT_YUV,          /* YYYY... UV... (NV12)      */
```

```
    MPP_FMT_YUV420SP_10BIT,
    MPP_FMT_YUV422SP,                       /* YYYY... UVUV... (NV24)    */
    MPP_FMT_YUV422SP_10BIT,                  ///< Not part of ABI
    MPP_FMT_YUV420P,                       /* YYYY... U...V...   (I420) */
    MPP_FMT_YUV420SP_VU,                    /* YYYY... VUVUVU... (NV21) */
    MPP_FMT_YUV422P,                       /* YYYY... UU...VV...(422P) */
    MPP_FMT_YUV422SP_VU,                    /* YYYY... VUVUVU... (NV42) */
    MPP_FMT_YUV422_YUYV,                    /* YUYVYUYV... (YUY2)        */
    MPP_FMT_YUV422_UYVY,                    /* UYVYUYVY... (UYVY)        */
    MPP_FMT_YUV400SP,                       /* YYYY...                   */
    MPP_FMT_YUV440SP,                       /* YYYY... UVUV...           */
    MPP_FMT_YUV411SP,                       /* YYYY... UV...             */
    MPP_FMT_YUV444SP,                       /* YYYY... UVUVUVUV...       */
    MPP_FMT_YUV_BUTT,
    MPP_FMT_RGB565          = MPP_FRAME_FMT_RGB,        /* 16-bit RGB              */
    MPP_FMT_BGR565,                         /* 16-bit RGB                */
    MPP_FMT_RGB555,                         /* 15-bit RGB                */
    MPP_FMT_BGR555,                         /* 15-bit RGB                */
    MPP_FMT_RGB444,                         /* 12-bit RGB                */
    MPP_FMT_BGR444,                         /* 12-bit RGB                */
    MPP_FMT_RGB888,                         /* 24-bit RGB                */
    MPP_FMT_BGR888,                         /* 24-bit RGB                */
    MPP_FMT_RGB101010,                       /* 30-bit RGB                */
    MPP_FMT_BGR101010,                       /* 30-bit RGB                */
    MPP_FMT_ARGB8888,                        /* 32-bit RGB                */
    MPP_FMT_ABGR8888,                        /* 32-bit RGB                */
```

```
    MPP_FMT_RGB_BUTT,

    /* simliar to I420, but Pixels are grouped in macroblocks of 8x4 size   */

    MPP_FMT_YUV420_8Z4        = MPP_FRAME_FMT_COMPLEX,

    /* The end of the formats have a complex layout */

    MPP_FMT_COMPLEX_BUTT,

    MPP_FMT_BUTT              = MPP_FMT_COMPLEX_BUTT,

} MppFrameFormat;
```

测试多分辨率的多路 H264 编码（丢弃编码后的数据，排除 io 性能影响解码）：

```
./ mpi_multi_test {i=/data/640x480.yuv:w=640:h=480:f=4:n=100:t=7:p=1:q=2}

{i=/data/1280x720.yuv:w=1280:h=720:f=4:n=100:t=7:p=1:q=2}

{i=/data/1920x1080.yuv:w=1920:h=1080:f=4:n=100:t=7:p=1:q=2}
```

4.2 编码逻辑

编码 h264，在调用 test_mpp_preprare 后会生成 sps 和 pps，这两个辅助说明信息在后续解码时会用到，缺少 sps 和 pps 无法解码。

```
MPP_RET test_mpp_preprare(MpiEncTestData *p)
{
    MPP_RET ret;
    MppApi *mpi;
    MppCtx ctx;
    MppPacket packet = NULL;

    if (NULL == p)
        return MPP_ERR_NULL_PTR;

    mpi = p->mpi;
    ctx = p->ctx;
    ret = mpi->control(ctx, MPP_ENC_GET_EXTRA_INFO, &packet);
    if (ret) {
        mpp_err("mpi control enc get extra info failed\n");
        goto RET;
    }

    /* get and write sps/pps for H.264 */
    if (packet) {
        void *ptr   = mpp_packet_get_pos(packet);
        size_t len  = mpp_packet_get_length(packet);

        if (p->fp_output)
            fwrite(ptr, 1, len, p->fp_output);

        packet = NULL;
    }
RET:
    return ret;
} ? end test_mpp_preprare ?
```

编码只有一个 encode 接口需要调用，每次调用输入一帧图像的同时获取一帧输出。

4.3 码率控制

编码时的码率控制通过调用 test_mpp_setup 接口来设定。编码时，根据实际需求修改码率等参数，修改 mpi_multi_test.c 文件 test_mpp_setup 函数里面的参数赋值。

```
MPP_RET test_mpp_setup(MpiEncTestData *p)
{
    MPP_RET ret;
    MppApi *mpi;
    MppCtx ctx;
    MppEncCodecCfg *codec_cfg;
    MppEncPrepCfg *prep_cfg;
    MppEncRcCfg *rc_cfg;

    if (NULL == p)
        return MPP_ERR_NULL_PTR;

    mpi = p->mpi;
    ctx = p->ctx;
    codec_cfg = &p->codec_cfg;
    prep_cfg = &p->prep_cfg;
    rc_cfg = &p->rc_cfg;

    /* setup default parameter */
    p->fps = 30;
    p->gop = 60;
    p->bps = p->width * p->height / 8 * p->fps;
    p->qp_init  = (p->type == MPP_VIDEO_CodingMJPEG) ? (10) : (26);
```

```
typedef struct MppEncRcCfg_t {
    RK_U32  change;
    /*
     * rc_mode - rate control mode
     *
     * mpp provide two rate control mode:
     *
     * Constant Bit Rate (CBR) mode
     * - paramter 'bps*' define target bps
     * - paramter quality and qp will not take effect
     *
```

```
 * Variable Bit Rate (VBR) mode
 * - paramter 'quality' define 5 quality levels
 * - paramter 'bps*' is used as reference but not strict condition
 * - special Constant QP (CQP) mode is under VBR mode
 *    CQP mode will work with qp in CodecCfg. But only use for test
 *
 * default: CBR
 */
MppEncRcMode rc_mode;

/*
 * quality - quality parameter, only takes effect in VBR mode
 *
 * Mpp does not give the direct parameter in different protocol.
 *
 * Mpp provide total 5 quality level:
 * Worst - worse - Medium - better - best
 *
 * extra CQP level means special constant-qp (CQP) mode
 *
 * default value: Medium
 */
MppEncRcQuality quality;
/*
 * bit rate parameters
 * mpp gives three bit rate control parameter for control
```

```
 * bps_target    - target   bit rate, unit: bit per second
 * bps_max       - maximun bit rate, unit: bit per second
 * bps_min       - minimun bit rate, unit: bit per second
 * if user need constant bit rate set parameters to the similar value
 * if user need variable bit rate set parameters as they need
 */
RK_S32   bps_target;
RK_S32   bps_max;
RK_S32   bps_min;

/*
 * frame rate parameters have great effect on rate control
 *
 * fps_in_flex
 * 0 - fix input frame rate
 * 1 - variable input frame rate
 *
 * fps_in_num
 * input frame rate numerator, if 0 then default 30
 *
 * fps_in_denorm
 * input frame rate denorminator, if 0 then default 1
 *
 * fps_out_flex
 * 0 - fix output frame rate
 * 1 - variable output frame rate
```

```
 *
 * fps_out_num
 * output frame rate numerator, if 0 then default 30
 *
 * fps_out_denorm
 * output frame rate denorminator, if 0 then default 1
 */
RK_S32   fps_in_flex;
RK_S32   fps_in_num;
RK_S32   fps_in_denorm;
RK_S32   fps_out_flex;
RK_S32   fps_out_num;
RK_S32   fps_out_denorm;

/*
 * gop - group of picture, gap between Intra frame
 * 0 for only 1 I frame the rest are all P frames
 * 1 for all I frame
 * 2 for I P I P I P
 * 3 for I P P I P P
 * etc...
 */
RK_S32   gop;
/*
 * skip_cnt - max continuous frame skip count
 * 0 - frame skip is not allow
```

```
    */
    RK_S32   skip_cnt;
} MppEncRcCfg;
```

rc_mode 可选固定 VBR 模式或 CBR 模式。CBR mode 是固定码率，VBR mode 是变码率。测试 demo 里面用的是 CBR 固定码率，实际使用一般建议用 VBR 模式。

5 多路解码和编码

5.1 mpi_multi_test 使用

测试多分辨率的多路 H264 编解码:（写文件）

```
./ mpi_multi_test
{i=/data/1280x720.yuv:o=/data/1280x720.yuv.h264:w=1280:h=720:f=0:n=100:t=7:p=1:
q=2}
{i=/data/1920x1080.yuv:o=/data/1920x1080.yuv.h264:w=1920:h=1080:f=0:n=100:t=7:p
=1:q=2}
{i=/data/1920x1080.h264:o=/data/1920x1080.h264.yuv:w=1920:h=1080:t=7:p=1:q=1}
```

测试多分辨率的多路 H264 编解码:（丢弃编解码后的数据，排除 io 性能影响解码）

```
./ mpi_multi_test {i=/data/1280x720.yuv:w=1280:h=720:f=0:n=100:t=7:p=1:q=2}
{i=/data/1920x1080.yuv:w=1920:h=1080:f=0:n=100:t=7:p=1:q=2}
{i=/data/1920x1080.h264:w=1920:h=1080:t=7:p=1:q=1}
```

5.2 多路编解码自动化测试

多路编解码测试可以参考 mpp_test.sh 脚本：

```
./mpp_test.sh 1   ---decode_bench is not save yuv file
./mpp_test.sh 2   ---decode_bench_save_file is save yuv file
./mpp_test.sh 3   ---encode_bench is not save yuv file
```

```
./mpp_test.sh 4   ---encode_bench_save_file is save h264 file

./mpp_test.sh 5   ---decode_4K_bench is not save yuv file

./mpp_test.sh 6   ---mpi_test_bench is all-around test
```

测试帧率 log：

```
==> DEC [640 x 360] rate=54
==> DEC [640 x 360] rate=56
==> DEC [640 x 360] rate=54
==> DEC [640 x 360] rate=52
==> DEC [640 x 360] rate=56
==> DEC [640 x 360] rate=56
==> DEC [640 x 360] rate=56
==> DEC [640 x 360] rate=56
==> DEC [640 x 360] rate=56
==> DEC [640 x 360] rate=56
==> DEC [640 x 360] rate=56
==> DEC [640 x 360] rate=54
==> DEC [640 x 360] rate=56
==> DEC [640 x 360] rate=57
==> DEC [640 x 360] rate=58
==> DEC [640 x 360] rate=57
==> DEC [640 x 360] rate=56
==> DEC [640 x 360] rate=63
==> DEC [640 x 360] rate=63
==> DEC [640 x 360] rate=63
==> DEC [640 x 360] rate=63
==> DEC [640 x 360] rate=64
==> DEC [640 x 360] rate=63
==> DEC [640 x 360] rate=63
==> DEC [640 x 360] rate=63
==> DEC [640 x 360] rate=63
==> DEC [640 x 360] rate=62
==> DEC [640 x 360] rate=63
==> DEC [640 x 360] rate=62
==> DEC [640 x 360] rate=63
==> DEC [640 x 360] rate=62
==> DEC [640 x 360] rate=62
==> DEC [640 x 360] rate=63
==> DEC [640 x 360] rate=63
==> DEC [640 x 360] rate=64
==> DEC [640 x 360] rate=64
==> DEC [1920 x 1080] rate=46
==> DEC [640 x 360] rate=58
==> DEC [640 x 360] rate=59
==> DEC [640 x 360] rate=59
==> DEC [1280 x 720] rate=58
==> DEC [640 x 360] rate=57
==> DEC [640 x 360] rate=58
```