全 志 科 技
Allwinner Technology

AWCodec 开发参考手则

V1.2

2013-11-27

Revision History

Version	Date	Changes compared to previous issue
1.0	2013/09/12	创建初始版本
1.1	2013/09/26	修正了编码库中的 BUG，增加了编码 Demo
1.2	2013/11/27	编码加入裁剪接口，MJPEG 编码，移动侦测和码率控制，增加了库的版本号接口，更新了编解码 Demo

目录

1. 概述

1.1. 编写目的

让需要使用 AWCodec 编解码中间件进行开发的开发人员熟悉该中间件的使用方法

1.2. 使用范围

A10s、A13、A20 芯片， Linux/Android 平台

1.3. 目标读者

使用编解码进行开发的同事和客户

2. 模块介绍

AWCodec是全志监控处理平台提供的一个在Linux/Android下使用软硬件编解码音视频的中间件模块，包括编码和解码二个模块。使用 AWCodec 可以实现以下功能：输入视频捕获，视频图像处理，H264/MJPEG/JPEG 编码，H264/AVS/MPEG2/MPEG4/VC1/VP8 解码，视频输出显示，音频捕获及输出，音频编解码等。编码和解码二个模块相互独立，互不影响，支持多线程协同工作，也可以独立多线程运行。

2.1. 模块功能介绍

编码模块：
输入视频捕获：从外部输入视频设备捕获视频
输入音频捕获：从外部输入设备捕获音频
视频编码：可以编码 H264，MJPEG，JPEG
音频编码：可以编码 ADPCM ,AAC, WMA ,MP3

解码模块：
视频解码：可以对 H264，AVS，MPEG2，MPEG4，VC1，VP8 视频格式进行解码
音频解码：可以对 ADPCM ,AAC WMA ,MP3 等音频格式进行解码
视频输出显示：可以直接把数据显示到输出设备，也可以给 GUI 进行显示合成

2.2. 相关术语介绍

SPS:(Sequence Parameter Set)序列参数集，包含了初始化解码器所需要的信息
PPS:(Picture Parameter Set)图像参数集，包含了初始化解码器所需要的的信息
IDR:(Instantaneous Decoding Refresh)即时解码刷新，类似 I 帧，IDR 之后的所有帧不能引用 IDR 之前的帧作参考。
CSI:(CMOS Sensor interface)CMOS 摄像头接口
VBV: (Video bitstream verification) 视频码流管理模块
FBM: (Frame buffer manager)帧缓存管理模块
ISR: (Interrupt service routine)中断服务
PTS: (Presentation time stamp)显示时间戳
VE: (video codec engine)编解码引擎

2.3. 模块配置介绍

编码模块：
需要配置交叉编译工具链，见 Demo 程序

解码模块：

需要配置交叉编译工具链，见 Demo 程序

2.4.源码结构介绍

```
.
├── decDemo
├── decDemo1
├── doc
│   ├── AWCodec 开发参考手则.pdf
│   ├── Libve_User_Guide.pdf
│   └── PMP_file_format.pdf
├── encDemo
├── encDemo1
├── encDemo2
├── include
│   ├── CDX_Debug.h
│   ├── include_base
│   ├── include_camera
│   ├── include_platform
│   ├── include_system
│   ├── include_vdecoder
│   └── include_vencoder
├── lib
│   ├── A13
│   ├── A20
│   └── A20-sunxi
└── lib-android
```

decDemo：解码库 Demo，从 4 个 PMP 文件中获取数据解码并同时显示 4 画面
decDemo1：解码库 Demo，从 PMP 文件中获取数据解码并显示到屏幕
encDemo：编码库 Demo，从文件中获取 YUV 数据进行编码，编码后数据保存到磁盘
encDemo1: 编码库 Demo，从摄像头获取数据和当前时间一起进行编码，编码后数据保存到磁盘
encDemo2: 编码库 Demo，从摄像头获取数据进行编码，编码后数据保存到磁盘同时进行解码显示
include：解码库和编码库提供出来的头文件以及其他需要用到的头文件
lib：A20-sunxi 目录使用 A20 芯片并使用 sunxi 进行内存分配的库。
A20 目录下是使用 A20 芯片，用 ION 进行内存分配的库。
A13 目录下是使用 A13 或者 A10s 的芯片，使用 sunxi 进行内存分配的库。
libvdecoder.so 解码库，linux 平台使用
libvencoder.so 编码库，linux 平台使用
libvecore.so 解码核心库，解码库需要用到，linux 平台使用
这三个库如何使用请参考 Demo 中的 makefile 文件

3. 架构介绍

3.1. 系统架构介绍

AWCodec 所支持的典型系统层次如图 3-1 所示，主要分为以下层次：

- 硬件层

  由 A10s、A13 或者 A20 加上必要的外围器件构成。

- 操作系统层

  支持 Linux 和 Android。

- AWCodec

  基于操作系统，控制芯片完成相应多媒体处理功能，提供 API 接口给应用层，屏蔽了硬件处理细节。

- 其他中间件

  GUI

- 应用层

  基于 AWCodec 以及其他中间件，由客户开发的上层应用软件系统

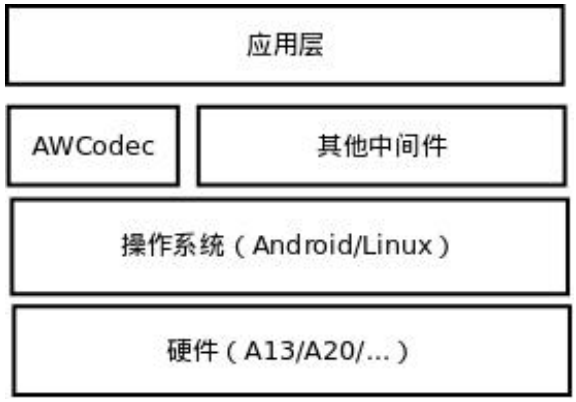

图 3-1

4. 接口描述

4.1.公共模块函数定义

4.1.1. 函数定义：int cedarx_hardware_init(int mode);

参数说明：mode 代表模式，暂时预留，传 0 即可
函数说明：打开 VE 驱动，内存分配驱动，初始化 VE 硬件寄存器
返回值　：返回 0 代表成功，返回-1 代表失败
附加说明：需要在所有函数调用之前调用

4.1.2. 函数定义：int cedarx_hardware_exit(int mode);

参数说明：mode 代表模式，暂时预留，传 0 即可
函数说明：关闭 VE 驱动，关闭内存分配驱动
返回值　：返回 0 代表成功，返回-1 代表失败
附加说明：退出模块最后调用

4.2.编码模块函数定义

4.2.1. 函数定义：cedarv_encoder_t* cedarvEncInit()

参数说明：无
函数说明：初始化编码模块
返回值　：返回编码模块句柄
附加说明：通过编码模块句柄可以操作编码模块提供出来的接口，见 cedarv_encoder_t

4.2.2. 函数定义：void cedarvEncExit(cedarv_encoder_t* handle)

参数说明：handle 代表编码模块句柄
函数说明：退出编码模块
返回值　：0 代表成功，-1 代表失败
附加说明：无

4.2.3. 数据结构：cedarv_encoder_t

接口说明：编码模块句柄
接口定义：

```
typedef struct CEDARV_ENCODER
{
    void *context;
    int (*ioctrl)(void *encoder, VENC_IO_COMMAND cmd, void* param);
} cedarv_encoder_t;
```

int (*ioctrl)(void *encoder, VENC_IO_COMMAND cmd, void* param);
编码模块控制接口，用来完成编码模块的大部分功能，见 VENC_IO_COMMAND

context：编码模块上下文信息

4.2.4. 数据结构：VENC_IO_COMMAND

接口说明：ioctrl 命令
接口定义：

```
enum VENC_IO_COMMAND
{
    VENC_CMD_NOP    =    0,              // 测试接口
    VENC_CMD_BASE_CONFIG,               //   设置基本信息
    VENC_CMD_SET_FROFILE_LEVEL,    //   设置 level 与 profile
    VENC_CMD_SET_QP_RANGE,         //    设置 qp 的范围（maxqp 与 minqp）
    VENC_CMD_SET_INTRA_REFRESH, //
    VENC_CMD_SET_BITRATE,          // 设置目标码率
    VENC_CMD_OPEN,                 // 打开编码设备
    VENC_CMD_CLOSE,                //   关闭编码设备
    VENC_CMD_ENCODE,               //   编一帧数据

    VENC_CMD_ALLOCATE_INPUT_BUFFER,         //   申请 input buffer
    VENC_CMD_GET_ALLOCATE_INPUT_BUFFER , // 获取一个 alloc buffer
    VENC_CMD_FLUSHCACHE_ALLOCATE_INPUT_BUFFER,// alloc buffer 刷 cache
    VENC_CMD_RETURN_ALLOCATE_INPUT_BUFFER,        //还  alloc buffer

    VENC_CMD_ENQUENE_INPUT_BUFFER, // 把输入 buffer 放入输入缓冲队列
    VENC_CMD_DEQUENE_INPUT_BUFFER, // 从输入 buffer 缓冲队列获取 buffer
```

```
        VENC_CMD_SET_MOTION_PAR_FLAG, //设置移动侦测参数
        VENC_CMD_GET_MOTION_FLAG,     //得到移动侦测的标志位

        VENC_CMD_GET_BITSTREAM,       // 获取输出码流
        VENC_CMD_RETURN_BITSTREAM,    // 释放输出码流占用的内存
        VENC_CMD_HEADER_DATA,         // 获取编码的头信息
};
```

VENC_CMD_NOP：测试使用
VENC_CMD_BASE_CONFIG：设置基本信息，见 VencBaseConfig
VENC_CMD_SET_FROFILE_LEVEL：设置 level 与 profile，见 VencProfileLevel
VENC_CMD_SET_QP_RANGE：设置 qp 的范围，见 VencQPrange
VENC_CMD_SET_INTRA_REFRESH：见 VencIntraRefresh
VENC_CMD_SET_BITRATE：设置目标码率，见 VencTargetBitrate
VENC_CMD_OPEN：打开编码设备
VENC_CMD_CLOSE：关闭编码设备
VENC_CMD_ENCODE：编一帧数据，返回结果见__pic_enc_result;

VENC_CMD_ALLOCATE_INPUT_BUFFER：申请 buffer，见 VencAllocateBufferParam
VENC_CMD_GET_ALLOCATE_INPUT_BUFFER：获取 input buffer,见 VencInputBuffer
VENC_CMD_FLUSHCACHE_ALLOCATE_INPUT_BUFFER：buffer 刷 cache
VENC_CMD_RETURN_ALLOCATE_INPUT_BUFFER：释放 buffer,见 VencInputBuffer

VENC_CMD_ENQUENE_INPUT_BUFFER：把输入 buffer 放入输入缓冲队列
VENC_CMD_DEQUENE_INPUT_BUFFER：从输入 buffer 缓冲队列获取 buffer
VENC_CMD_GET_BITSTREAM：获取输出码流，见 VencOutputBuffer
VENC_CMD_RETURN_BITSTREAM：释放输出码流，见 VencOutputBuffer
VENC_CMD_HEADER_DATA：获取编码输出的头信息，见 VencSeqHeader

4.2.5. 数据结构：VencAllocateBufferParam

接口说明：分配连续物理 buffer 用于输入数据
附件说明： 当给编码器送数据的源不能像 CSI 那样提供物理地址连续的 buffer 的时候，需要编码器自己去 allocate 输入 buffer，并对输入 buffer 进行管理，
接口定义：

```
typedef struct VencAllocateBufferParam{
    unsigned int        buffernum;
    unsigned int        buffersize;   // witdh*heigh*3/2 for yuv420
}VencAllocateBufferParam;
```

4.2.6. 数据结构：VencInputBuffer

接口说明：输入 buffer 的信息，通过 CSI 提供的连续物理地址赋值或者 ioctl 控制接口 VENC_CMD_ALLOCATE_INPUT_BUFFER 来进行物理地址的分配，VENC_CMD_GET_ALLOCATE_INPUT_BUFFER 来获取 VencInputBuffer。

附加说明： addrphyY 和 addrphyC 为编码必须的，他们指向输入 buffer 的物理地址；

接口定义：

```
typedef struct VencInputBuffer{
    int                 id;
    unsigned char       *addrvirY; //the luma    vir address for yuv format
    unsigned char       *addrvirC; //the color vir address for yuv format
    unsigned char       *addrphyY; //the luma    phy address for yuv format
    unsigned char       *addrphyC; //the color phy address for yuv format

    int                 bufsize;
    int                 color_fmt;
    long long           pts;        //unit:us
    CROP_INFO           crop_info;
    void*               *addrOverlay;
}VencInputBuffer;
```

id：输入 buffer 的 id 号

addrvirY：输入 buffer 的 Y 分量或 argb 的虚拟地址

addrCb：输入 buffer 的 C 分量的虚拟地址

addrphyY：输入 buffer 的 y 分量或 argb 数据的物理地址；

addrphyC：输入 buffer 的 c 分量的物理地址；

pts：时间戳

color_fmt：颜色编码数据格式，一般 sensor 会给出，见 VENC_PIXEL_FMT

crop_info: 裁剪信息，可以在原始数据在编码时进行裁剪，见 CROP_INFO

addrOverlay: 水印信息，可以在原始数据编码时叠加自定义图片，见 overlay_info

4.2.7. 数据结构：VencOutputBuffer

接口说明：输出 buffer 的信息

附件说明：编码硬件输出用的是一块环形 buffer，当硬件输出的数据在这块 buffer 尾部时，有可能一部分信息输出在这块 buffer 的尾部，另一部分信息在这块 buffer 的起始位置，一帧数据分成两部分来存储；size0 一定大于 0，当 size1 = 0，时，说明这一帧数据只在 ptr0 所指向的地址；当 size1 > 0 时，说明这一帧数据由两部分组成，第一部分的数据在 ptr0 所指向的地址，第二部分在 ptr1 所指向的地址；

接口定义：

```
typedef struct VencOutputBuffer
```

```
{
    unsigned char *ptr0;
    unsigned int   size0;
    unsigned char *ptr1;
    unsigned int   size1;

    long long          pts;        //unit:us
    int                id;
} VencOutputBuffer;
```

id: buffer 的 id
ptr0: 数据 ptr0 地址
size0: 数据 size0 大小
ptr1: 数据 ptr1 地址
size1: 数据 size1 大小
pts:时间戳

4.2.8. 数据结构：VencBaseConfig

接口说明：配置编码器
附件说明：
接口定义：

```
typedef struct VencBaseConfig{
    unsigned int   input_width;
    unsigned int   input_height;
    unsigned int   dst_width;
    unsigned int   dst_height;
    unsigned int   framerate;
    unsigned int   targetbitrate;
    int    inputformat;
    int     codectype;
    Int     maxKeyInterval;
} VencBaseConfig;
```

input_width：输入的 picture 的宽度；
input_heigh：输入的 picture 的高度；
dst_width：编码输出的 width；（编码器有缩放功能，在 Ax 系列 IC 中缩放不能超过四倍， 例如： input_width = 1280， 则 dst_width 最小为 320， 不能小于 320）
dst_height：编码输出的 height;
framerate：帧率 （例如，如果帧率为 30， 则设置 framerate = 30）
targetbitrate：目标码率，单位为 bps，编码器默认的码率控制为 CBR(恒定码率控制)，例如设置 Targetbitrate = 3*1024*1024，那么编码器的码率在统计时间内将保持在 3 M bps
codectype：编码器的选择，见 VENC_CODEC_TYPE

4.2.9. 数据结构：VencSeqHeader

接口说明：指向编码输出的头信息的指针，在 h264 编码中表示的是 sps 与 pps 信息

接口定义：

```
typedef struct VencSeqHeader{
    unsigned char   *bufptr;
    unsigned int length;
    unsigned int bufsize;
}VencSeqHeader;
```

4.2.10. 数据结构：VencProfileLevel

接口说明：用于设置 profile 和 Level

接口定义：

```
typedef struct VencProfileLevel{
    unsigned int profile;
    unsigned int level;
}VencProfileLevel;
```

profile：配置信息(profileidc)，默认为 baseline

level：级别信息(levelidc)，默认为 31，支持 720P

4.2.11. 数据结构：VencQPrange

接口说明：设置最大量化值和最小量化值

附加说明：当码率控制为 VBR(动态码流控制)的时候使用，用此结构体来设置量化值的范围，当码率控制为 CBR(恒定码率控制)时，不需要外部设置

接口定义：

```
typedef struct VencQPrange{
    unsigned int maxqp;
    unsigned int minqp;
}VencQPrange;
```

4.2.12. 数据结构：VencIntraRefresh

接口说明：预留

接口定义：

```
typedef struct VencIntraRefresh{
    unsigned int irmode;
    unsigned int mbcount;
}VencIntraRefresh;
```

4.2.13. 数据结构：VencTargetBitrate

接口说明：码率控制需要设置的目标码率

接口定义：

```
typedef struct VencTargetBitrate{
    unsigned int target_bitrate;
}VencTargetBitrate;
```

4.2.14. 数据结构：VENC_PIXEL_FMT

接口说明：图像编码格式

附加说明：编码器支持的输入格式，在 A1x 系列芯片和 A20 中只支持 VENC_PIXEL_YUV420 格式（yuv420，uv combied 格式， uv 的排列顺序是先是一个 byte 的 u，然后是一个 byte 的 v），在 A31 与 A23 中支持 VENC_PIXEL_YVU420（u 与 v 的顺序与 VENC_PIXEL_YUV420 相反）与 ARGB（包括以下定义的四种不同的排列顺序）；

接口定义：

```
typedef enum {
    VENC_PIXEL_YUV420,
    VENC_PIXEL_YVU420,
    VENC_PIXEL_ARGB,
    VENC_PIXEL_RGBA,
    VENC_PIXEL_ABGR,
    VENC_PIXEL_BGRA,
    VENC_PIXEL_TILE_32X32,
}VENC_PIXEL_FMT;
```

4.2.15. 数据结构：VENC_CODEC_TYPE

接口说明：编码器类型的选择
附加说明：A1x 系列和 A20 只支持 h264 和 MJPEG 编码，aw1639 支持 VP8 编码；
接口定义：

```
typedef enum {
    VENC_CODEC_H264,
    VENC_CODEC_MJPEG,
    VENC_CODEC_VP8,
}VENC_CODEC_TYPE;
```

4.2.16. 数据结构：__pic_enc_result

接口说明：编码后的返回值
附加说明：成功返回 0，返回负值见下面；
接口定义：

```
typedef enum __PIC_ENC_RESULT
{
    PIC_ENC_ERR_NO_CODED_FRAME = -4, //编码错误
    PIC_ENC_ERR_NO_MEM_SPACE = -3, //没有内存可以分配
    PIC_ENC_ERR_VE_OVERTIME = -2, //VE 编码超时
    PIC_ENC_ERR_VBS_UNOVERDERFLOW = -1, //没有数据进行编码
    PIC_ENC_ERR_NONE = 0, //成功
    PIC_ENC_ERR_
} __pic_enc_result;
```

4.2.17. 数据结构：CROP_INFO

接口说明：裁剪信息
附加说明：
接口定义：

```
typedef struct CROP_INFO
```

```
{
    int is_enable; //开启标志
    RECT_t crop_rect; //裁剪范围
}CROP_INFO;
```

4.3.解码模块函数定义

4.3.1. 函数定义：cedarv_decoder_t* cedarvDecInit(s32 *ret)

参数说明：调用该函数后，ret 小于 0 则失败，可以根据 ret 获取失败的具体原因
函数说明：初始化解码模块
返回值　：返回解码模块句柄
附加说明：通过解码模块句柄可操作解码模块提供出来的接口。见 cedarv_decoder_t

4.3.2. 函数定义：s32 cedarvDecExit(cedarv_decode_t *handle)

参数说明：handle 代表解码模块句柄
函数说明：退出解码模块
返回值　：0 代表成功，-1 代表失败
附加说明：无。

4.3.3. 数据结构：cedarv_decoder_t

接口说明：解码模块句柄
接口定义：

```
typedef struct CEDARV_DECODER    cedarv_decoder_t;
struct CEDARV_DECODER
{
    CDX_VIDEO_STREAM_TYPE video_stream_type;
    void *cedarx_cookie;

    s32 (*open)(cedarv_decoder_t* p);
    s32 (*close)(cedarv_decoder_t* p);
    s32 (*decode)(cedarv_decoder_t* p);
    s32 (*ioctrl)(cedarv_decoder_t* p, u32 cmd, u32 param);
    s32 (*update_data)(cedarv_decoder_t* p, cedarv_stream_data_info_t* info);
    s32 (*display_request)(cedarv_decoder_t* p, cedarv_picture_t* picture);
```

```
        s32 (*display_release)(cedarv_decoder_t* p, u32 frame_index);
        s32 (*picture_ready)(cedarv_decoder_t* p);
        s32 (*display_dump_picture)(cedarv_decoder_t* p, cedarv_picture_t* picture);
        s32 (*set_vstream_info)(cedarv_decoder_t* p, cedarv_stream_info_t* info);
        s32 (*query_quality)(cedarv_decoder_t* p, cedarv_quality_t* vq);
        void (*release_frame_buffer_sem)(void* cookie);
        void (*free_vbs_buffer_sem)(void* cookie);
        s32 (*request_write)(cedarv_decoder_t*p, u32 require_size, u8** buf0, u32*
size0, u8** buf1, u32* size1);
    };
```

s32 (*open)(cedarv_decoder_t* p);
打开解码器，参数为解码模块句柄

s32 (*close)(cedarv_decoder_t* p);
关闭解码器，参数位解码库句柄

s32 (*set_vstream_info)(cedarv_decoder_t* p, cedarv_stream_info_t* info);
设置码流信息，需要在 open 之前设置，见 cedarv_stream_info_t

s32 (*ioctrl)(cedarv_decoder_t* p, u32 cmd, u32 param);
Ioctrl 设置解码器命令，见 CEDARV_IO_COMMAND

s32 (*request_write)(cedarv_decoder_t*p, u32 require_size, u8** buf0, u32* size0, u8** buf1, u32* size1);
从解码器中获取 buffer，用于填充码流，require_size 为码流的长度

s32 (*update_data)(cedarv_decoder_t* p, cedarv_stream_data_info_t* info);
告诉解码器填充后的码流数据信息，见 cedarv_stream_data_info_t

s32 (*decode)(cedarv_decoder_t* p);
解码，返回值见 cedarv_result_e

s32 (*picture_ready)(cedarv_decoder_t* p);
查询是否有解码后的数据，有为 1，没有为 0

s32 (*display_request)(cedarv_decoder_t* p, cedarv_picture_t* picture);
获取解码后的数据，数据存放在 picture 中，成功返回 0，见 cedarv_picture_t

s32 (*display_release)(cedarv_decoder_t* p, u32 frame_index);
释放解码后的帧数据，frame_index 为帧数据中的 id，见 cedarv_picture_t

s32 (*display_dump_picture)(cedarv_decoder_t* p, cedarv_picture_t* picture);
获取当前解码的数据，该函数和 disp_request 的区别在于后者会把数据从 FBM 中移除

s32 (*query_quality)(cedarv_decoder_t* p, cedarv_quality_t* vq);
查询 vbv 中 buffer 信息，见 cedarv_quality_t

void (*release_frame_buffer_sem)(void* cookie);
空函数，预留

void (*free_vbs_buffer_sem)(void* cookie);
空函数，预留

4.3.4. 数据结构：cedarv_stream_info_t

接口说明：输入码流信息
接口定义：

```
typedef struct CEDARV_STREAM_INFORMATION
{
    cedarv_stream_format_e      format;
    cedarv_sub_format_e         sub_format;
    cedarv_container_format_e   container_format;
    u32                         video_width;
    u32                         video_height;
    u32                         frame_rate;
    u32                         frame_duration;
    u32                         aspect_ratio;
    u32                         init_data_len;
    u8*                         init_data;
    u32                         is_pts_correct;
    cedarv_3d_mode_e            _3d_mode;
}cedarv_stream_info_t;
```

format：码流格式，见 cedarv_stream_format_e
sub_format：字幕格式，见 cedarv_sub_format_e
container_format：封装格式，见 cedarv_container_format_e
video_width：视频宽
video_height：视频高
frame_rate：帧率*1000，比如 29.970 帧每秒，frame_rate=29970
frame_duration：一帧开始到一帧结束的时间，us 为单位，不确定请赋值为 0
aspect_ratio：该成员变量用于显示，和解码器无关，请固定传 1000
init_data_len：初始化数据长度，没有赋值为 0

init_data：初始化数据，没有赋值为 0

is_pts_correct：PTS 是否正确，h264 解码需要，0 代表正确，1 代表需要根据帧率计算 pts. 该标志位配合 cedarv_stream_data_info_t 的 flag 标志位可以实现解码库根据帧率计算 pts

4.3.5. 数据结构：CEDARV_IO_COMMAND

接口说明：ioctrl 命令

接口定义：

```
typedef enum CEDARV_IO_COMMAND
{
    CEDARV_COMMAND_PLAY,
    CEDARV_COMMAND_PAUSE,
    CEDARV_COMMAND_FORWARD,
    CEDARV_COMMAND_BACKWARD,
    CEDARV_COMMAND_STOP,
    CEDARV_COMMAND_JUMP,
    CEDARV_COMMAND_ROTATE,
    CEDARV_COMMAND_SET_TOTALMEMSIZE,

    CEDARV_COMMAND_DROP_B_FRAME,
    CEDARV_COMMAND_DISABLE_3D,
    CEDARV_COMMAN_SET_SYS_TIME,
    CEDARV_COMMAND_PREVIEW_MODE,
    CEDARV_COMMAND_RESET,
    CEDARV_COMMAND_SET_MAX_OUTPUT_HEIGHT,
    CEDARV_COMMAND_SET_MAX_OUTPUT_WIDTH,
    CEDARV_COMMAND_GET_STREAM_INFO,
    CEDARV_COMMAND_GET_SEQUENCE_INFO,
    CEDARV_COMMAND_SET_SEQUENCE_INFO,
    CEDARV_COMMAND_SET_STREAM_3D_MODE,
    CEDARV_COMMAND_SET_ANAGLATH_TRANS_MODE,
    CEDARV_COMMAND_OPEN_ANAGLATH_TRANSFROM,
    CEDARV_COMMAND_CLOSE_ANAGLATH_TRANSFROM,
    CEDARV_COMMAND_GET_CHIP_VERSION,
    CEDARV_COMMAND_FLUSH,
    CEDARV_COMMAND_CLOSE_MAF,
    CEDARV_COMMAND_SET_DEMUX_TYPE,
    CEDARV_COMMAND_DECODE_NO_DELAY,
    CEDARV_COMMAND_SET_DYNAMIC_ROTATE_ANGLE,
    CEDARV_COMMAND_DYNAMIC_ROTATE,
    CEDARV_COMMAND_SET_VBV_SIZE,
    CEDARV_COMMAND_SET_PIXEL_FORMAT,
```

```
        CEDARV_COMMAND_MULTI_PIXEL,
        CEDARV_COMMAND_OPEN_YV32_TRANSFROM,
        CEDARV_COMMAND_CLOSE_YV32_TRANSFROM,
        CEDARV_COMMAND_SET_DISPLAYFRAME_REQUESTMODE,
        CEDARV_COMMAND_ENABLE_SCALE_DOWN,
        CEDARV_COMMAND_SET_HORIZON_SCALE_RATIO,
        CEDARV_COMMAND_SET_VERTICAL_SCALE_RATIO,
    }
```

CEDARV_COMMAND_PLAY: 播放，解码器播放状态为 PLAY

CEDARV_COMMAND_PAUSE：预留，现不起作用

CEDARV_COMMAND_FORWARD：快进

CEDARV_COMMAND_BACKWARD：快退

CEDARV_COMMAND_STOP：停止，解码器的播放状态为 STOP

CEDARV_COMMAND_JUMP：跳播，解码器清空缓存，等待 seek 后的数据

CEDARV_COMMAND_ROTATE：旋转，参数 0：不变；1：90；2：180；3：270；4：垂直翻转

CEDARV_COMMAND_SET_TOTALMEMSIZE：设置 VE 预留内存大小，为 0 默认 64M

CEDARV_COMMAND_DROP_B_FRAME：非 0 则解码过时时允许丢弃 B 帧

CEDARV_COMMAND_DISABLE_3D：禁止 3D 模式

CEDARV_COMMAN_SET_SYS_TIME：设置当前系统时间，该设置只有在设置了 CEDARV_COMMAND_DROP_B_FRAME 时起作用，过时会根据当前时间和 PTS 进行判断。

CEDARV_COMMAND_PREVIEW_MODE：预览模式

CEDARV_COMMAND_RESET：重设 VBV，FBM 和解码器

CEDARV_COMMAND_SET_MAX_OUTPUT_HEIGHT：设置最大的输出高度

CEDARV_COMMAND_SET_MAX_OUTPUT_WIDTH：设置最大的输出宽度,注意，设置最大的输出高度和宽度必须要在 open 之前调用，该二条命令可以对输出图像进行 1/2 或者 1/4 的 scale

CEDARV_COMMAND_GET_STREAM_INFO：得到码流信息

CEDARV_COMMAND_GET_SEQUENCE_INFO：预留，不起作用

CEDARV_COMMAND_SET_SEQUENCE_INFO：预留，不起作用

CEDARV_COMMAND_SET_STREAM_3D_MODE：设置 3D 模式

CEDARV_COMMAND_SET_ANAGLATH_TRANS_MODE：预留，不起作用

CEDARV_COMMAND_OPEN_ANAGLATH_TRANSFROM：预留，不起作用

CEDARV_COMMAND_CLOSE_ANAGLATH_TRANSFROM：预留，不起作用

CEDARV_COMMAND_GET_CHIP_VERSION：预留，不起作用

CEDARV_COMMAND_FLUSH：刷寄存器和 FBM

CEDARV_COMMAND_CLOSE_MAF：预留，不起作用

CEDARV_COMMAND_SET_DEMUX_TYPE：预留，不起作用

CEDARV_COMMAND_DECODE_NO_DELAY：解码后不排序，直接输出

CEDARV_COMMAND_SET_DYNAMIC_ROTATE_ANGLE：预留，不起作用

CEDARV_COMMAND_DYNAMIC_ROTATE：设置动态旋转角度

CEDARV_COMMAND_SET_VBV_SIZE：设置 VBV 大小

CEDARV_COMMAND_SET_PIXEL_FORMAT：设置是否支持 YV12 输出，A13 和 A20 不起作用

CEDARV_COMMAND_MULTI_PIXEL：动态改变分辨率，当解码器识别到有分辨率改变的时候，decode 接口会返回 CEDARV_RESULT_MULTI_PIXEL，此时需要调用该 COMMAND.

CEDARV_COMMAND_OPEN_YV32_TRANSFROM：预留，不起作用

CEDARV_COMMAND_CLOSE_YV32_TRANSFROM：预留，不起作用

CEDARV_COMMAND_SET_DISPLAYFRAME_REQUESTMODE：预留，不起作用

CEDARV_COMMAND_ENABLE_SCALE_DOWN：设置是否支持 scale

CEDARV_COMMAND_SET_HORIZON_SCALE_RATIO：设置水平 scale,参数 1 代表 1/2，参数 2 代表 1/4

CEDARV_COMMAND_SET_VERTICAL_SCALE_RATIO：设置垂直 scale,参数 1 代表 1/2，参数 2 代表 1/4，以上三条命令用来设置缩放因子需要在 open 之前调用，没有顺序关系

4.3.6. 数据结构：cedarv_stream_data_info_t

接口说明：码流数据信息

接口定义：

```
typedef struct CEDARV_STREAM_DATA_INFORMATION
{
    u32 flags;
    u32 length;
    u64 pts;
    u32 type;
}cedarv_stream_data_info_t;
```

length ：码流长度

pts ：pts 值

type ：码流类型，见 CDX_VIDEO_STREAM_TYPE

flags ：标志位

CEDARV_FLAG_PTS_VALID	(PTS 是否有效)
CEDARV_FLAG_FIRST_PART	(开始标志位)
CEDARV_FLAG_LAST_PART	(结束标志位)
CEDARV_FLAG_MPEG4_EMPTY_FRAME	(MPEG4 空帧标志位)
CEDARV_FLAG_DECODE_NO_DELAY	(解码后不排序，直接输出)
CEDARV_FLAG_DATA_INVALID	(码流数据无效)

4.3.7. 数据结构：cedarv_picture_t

接口说明：解码后数据信息

附加说明：一般关注 disp_width, disp_height 以及 y,u 即可

接口定义：

```
typedef struct CEDARV_PICTURE_INFORMATION
{
```

```
    u32                         id;         //外部设置，解码器不关注
    u32                         width;      //解码后数据宽
    u32                         height;     //解码后数据高
    u32                         top_offset;       //预留，不用
    u32                         left_offset;      //预留，不用
    u32                         display_width;    //显示宽度
    u32                         display_height;   //显示高度
    u32                         store_width;      //预留，不用
    u32                         store_height;     //预留，不用
    u8                          rotate_angle;     //旋转角度
    u8                          horizontal_scale_ratio; //水平缩放大小
    u8                          vertical_scale_ratio;   //垂直缩放大小
    u32                         frame_rate;             //帧率
    u32                         aspect_ratio;           //宽高比，不用
    u32                         pict_prop;
    u8                          is_progressive;         //是否是交错
    u8                          top_field_first;        //预留，不用
    u8                          repeat_top_field;           //预留，不用
    u8                          repeat_bottom_field;        //预留，不用
    cedarv_pixel_format_e       pixel_format; //输出格式，见 cedarv_pixel_format_e
    u64                         pts;                        //pts
    u64                         pcr;                        //预留，不用
    cedarv_3d_mode_e            _3d_mode;                   //3D 模式
    cedarv_anaglath_trans_mode_e anaglath_transform_mode; //预留，不用

    u32                         size_y;                     //y 数据大小
    u32                         size_u;                     //u 数据大小
    u32                         size_v;                     //v 数据大小
    u32                         size_alpha;                 //alpha 数据大小

//这里需要注意，y,u,v 的数据和 cedarv_pixel_format_e 有关
    u8*                         y;          //y 数据，  物理地址
    u8*                         u;          //u 数据，物理地址
    u8*                         v;          //v 数据，物理地址

    u8*                         alpha;                      //alpha 数据
    u32                         size_y2;                //y2 大小，双码流数据使用
    u32                         size_u2;                //u2  大小，双码流数据使用
    u32                         size_v2;                //v2  大小，双码流数据使用
    u32                         size_alpha2;            //alpha2  大小，双码流数据使用
    u8 *                        y2;                     //y2  数据，双码流数据使用
    u8 *                        u2;                     //u2 数据，双码流数据使用
```

```
    u8 *                    v2;                 //v2 数据，双码流数据使用
    u8 *                    alpha2;             //alpha2 数据，双码流数据使用

    u32                     display_3d_mode; //预留，不用
    u32                     flag_addr;          //预留，不用
    u32                     flag_stride;        //预留，不用
    u8                      maf_valid;          //预留，不用
    u8                      pre_frame_valid;    //预留，不用
}cedarv_picture_t;
```

4.3.8. 数据结构：cedarv_quality_t

接口说明：获取 VBV 信息

接口定义：

```
typedef struct CEDARV_QUALITY
{
    u32 vbv_buffer_usage;       //*vbv 使用比例;
    u32 frame_num_in_vbv;       //*vbv 中存在的帧数;
}cedarv_quality_t;
```

4.3.9. 数据结构：cedarv_result_e

接口说明：解码返回值

附加说明：注意，**返回 0 说明已经丢帧**；正常解码成功返回 **1，3**；返回 5 说明需要送数据，返回 6 说明分辨率有改变，返回负值说明解码错误。

接口定义：

```
typedef enum CEDARV_RESULT
{
    CEDARV_RESULT_OK                    = 0x0,  //丢帧
    CEDARV_RESULT_FRAME_DECODED         = 0x1,  //已经解码好一帧
    CEDARV_RESULT_KEYFRAME_DECODED = 0x3,  //已经解码好一个关键帧
    CEDARV_RESULT_NO_FRAME_BUFFER   = 0x4,  //没有帧 buffer 可用
    CEDARV_RESULT_NO_BITSTREAM          = 0x5,  //没有码流可以解
    CEDARV_RESULT_MULTI_PIXEL           = 0x6,  //分辨率有改变
    CEDARV_RESULT_ERR_FAIL              = -1,   //解码错误
    CEDARV_RESULT_ERR_INVALID_PARAM = -2,   //参数错误
    CEDARV_RESULT_ERR_INVALID_STREAM = -3,  //码流数据错误
    CEDARV_RESULT_ERR_NO_MEMORY     = -4,   //内存分配失败
```

```
    CEDARV_RESULT_ERR_UNSUPPORTED    = -5,      //不支持码流
}cedarv_result_e;
```

4.3.10. 数据结构：cedarv_pixel_format_e

接口说明：解码数据格式

附加说明：解码输出一般为 CEDARV_PIXEL_FORMAT_MB_UV_COMBINE_YUV420 和 CEDARV_PIXEL_FORMAT_MB_UV_COMBINE_YUV422，A31 以及后续芯片可以直接解码为 CEDARV_PIXEL_FORMAT_PLANNER_YUV420 和 CEDARV_PIXEL_FORMAT_PLANNER_YVU420 格式。

CEDARV_PIXEL_FORMAT_MB_UV_COMBINE_YUV420 的特点是：

4：2：0 宏块格式，Y 地址由 cedarv_picture_t 中的*y 给出，UV 是混合的，由 cedarv_picture_t 中的*u 给出，cedarv_picture_t 中的*v 为 0

CEDARV_PIXEL_FORMAT_MB_UV_COMBINE_YUV422 的特点是：

同 CEDARV_PIXEL_FORMAT_MB_UV_COMBINE_YUV420，但是是 4：2：2 格式

接口定义：

```
typedef enum CEDARV_PIXEL_FORMAT
{
    CEDARV_PIXEL_FORMAT_1BPP          = 0x0,
    CEDARV_PIXEL_FORMAT_2BPP          = 0x1,
    CEDARV_PIXEL_FORMAT_4BPP          = 0x2,
    CEDARV_PIXEL_FORMAT_8BPP          = 0x3,
    CEDARV_PIXEL_FORMAT_RGB655        = 0x4,
    CEDARV_PIXEL_FORMAT_RGB565        = 0x5,
    CEDARV_PIXEL_FORMAT_RGB556        = 0x6,
    CEDARV_PIXEL_FORMAT_ARGB1555      = 0x7,
    CEDARV_PIXEL_FORMAT_RGBA5551      = 0x8,
    CEDARV_PIXEL_FORMAT_RGB888        = 0x9,
    CEDARV_PIXEL_FORMAT_ARGB8888      = 0xa,
    CEDARV_PIXEL_FORMAT_YUV444        = 0xb,
    CEDARV_PIXEL_FORMAT_YUV422        = 0xc,
    CEDARV_PIXEL_FORMAT_YUV420        = 0xd,
    CEDARV_PIXEL_FORMAT_YUV411        = 0xe,
    CEDARV_PIXEL_FORMAT_CSIRGB        = 0xf,
    CEDARV_PIXEL_FORMAT_MB_UV_COMBINE_YUV420    = 0x10,
    CEDARV_PIXEL_FORMAT_MB_UV_COMBINE_YUV422    = 0x11,
    CEDARV_PIXEL_FORMAT_MB_UV_COMBINE_YUV411    = 0x12,
    CEDARV_PIXEL_FORMAT_PLANNER_YUV420          = 0x13,
```

```
        CEDARV_PIXEL_FORMAT_PLANNER_YVU420          = 0x14
    }cedarv_pixel_format_e;

    #define                                 CEDARV_PIXEL_FORMAT_AW_YUV420
CEDARV_PIXEL_FORMAT_MB_UV_COMBINE_YUV420

    #define                                 CEDARV_PIXEL_FORMAT_AW_YUV422
CEDARV_PIXEL_FORMAT_MB_UV_COMBINE_YUV422

    #define                                 CEDARV_PIXEL_FORMAT_AW_YUV411
CEDARV_PIXEL_FORMAT_MB_UV_COMBINE_YUV411
```

4.3.11. 数据结构：cedarv_stream_format_e

接口说明：码流格式类型

接口定义：

```
    typedef enum CEDARV_STREAM_FORMAT
    {
        CEDARV_STREAM_FORMAT_UNKNOW,
        CEDARV_STREAM_FORMAT_MPEG2,
        CEDARV_STREAM_FORMAT_MPEG4,
        CEDARV_STREAM_FORMAT_REALVIDEO,
        CEDARV_STREAM_FORMAT_H264,
        CEDARV_STREAM_FORMAT_VC1,
        CEDARV_STREAM_FORMAT_AVS,
        CEDARV_STREAM_FORMAT_MJPEG,
        CEDARV_STREAM_FORMAT_VP8,
        CEDARV_STREAM_FORMAT_NETWORK
    }cedarv_stream_format_e;
```

4.3.12. 数据结构：CDX_VIDEO_STREAM_TYPE

接口说明：主码流还是从码流
附加说明：对应一些特殊格式的文件，有主码流和从码流，一般文件只有主码流。
接口定义：

```
typedef enum CDX_DECODE_VIDEO_STREAM_TYPE
{
    CDX_VIDEO_STREAM_MAJOR = 0,
    CDX_VIDEO_STREAM_MINOR,
    CDX_VIDEO_STREAM_NONE,
}CDX_VIDEO_STREAM_TYPE;
```

5. Demo

5.1. 编码 Demo

5.1.1. 目录结构

```
.
├── main.c
└── Makefile
```

5.1.2. 代码片段分析

```
int main()
{
    VencBaseConfig base_cfg;
    VencInputBuffer input_buffer;
    VencOutputBuffer output_buffer;
    VencAllocateBufferParam alloc_parm;
    VencSeqHeader header_data;
    int err = 0;

    // init base config param
```

```
    base_cfg.codectype = VENC_CODEC_H264;
    base_cfg.framerate = 30;
    base_cfg.input_width = mwidth;
    base_cfg.input_height= mheight;
    base_cfg.dst_width = mwidth;
    base_cfg.dst_height = mheight;
    base_cfg.maxKeyInterval = 25;
    base_cfg.inputformat = VENC_PIXEL_YUV420; //uv combined
    base_cfg.targetbitrate = 2*1024*1024;

    // init allocate param
    alloc_parm.buffernum = 4;

    //初始化 VE
    cedarx_hardware_init(0);

    cedarv_encoder_t *venc_device = cedarvEncInit();//初始化解码器
    venc_device->ioctrl(venc_device, VENC_CMD_BASE_CONFIG, &base_cfg);
    venc_device->ioctrl(venc_device,VENC_CMD_ALLOCATE_INPUT_BUFFER,
&alloc_parm);
    venc_device->ioctrl(venc_device, VENC_CMD_OPEN, 0);
    venc_device->ioctrl(venc_device, VENC_CMD_HEADER_DATA, &header_data);

    //数据源是从文件中获取
    test_file = fopen("/mnt/in.yuv", "r");
    if(test_file == NULL) {
        LOGE("open file error");
        return 0;
    }

    out_file = fopen("/mnt/out.h264", "wb");
    if(out_file == NULL) {
        LOGE("open file error");
        fclose(test_file);
        return 0;
    }

    fwrite(header_data.bufptr, 1, header_data.length, out_file);
    mstart = 1;

    // 创建取数据线程
    err = pthread_create(&thread_source_id, NULL, SourceThread, venc_device);
```

```
    if (err || !thread_source_id) {
        LOGE("Create thread_source_id fail !\n");
    }

    // 创建编码线程
    err = pthread_create(&thread_enc_id, NULL, EncoderThread, venc_device);
    if (err || !thread_enc_id) {
        LOGE("Create thread_enc_id fail !\n");
    }

    if(thread_source_id !=0) {
        pthread_join(thread_source_id,NULL);
    }

    if(thread_enc_id !=0) {
        pthread_join(thread_enc_id,NULL);
    }

    venc_device->ioctrl(venc_device, VENC_CMD_CLOSE, 0);
    cedarvEncExit(venc_device);
    venc_device = NULL;

    cedarx_hardware_exit(0);

    if(test_file) {
        fclose(test_file);
    }

    if(out_file) {
        fclose(out_file);
    }

    return 0;
}
```

5.2.解码 Demo

5.2.1. 目录结构

```
.
├── file_parser
│   ├── pmp.c
│   ├── pmp_ctrl.c
│   ├── pmp_ctrl.h
│   ├── pmp_ctrl.o
│   ├── pmp.h
│   └── pmp.o
├── main.c
├── main.o
├── Makefile
└── stream_parser
    ├── stream_ctrl.c
    └── stream_ctrl.h
```

5.2.2. 代码片段分析

```
int main(int argc, char** argv)
{
    int ret;
    pthread_t t0;

    cedarx_hardware_init(0);

    pthread_create(&t0, NULL, decode_thread, (void*)0);

    pthread_join(t0, (void**)&ret);

    cedarx_hardware_exit(0);

    return 0;
}

void* decode_thread(void* param)
```

```
{
    ...
    //打开 parser 去解析输入流或者文件
    ret = OpenMediaFile(&parser, file_path);

    ...

    //获取解码器句柄
    pthread_mutex_lock(&gDecoderMutex);
    decoder = cedarvDecInit(&ret);
    pthread_mutex_unlock(&gDecoderMutex);

    ...

    //设置码流信息
    decoder->set_vstream_info(decoder, &stream_info);

    //*打开解码器
    pthread_mutex_lock(&gDecoderMutex);
    ret = decoder->open(decoder);
    pthread_mutex_unlock(&gDecoderMutex);

    ...

    //开始工作
    pthread_mutex_lock(&gDecoderMutex);
    decoder->ioctrl(decoder, CEDARV_COMMAND_PLAY, 0);
    pthread_mutex_unlock(&gDecoderMutex);

    do
    {
        ...

        ...

        if(pkt_type == VIDEO_PACKET_TYPE)
        {
            ...
            //获取输入 buffer
            ret = decoder->request_write(decoder, pkt_length, &buf0, &bufsize0,
&buf1, &bufsize1);
            ...
```

```
            //读数据到 buffer
            GetChunkData(parser, buf0, bufsize0, buf1, bufsize1, &data_info);

            //设置数据到解码器
            decoder->update_data(decoder, &data_info);

            //开始解码
            pthread_mutex_lock(&gDecoderMutex);
            ret = decoder->decode(decoder);
            pthread_mutex_unlock(&gDecoderMutex);

            //解码失败
            if(ret == CEDARV_RESULT_ERR_NO_MEMORY || ret
== CEDARV_RESULT_ERR_UNSUPPORTED)
            {
                printf("bit stream is unsupported.\n");
                break;
            }

            //获取解码后的数据
            ret = decoder->display_request(decoder, &picture);
            if(ret == 0)
            {
                ...

                //释放数据所占用的 buffer
                decoder->display_release(decoder, picture.id);
            }
        }
        else
        {
            //* skip audio or other media packets.
            SkipChunkData(parser);
        }

    }while(1);

    //* 停止解码器
    pthread_mutex_lock(&gDecoderMutex);
    decoder->ioctrl(decoder, CEDARV_COMMAND_STOP, 0);
    pthread_mutex_unlock(&gDecoderMutex);
```

```
    //关闭解码器并退出 VE
    pthread_mutex_lock(&gDecoderMutex);
    decoder->close(decoder);
    cedarvDecExit(decoder);
    pthread_mutex_unlock(&gDecoderMutex);

    ...
    return (void*)0;
}
```

6. Declaration

This document is the original work and copyrighted property of Allwinner Technology ("Allwinner"). Reproduction in whole or in part must obtain the written approval of Allwinner and give clear acknowledgement to the copyright owner.

The information furnished by Allwinner is believed to be accurate and reliable. Allwinner reserves the right to make changes in circuit design and/or specifications at any time without notice. Allwinner does not assume any responsibility and liability for its use. Nor for any infringements of patents or other rights of the third parties which may result from its use. No license is granted by implication or otherwise under any patent or patent rights of Allwinner. This datasheet neither states nor implies warranty of any kind, including fitness for any particular application.