Rockchip Android Libhwjpeg Interface Introduction

ID: RK-SM-YF-E19

Release Version: V1.0.0

Release Date: 2024-08-20

Security Level: □Secret □Internal □Public ■Public

Rockchip Electronics Co., Ltd.

No.18 Building, A District, No.89, software Boulevard Fuzhou, Fujian,PRC

Website: www.rock-chips.com

Customer service Tel: +86-4007-700-590

Customer service Fax: +86-591-83951833

Customer service e-Mail: fae@rock-chips.com

Preface

Overview

This document primarily introduces the interfaces and usage methods of the JPEG hardware codec package library libhwjpeg. Developers can integrate the hardware-based JPEG encoding and decoding functionality into their applications according to the interface descriptions provided in this document.

Chip Name	Kernel Version
Supports all chipsets	Linux-4.19, Linux-5.10

Intended Audience

This document (this guide) is mainly intended for

Technical Support Engineer

Software Development Engineer

Revision History

Version	Author	Date	Change Description
V1.0.0	Chen Jinsen	2024-08-20	Initial version release

Contents

---

1. Overview

The `libhwjpeg` library is used to support hardware JPEG encoding and decoding on Rockchip platforms, encapsulating the JPEG encoding and decoding logic of the platform's MPP (Media Process Platform) library.

The MpiJpegEncoder class encapsulates hardware encoding-related interfaces, while the MpiJpegDecoder class encapsulates hardware decoding-related interfaces, supporting the decoding of images or MJPEG streams.

The project contains main directories:

```
- inc: libhwjpeg header file
- src: libhwjpeg implementation code
- test: libhwjpeg test instance
```

> The project code is organized using bp files and can be directly compiled and used in the Android SDK environment.

2. MpiJpegDecoder

The MpiJpegDecoder class is an encapsulation of platform-specific JPEG hardware decoding. It supports input of JPG images and MJPEG streams, and provides both synchronous and asynchronous decoding methods.

```
- decodePacket(char* data, size_t size, OutputFrame_t *frameOut);
- decodeFile(const char *inputFile, const char *outputFile);
```

decodePacket & decodeFile are synchronous decoding methods. Synchronous decoding methods are simple to use and block until the decoding output is available. OutputFrame_t is the encapsulation of decoding output, which includes information such as the width, height, physical address, and virtual address of the output frame.

```
- sendpacket(char* data, size_t size);
- getoutframe(OutputFrame_t *frameOut);
```

sendpacket & getoutframe are used in conjunction to implement asynchronous decoding output. The application launches two threads, one thread sends input via sendpacket, and the other thread asynchronously retrieves output via getoutframe.

Notes:

1. Decoding defaults to output RAW NV12 data
2. Hardware decoder characteristics, the decoding output YUV is aligned by default. When actually used before display, the real image must be cropped from the buffer based on its virtual width and virtual height, otherwise, issues like green edges may occur. libhwjpeg provides a channel to implement the internal self-cropping, through setting the property - setprop hwjpeg_dec_debug 0x10.
3. The OutFrame buffer encapsulates the decoding output dmaBuffer and is reused internally within the decoding library. After the decoding display is completed, the memory is released using deinitOutputFrame.
4. By default, the output buffer uses the internally allocated buffer rotation pool, but it also allows external transmission of dmaBuffer fd, the additionally transmitted dmaBuffer fd is passed via the parameter of the decodePacket function OutputFrame_t->outputPhyAddr.

Decoding Usage Example:

```
    MpiJpegDecoder decoder;
    MpiJpegDecoder::OutputFrame_t frameOut;

    memset(&frameOut, 0, sizeof(frameOut));

    err = decoder.prepareDecoder();
    if (!err) {
        ALOGE("failed to prepare JPEG decoder");
        goto cleanUp;
    }

    err = decoder.decodePacket(data, size, &frameOut);
    if (!err) {
        ALOGE("failed to decode packet");
        goto cleanUp;
    }

    /* TODO - Get diaplay for the frameOut.
     * - frame address: frameOut.MemVirAddr
     * - frame size: frameOut.OutputSize */

    /* output buffer count within limits, so release frame buffer if one
       frame has been display successfully. */
    decoder.deinitOutputFrame(&frameOut);

    decoder.flushBuffer();
```

3. MpiJpegEncoder

The MpiJpegEncoder class is an encapsulation of platform-specific JPEG hardware encoding. The primary interfaces currently provided include:

```
- encodeFrame(char *data, OutputPacket_t *packetOut);
- encodeFile(const char *inputFile, const char *outputFile);
```

encodeFrame & encodeFile are synchronous blocking encoding methods. OutputPacket_t is the encapsulation for encoding output, containing memory address information of the output data.

```
- encode(EncInInfo *inInfo, EncOutInfo *outInfo);
```

The encode interface is a encoding scheme designed for the camera HAL, which receives user-provided input/output dmaBuffer fd. The output JPEG includes encoding thumbnails, APP1 EXIF header information, and other related information.

Notes:

1. OutputPacket_t is the encapsulation for encoding output. The buffer is reused internally within the encoding library and should be released using deinitOutputPacket after the encoding process is completed.

Encoding Usage Example:

```
    MpiJpegEncoder encoder;
    MpiJpegEncoder::OutputPacket_t pktOut;

    memset(&pktOut, 0, sizeof(pktOut));

    err = encoder.prepareEncoder();
    if (!err) {
        ALOGE("failed to prepare JPEG encoder");
        goto cleanUp;
    }

    err = encoder.updateEncodeCfg(
            720 /*width*/, 1080 /*height*/, MpiJpegEncoder::INPUT_FMT_YUV420SP);
    if (!err) {
        ALOGE("failed to update encode config");
        goto cleanUp;
    }

    err = encoder.encodeFrame(data, &pktOut);
    if (!err) {
        ALOGE("failed to encode packet");
        goto cleanUp;
    }

    /* TODO - Get diaplay for the PacketOut.
     * - Pakcet address: pktOut.data
     * - Pakcet size: pktOut.size */

    /* output buffer count within limits, so release frame buffer if one
       frame has been display successful. */
    encoder.deinitOutputPacket(&pktOut);

    encoder.flushBuffer();
```